AFIPS CONFERENCE PROCEEDINGS

1977 NATIONAL COMPUTER CONFERENCE

June 13-16, 1977

Dallas, Texas

ROBERT R. KORFHAGE
Editor and Program
Chairman

PORTIA ISAACSON
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 1977 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 PORTIA ISAACSON
Conference Chairman
The University of Texas at Dallas
Richardson, Texas

To chair the National Computer Conference is a rare opportunity to contribute to one's profession. I could never have passed it by. As is common with opportunities, however, this one has not been without its challenges. Those challenges have been well met by the 1977 NCC Steering Committee. The recruitment of that team of professionals has been by far my greatest contribution to the 1977 National Computer Conference. Their dedication and enthusiasm for the enormous task we faced surpassed even my expectations.

The National Computer Conference is unique in our industry—not just because it is our largest conference—but because it does not restrict itself to a narrow view of the industry; each year, it bends and reaches to point to new and different directions. One of the most important challenges of each steering committee, I believe, is to determine the directions of reach for the NCC in a particular year, while retaining the broad base of the conference.

Thus each NCC is unique, bearing the imprint of the particular group of people that brought it to fruition. The 1977 NCC is no exception. While reinforcing the NCC tradition of providing a broad technical program, the 1977 NCC has added its own innovations. Among the unique features of this conference are the program of Professional Seminars, the Programming Contest, and the Personal Computing Fair and Exposition.

A National Computer Conference is an enormous project. Without the hundreds of individual volunteers and many supporting companies, we could not have such a conference. We must especially thank our Conference Steering Committee, which is responsible for the design and much of the implementation of the conference. AFIPS Headquarters' staff has provided top-notch support, with an obviously successful exhibit sales program, innovative communications plan, and a multitude of other tasks. The NCC Board and NCC Committee have generously donated their time to provide the overall guidance necessary to make the NCC happen.

It is an especially difficult task to assemble a program of technical sessions with sufficient breadth of coverage to be worthy of a conference that calls itself the National Computer Conference. However, as difficult as it is, Dr. Robert Korfhage has been more than equal to the challenge as evidenced by these proceedings.

The National Computer Conferences have established a strong tradition of excellence. We are proud to have contributed to that tradition.

Introduction

by ROBERT R. KORFHAGE
Program Chairman
Southern Methodist University
Dallas, Texas

If you are ever tempted to project ahead a quarter century or so, you should first look back an equal span of time, and observe the changes that have taken place, particularly within the world of computing, in that time. Each National Computer Conference stands at this juncture between an existing past, and an even more exciting future. Thus the Conference not only represents computing as it has come to be, but also projects an image of where we are going.

Where *are* we going? For better or worse, computing is now involved in every type of human activity. Thus we take it as the mission of the 1977 NCC Program to reflect this involvement and to point out promising directions. Gone are the "good old days" of the esoteric, highly technical major conference. In the first place, the field has expanded far too much for anyone to have a sound and deep technical knowledge of all aspects. In the second place, many highly specialized conferences now exist at which various subsets of computer practitioners can discuss the intricacies of their particular area of interest. Thus we see the National Computer Conference as presenting a broader view of computing—in a sense a "world's fair" of computing, with presentations ranging from tutorials and seminars aimed at those who have just heard of a given topic, to technical presentations that will hold the interest of those more deeply involved. We hope that you will see the conference in this way. Not everything will interest you. Select what you will—but read the papers, attend the sessions, participate, and enjoy four days of the best in computing!

Roughly half of the program this year is technical in nature, with the major topics being computer architecture and database management. This portion of the program is well represented by papers in the Proceedings. The less technical portion of the program is less heavily represented within these pages. This is due to the fact that roughly one third of this portion consists of "management briefings"—presentations without formal written papers, that are aimed at present and aspiring managers in the computer field. However, within this half of the program there are also papers on management problems, on a variety of interesting applications, and on the exploding field of personal computing.

Putting together a program for this large a conference involves more than a thousand people. Most evident of course are the speakers and the session chairmen. But we tend to forget the other authors—both the coauthors who never appear "on stage," and the authors whose work we have not been able to use, despite the efforts that they have put in. Hundreds of referees devoted many hours to reading all of these papers. If, indeed, one benefits in proportion to the effort put in, then the referees will gain much from the Conference.

Finally, I would like very specially and personally to acknowledge the work done by the small core of people who have been involved in the entire program production—the Program Committee, my staff in Dallas, and the AFIPS staff in Montvale. When we started this effort I told the Program Committee that they would get the credit and I would get the blame—and that's the way it is. Because of the enthusiasm and hard work of this small core, the whole production has gone very smoothly. The few difficulties that we have had are indeed traceable directly to my desk.

CONTENTS

DATA BASE ADMINISTRATION

COMPUTER SYSTEMS ARCHITECTURE

COMPUTER SECURITY TECHNIQUES

COMPUTER GRAPHICS

CLINICAL APPLICATIONS OF THE COMPUTER

DATA MODELS AND THEIR APPLICATIONS

COMPUTER SECURITY RISK ASSESSMENT

COMPUTER-BASED INFORMATION SYSTEMS FOR THE SMALL FIRM

SELF-ORGANIZING/SELF-OPTIMIZING DATA BASES

SELECTION METHODS FOR A FAMILY OF COMPUTER ARCHITECTURES

MICROPROCESSOR ARCHITECTURES

SOFTWARE MANAGEMENT—PLANNING FOR A NEW SOFTWARE DEVELOPMENT PROJECT

COMPUTER SYSTEMS IN HEALTH CARE DELIVERY AND MEDICAL LABORATORIES

ADVANCED CONCEPTS IN DATA BASE MANAGEMENT

EDUCATION FOR MEDICAL INFORMATION SCIENCE

COMPUTER HARDWARE DESIGN

OUTPUT—RESULTS AND RHETORIC

DATA STRUCTURES

MAN-MACHINE INTERFACE

COMPUTER TECHNOLOGY IN THE INFORMATION/LIBRARY FIELD

APPLICATIONS OF COMPUTER NETWORKS

COMPUTER SYSTEMS—A GLOBAL VIEW

CONTINUING EDUCATION FOR THE COMPUTER SPECIALIST

PERSONAL COMPUTING—HISTORY AND FORESEEABLE FUTURE

DATA BASE STRUCTURE AND ORGANIZATION

SOFTWARE VALIDATION

COMMUNICATION AND EDUCATION

DATA BASE APPLICATIONS

INTERNATIONAL NETWORKS AND PACKET-SWITCHING

PROGRAMMING LANGUAGES—HIGH-LEVEL PROGRAMMING FOR LOW-LEVEL MACHINES

Cm*—A MULTI-MICROPROCESSOR COMPUTER SYSTEM

PROGRAMMING LANGUAGE THEORY

ARCHITECTURE FOR DATA BASE MANAGEMENT

FAULT-TOLERANT COMPUTING—I

REMOTE TERMINAL EMULATION

SOFTWARE FOR USERS AND MANAGERS

ADP ACQUISITIONS—CONTRACTING, FINANCING, INVESTING, THIRD-PARTY MAINTENANCE

APPLICATION OF DIVERSE TECHNOLOGY AND ITS RELATION TO OCR SYSTEMS

TEXT PROCESSING SYSTEMS

FAULT-TOLERANT COMPUTING II

PERFORMANCE EVALUATION

DISCRETE MATHEMATICAL MODELS

DATA BASE—EXTENSIONS AND APPLICATIONS

APPLICATIONS OF COMPUTER SYSTEMS

INFORMATION SYSTEMS—PERFORMANCE ORIENTED DESIGN AND EVALUATION

COMPUTER ARCHITECTURE DESIGN

SPECIAL MEMORY ARCHITECTURES

THE COMPUTER IN MANAGEMENT AND BUSINESS

SIMULATION METHODS

PERSONAL COMPUTING SYSTEMS

TRENDS IN COMPUTER STORES

Data base administration—Classical pattern, some experiences and trends

by JEAN-PAUL DE BLASIS
Centre d'Enseignement Superieur des Affaires
Jouy-en-Josas, France

and

THOMAS H. JOHNSON
The Wharton School
University of Pennsylvania
Philadelphia, Pennsylvania

ABSTRACT

This three part paper covers the growth and classical patterns of data base administration along with a survey of some currently practicing data base administrators. As a result, some trends and evolutions of the data base administrator position are presented.

After a brief historical introduction, the paper first sets forth basic definitions and organizational considerations for the DBA function. Interfaces, both internal and external are defined and the associated problems are discussed. The functions and responsibilities of the DBA are presented along with the tools available for carrying them out as recommended by various committees including CODASYL, GUIDE/SHARE and other reports.

Then, a survey of organizations utilizing data base administrators is presented. The survey looks at organizations of varying sizes and commitments to data base technology. The results of the survey try to show how the previous recommendations are reflected in reality. An attempt to rationalize the plans and the actual status of the DBA in an organization is carried out.

Finally, from the survey results and from projections provided by the data base administrators themselves, some trends in the evolution of their functions are outlined. In light of these projections an attempt is made to review some of the recommendations currently put forward.

INTRODUCTION

Does the data base administrator portrayed in the current literature exist in practice? What are the differences, and what are the trends for the DBA function? This paper attempts to answer these questions by briefly reviewing the literature, discussing actual practice, and outlining the trends in the data base administrator's role.

We developed a central thesis from pre-screening interviews and literature search to focus our research on the data base administrator. In testing this thesis, we uncovered areas for future research and application emphasis. After posing the central problem, we reviewed the definitions and functions of the DBA as defined in current literature to frame our discussion. Next, we focused on the current state of the DBA, summarizing interviews of over twenty practitioners. We present results in the form of evolutionary trends occurring in this area and conclude with opportunities for research suitable to the support of practitioners.

The central thesis is:

> The data base administrator now, and in the future, is an individual who performs the function of planning, designing, operating and controlling the data base of an organization at both the policy and operational level.

In order to frame the results from our study, we must first establish the definition, function and organization proposed for data base administrators. We have synthesized proposals and reports from both committees studying the area and authors publishing in the area. Committees include GUIDE-SHARE,[12,13] CODASYL,[6,7] and ANSI-SPARC.[1] They are very active in the DBA area, especially GUIDE-SHARE which has advocated recent expansions of the DBA role to data administrator and put forward techniques for focusing on data resource policy.

DEFINITIONS AND FUNCTIONS

Definitions of the data base administrator's function

The individual introduced in both Codasyl and Share Reports, called the data base administrator, is a person meant to solve many of the problems in file integration and in maintenance of any organizational data base. He is supposed to be familiar with computerized systems, with data management, and above all, with every aspect of the corporate data base. The DBA function has been previously defined as a human function, responsible for the coordination of all data related activities.[4,17] The following is an extended definition specifying some of the generic areas of the DBA's functional responsibility. The data base administrator is the individual providing the coordination, perspective, and administration of the data base by exercising specific responsibilities. His responsibilities should include the definition, organization, protection, efficiency, and documentation of the data base. He should also be responsible for defining the rules by which data is to be accessed and stored.

To decide where to place the DBA involves trade-offs almost always because of the distinct organizational characteristics in which every systems environment operates. The answer provided by the literature is generally that the DBA should report to the highest full-time information systems manager.[17,20,21] Practically, however, there are very few DBA's who are aligned this way, as shown by some field studies.[10,14,19]

Interfaces

The data base administrator generally interfaces directly with three groups within the organization. The systems administration group is the first. This group is concerned with the operation, maintenance and performance of the information systems equipment. This includes the performance of data base management systems, as well as security, recovery and re-start of these systems. They are also concerned about the throughput of the system and the running of production systems in the most effective manner.

Another interface exists between the data base administrator and systems development. Systems development is considered to include the planning, analysis, design, and implementation of application processes. The data base administration would participate in the design effort and make determination of technical and economic feasibility in seeking to satisfy the data requirements of the applications processes. The data requirements of new systems may be satisfied by using data already collected and stored or by extending the definition of the data base to include new data requirements.

In cooperation with users, the data base administrator is supposed to seek to determine what data to collect and store and the criteria to use in validating input data and stored data. It has been suggested that the correctness of data is the responsibility of the user, while the protection of data is the responsibility of the data base administrator.[18] A Diebold Research Report[11] notes a trend towards "placing responsibility for data accuracy, validity, and so forth, in the hands of the user who will be served by the data base." The report suggests the appointment of a "Prime Responsible Authority" (PRA) for each data base from its primary user division. The PRA interfaces with all users on one side and with the data base administrator on the other. In other words, the "prime responsible authority," working in a user division rather than in the data base administrator's organization, would be responsible for content, integrity and use of the data base with respect to all users within the organization, both within and outside of his own using organization.

An Auerbach document[2] lists ten areas of functional responsibility for the data base administrator as it can be seen in Table I. Rather than discussing those ten areas, we shall focus on the unique functions of data base administration, which are: definition, storage and update of data; making the data base available to the using environment; informing and servicing users; maintaining data base integrity; and operations and performance.

Definition, storage, and update of data

The process of data definition begins in response to stated data requirements from the using environment. The first step in definition process is to design the logical data structure, incorporating as much as possible of the natural structure inherent in the data. In a sense, a logical data structure should model selected aspects of the operations and entities as they really exist. This is pointing out that to date very little work has been done on developing formal methodologies to aid in the process of data base design which is really needed though, with perhaps the exception of some current efforts in that direction being in progress. Once the logical data structure is developed, it is formalized in the Data Definition Language (DDL) of the particular data base management being used. Things that the data describe in logical data structure will eventually be stored physically in the computer system. Having defined some data to the system, the next step is to set up the mechanisms to acquire the new data and to bring it into the system.

Finally, the last step in the process is to execute management policies regarding update of data. Even if the value of data diminishes over time, it is not desirable to aggregate older data. An updating mechanism should be set up to store the older in off-line archival files for example.

Establishing data availability

One of the functions of the data administrator is to assist users in their search for data to satisfy their application

TABLE I—Areas of Data Base Administration Responsibility

1. Data Definition
2. Data Base Design and Implementation
3. Data Base Access
4. Data Base Standards Control
5. Documentation
6. Operations
7. Monitoring
8. Data Base Management Systems Enhancements
9. Education
10. Vendor Enhancements

requirements. He should maintain a Data Base Directory (DBD), in which are recorded the record types, and set types currently available to users. The DBD will then be the initial source for information relative to data availability. If it happens some data elements are not available within the confines of the existing data base, the data administrator will arrange the interface with the necessary data sources to satisfy the demands of the users. Such demands, of course, should remain within the cost constraints controlling the user and the data base administrator.

Some additional factors to be included when considering data availability are the following:

1. Present form and location of data.
2. Access techniques to be used.
3. Intended use of data in relation to its present accuracy, completeness and timeliness.
4. Need for modification of data.
5. Present authorizing agent for use of data.
6. Cost of providing the data.

Maintain data base integrity

The protection of the data base is an essential responsibility of the data base administrator. The data base represents a large dollar investment, and data contained in it is vital to all who use it. Alteration, destruction or disclosure of the data base may represent an enormous and irrevocable loss in time and money. Although complete integrity protection is never possible, a high level of protection should be kept as much as possible.

The general problem areas in data base protection are:

1. Data base access and manipulation;
2. Data base integrity;
3. Safe recovery/restart.

Data base documentation

In addition to being an organizer and an administrator of the data in the data base, the DBA is the prime documentarian and educator with regard to the Data Base Management System (DBMS) at his installation, and he should provide for the recording of procedures, standards, guidelines, and data base descriptions for proper use of the data base.[3]

STUDY RESULTS

Over twenty Data Base Administrators' groups have been studied to date. Their companies ranged in size from $3 billion to $20 million in revenues and data bases ranging in size from 100 billion characters to a few hundred thousand characters. Despite the range of application and size of companies, we found remarkable similarities among approaches, problems, and successes. The results are presented here in summary form to support the main thesis. The detailed survey is being published in working paper form.[14]

Recall the primary thesis we set out to prove:

> The data base administrator now and in the future is an individual who performs the functions of planning, designing, operating, and controlling the data base at both policy and operational levels.

Operations vs. policy

Critical in the thesis statement are the dual tasks of operations and policy making. This conflict, we found, is one area where the position statement breaks down. The data base administrator is said to be responsible for the definition, storage, and update of data; making the data base available to the using environment; informing and servicing users; maintaining data base integrity, operations and performance. The data base administrator is also said to be responsible for corporate data base policy; access authorities; definition of data base content and organization; selection of data to collect and store input; criteria to use in validating input; and conflict mediation among users, system designers, and computer operations.[23]

These latter functions required the DBA to be high enough in the organization's management, so that the position has authority to set and enforce policy. The former functions, more operational, required less top management involvement and usually meant a lower position in the organization. Furthermore, the operational dimension caused the DBA to bias the broader policy issues affecting many groups in favor of his own operations.

Evolution of the DBA

As the study began, we uncovered another framework which shed considerable light on the DBA. A definite maturing of the DBA function occurs within an organization. The newer DBA's were often focusing on problems that the more established DBA's had already resolved. DBA's at different stages of development did not articulate the stages that we observed. They related primarily to the current problem and to a continuum they were currently traversing.

Our formulation of these evolutionary steps made observation of DBA's much simpler and more logical. First, it helped separate the progress of a particular DBA group and the progress of the field as a whole. By classifying each DBA group according to their development stage, we could study that group's state relative to the stage and not to the general population. For instance, the DBA just getting started with a three person staff has significantly more support than the DBA in the height of development with three people. We then could ask if the problem definition phase is receiving greater attention instead of observing the gross state of the group.

We outlined five stages in the evolution of the data base administrator function: Introduction, initiation, integration, operation, and maturation.

Introduction generally took the form of either a study group or a manager's individual recommendation and decision to go with a data base management system. Although we did not take much specific data on this phase, most earlier introductions seemed to discover a need for a data base administrator after using a DBMS, while more recent introductions also discovered a need for the DBA to parallel or precede a DBMS.

Next, the initiation of a DBA consists of developing one or two data base systems. As much as total system design or top down approach may be touted, the corporate data base was never built in this phase, it was always started as one or two application data bases. Most often the DBA's speed of success rested on how well these applications were performed (we say speed of success because despite severe negative benefits in some systems, data bases are becoming essential to most business organizations and the question is generally how fast). It is in this stage lasting one or two years that the DBA spends a considerable amount of time and money to establish a base for future growth. Correct choice of system, personnel, applications, etc. are critical here.

Integration, the joining of several systems together into the corporate data base, is a development phase of several years duration. It really never stops, but we have established an (arbitrary) turning point into maturity when the DBA has control of data base's definition, design, access, and standards along with having a large majority of DP application systems on the DBMS. This points out the two maturation phases going on during integration phases. One is the actual integration of data into the data base, the other is the acceptance of the DBA as the group in control, mainly operational control, of the data base.

Maturity sees the focus of effort move from development to operations. It also means a focus of the DBA role as the center of data base systems responsibility from design to operation. Some organizations might believe they have matured with only one or two subsystems operating, but until the DBA is involved from the beginning in all data base work and also has the final power to make the operational decisions, the group has not matured.[22] In fact, during our survey we found very few mature groups.

General experiences

For the study, we outlined several questions to help us establish a more logical pattern to our research. Some questions were posed directly to the subject; others were broken down in more detail and synthesized here.

- Is there a measure of the size of the DBA's organization?
- What are the qualifications needed to fill the DBA position?
- Which of the ten functions does the DBA perform?
- Where does the DBA report in the organization?
- What is the cost of the DBA and where are the economies?
- What is the biggest problem faced by the DBA?

Size of DBA function

The size of the DBA group is proportional to the size of the data base up to a point. With one exception, the DBA group increased with the size of the data base. The increase was not linear, because the high initial cost to support the system causes a rapid growth initially followed by a more gradual but noticeable increase. The largest group interviewed had 14 people in the DBA group; the smallest had one. Economies of specialization and scale took over as the data base grew, and the staff became organized, usually functionally, but sometimes in project teams. We learned that the start-up effort can be substantial. The analyst/programmer training consumed a large portion of the DBA's time during the first two years of operation, but fell drastically after user acceptance. In many cases, training and marketing DBMS went together, so the work load further increased start-up staff size.

Once application systems arrived at an operational level, the group did not shrink because of required data base and program updates and other support functions. Surprisingly, all those interviewed who had been operational for sometime, sighted more man-hours consumed on system problems than data structure updates. As major applications became operational, the role of the DBA staff grew more operational and tended to increase, usually in support personnel.

There was one strategic milestone for staffing which passed quickly in some cases and never in others, and that is commitment to data base technology. With the commitment given, staff size no longer was a problem. Without it, the DBA found mustering resources, like staff, difficult. A few groups were able to overcome this obstacle by performing well despite resource constraints; one group disbanded, several more are not beyond this milestone.

Two final notes: first, staff size is not a measure of the quality of a data base group, but it can be a measure of the data base size. There may be a relationship between the size/complexity of a data base and the size of the DBA group.[14] Next, there appears to be little relationship between the type of DBMS and the size of the DBA group. Though this finding came as a surprise since different technologies would require different staffing, it now seems logical because the technologies are not all that different and the major problems requiring staff are fairly similar. There may also be a measure of technology improvement and staff size if these findings are verified.

What are the requirements for a DBA

It was felt by those interviewed that the primary requirements for a person to be a DBA or join the DBA group were both technical skills and knowledge of the company, administrative capability ranked a weak second.

Sixty percent of the current administrators had lengthy experience within their companies' DP department, not necessarily with DBMS. The others were DBMS experts brought in for the job. The overwhelming majority of the administrator's non-clerical staff were technically trained in DBMS. The large majority of administrators suggested that their replacement should have both technical training and a minimum of two to five years with the company. Their staff members were required to have a strong technology background before being considered for hiring or transfer from an application area being implemented under the DBMS.

The technical skills requirement was even more pronounced in the more mature groups. They were finding a greater demand on the part of analysts and users for more detailed systems assistance. In all of the mature groups, the DBA group found themselves spending a large amount of time in the system support function. Frequently, they were supporting applications programmers as systems programmers because the applications people considered the data base part of the operating system.

Administrative skills were secondary qualifications to three-fourths of the DBA's. A person with a Master's in computer sciences and with company acquired skills is much preferred as a DBA over someone having a Master's in Business with technical skills. It was felt that administrative skills could be learned on the job. This opinion held for the very large DBA groups and the very small ones.

DBA JOB FUNCTION

The DBA function among organizations is remarkably similar in the long run, but start-up situations were crucially different. The actual job descriptions, the 30 percent we were able to see, came directly from the DBA literature.

We were very surprised that generalization about the DBA function could be made between different-size companies using different DBMS products. Each company and package have difficulties unique to itself, but histories, problems, and operational organizations were similar. The literature explained the functional areas for setting up operation, and the DBA's generally read these major sources for advice. The DBA groups we found were primarily operational functions whose objectives were technical support for applications analysts. The organizations tended to grow more support-oriented with time.

Of the ten tasks outlined earlier, implementation, access, DBMS enhancements, education, and vendor enhancements were the five major areas of concentration found in young groups, in the initiation phase. In this phase, data definition and design were not relinquished by the applications analysts. Control, documentation and monitoring were not institutionalized very well in the DBA functions. Operations were almost always delegated to the company's operations department under advisory relationships with the DBA. Often backup recovery was a major headache for the newly initiated data base group until operations had accepted responsibility for its execution.

These young DBA groups were brought in at the last stages of applications development to "make this a data base system." The applications analyst would have done all the work, built the system, and as a last touch, attached the DBMS. Education, in the form of data base techniques and data base design principles, became a major consumer of the DBA group's time. The education served three purposes: sell DBMS as a concept, sell the need for involvement of the DBA early in the application design, and teach the analyst how to use the DBMS.

The evolution towards involvement of the DBA in application design became evident in the large majority of mature systems. The groups generally gained this involvement via user acceptance, not by fiat. As the groups matured, internal controls, documentation and monitoring became a part of daily operation. Backup and security was one area of responsibility serviced from the start. The DBMS generally voided normal file backup operations, yet, the operation was required. The DBA set up procedures for the operations group to follow to back up the DBMS.

Vast differences did occur in initiating the DBA function and bringing it to mature operation. These differences appear in organizational consideration, top level commitment to the DBA's, and interfaces with operations, applications, development and users. There is genuine need for further research in this area to support the start-up and evolution of a DBMS and the DBA, especially relating to these areas.

Where the DBA report in the organization

Data base administrators rarely report to the highest systems' administrator, but more frequently report to the manager of systems analysts.

Current authors in an attempt to improve the DBA position's strength have an extensive list of policy roles for the group to play. These roles do not yet occur in the practical world. Being the mediator and direct advisor to the chief systems administrator on data policy has been recently proposed.[14,19,24] In practice, we have not seen this occurring in the person of the data base administrator. The DBA is most frequently a senior analyst or group leader in applications or systems development organizations. In only two cases did the data base administrator report to the top level of systems management.

We recognize the need for better data base policy in organizations. But from the history of current DBA's, this policy maker would probably be another individual with the DBA serving as an important part of policy implementation. In fact, historically, there generally was an individual or group of individuals who took on this policy role. They instituted or supported the policy to use a DBMS and to initiate the DBA function. The creation of a position to plan and control the use of data resources is one way to institute data base policy. The formalization of a committee is another way to institute policy. Either way, it appears to be a job separate from the DBA's job.

Cost and economies of the DBA

Two facts stood out in our survey in this area. First, the organizations had little idea what the direct cost of the DBMS or the DBA were. Second, the organizations recognized that they required a DBMS and the DBA to economically operate their data processing system.

It is astounding how poorly the cost and economies of DBMS are measured, for that matter it is astounding how poorly the computer resources are measured. In general, an astute applications group within these companies should immediately take advantage of the DBA and attempt to use the data base because the charges for these resources were not accounted for directly. In some cases operations cost, computer run time, was charged back to the user, and served as a deterent to using the data base. But generally the DBA groups were an overhead item, a necessary expense to keep the system operational.

The rush to be "on the data base" did not occur. However, the costs or economies were rarely the reason given for not using the DBMS.

No good data about the economies of these data base systems came out of the study. At most it can be said that they are expensive to build and to run. A DBMS needs a group of experts, the DBA, to run and maintain it in order to keep the unaccounted for costs down to some unknown amount. Organizations are willing to pay some proportion of the budget to support the systems and the staff. And, the organizations who had committed to the use of a DBMS felt they were getting good results from the change.

The major problem stated by the DBA

Administrative problems, or organizational issues surfaced as the most important problems in the great majority of groups studied. After insisting on technical people to staff the group, the DBA manager cited his critical problems occurred in a variety of administrative areas. These problems were often a function of the point of evolution of the group.

The groups just starting up cited top management support, cooperation from user groups, and training or education as their most frequent problems. The more mature groups cited control and coordination problems along with evolving technology difficulties. The younger groups generally felt satisfied with the technology, but felt constrained by the environment, while the older groups although still bothered by administrative problems had technical difficulties as frequently mentioned as organizational ones.

For the DBA in the initiation phase the major problem revolved around either introducing a new technology into the organization or the creation of this cross organization body that the DBA represented. We expected to find these groups were highly critical of the technology or with an extensive list of needed advances.[24] Instead we found general satisfaction with the DBMS product they used and frustration with its users. Sometimes the problems appeared to be the overanxiousness of the DBA who had been mandated to put in an application and was trying to install the total corporate system. More often the reasons were problems with the education and training of others outside the DBA on proper approach. Most typically the young group was seeking the responsibility for design and development instead of just the programming of the DDL or DML. This interface with the users, the establishment of educational programs, the convincing of top management was the slowest and most painful effort, yet generally a successful one.

The mature groups had achieved these responsibilities and were established as a necessary component in any new development. Here we did find complaints about technology: a lack of a data dictionary, or better recovery control, or a solution to the on-line update problem. We also found more administrative control problems surfacing. Their concerns about participating in system change decisions and about data control problems were more pronounced. There was a distinct interest in establishing responsibility for the data but at the same time an unwillingness to take the responsibility. They also expressed concern about control over changes to the data base and its structure. Finally they were searching for people with the qualifications and the ability to work in the DBA group.

CONCLUSION

The DBA is a necessary part of any data base management system effort. No corporation should attempt to form a "corporate data base" without this individual or group to manage its internal operations. The DBA is not however the policy maker that we see in the literature. Corporations do not really view data as a resource nor do they view the DBA as a manager of a valuable corporate resource.

The DBA's we surveyed are highly qualified technical teams or individuals that first bring a new technology to bear on data processing problems and then make the technology operate with the maximum effectiveness possible. They are in a unique position of bridging various applications and have an extraordinarily difficult time bringing the available capabilities to potential users, but they seem to perform this job reasonably well. The DBA group now and in the future needs well qualified technical people who can interface with their users and colleagues. At the same time they need to attract those individuals who have a knowledge of the organization's problems and who can interface with the users from a different perspective.

The DBA is an entity with a future. We see it as a training ground for managers who need a broad view of the company's information system problems, yet who need to have the technical capabilities to manage some user groups. The DBA seems to naturally focus more and more on the technical issues which are within its control and which grow as the position grows. The DBMS technology is being applied to more and more problems in more and more organizations. As the organizations continue to utilize information systems the DBA will provide the data base support necessary, probably on a more expanded technical front.

As far as data base policy is concerned, we see a different individual or group becoming concerned with this problem. The DBA would be the most important implementer of the policy, but not the policy maker. The corporate information managers will need staff or support people who can advise them on policy matters without the vested interest of the DBMS implementation. These policy makers would be concerned with the data resource as one of the corporate resources. They would draw on the DBA group for talent and individuals to move into the advisory wing, but this would probably not be the same group.

The DBA will be involved with technology growth problems as well as the growth of his data base. The systems studied showed that most DBMS systems operate in batch oriented systems. Of those that don't, only a few allow sophisticated on-line update and access to the data base. Most of the systems were not using the operational data for more expanded MIS purposes, either. We expect the demand for on-line update, and the demand for more MIS applications to grow along with the introduction of more sophisticated data base techniques like distributed data bases.[8] The DBA's technology growth will also force the group to become highly technical in nature and highly specialized.

Further research and assistance is needed for the DBA and the user, who will be interfacing more with the DBA in the future. More automated design tools must be developed for the user to be able to present his design to the DBA in a clear manner both at initial development and at operation time. Very few of the DBA's had the time to monitor and optimize the use of the data base. More research into modeling and optimizing would aid in this area. The problem of an MIS still has not been solved in these systems. The data bases become so complex that managers are more lost than ever, and research on an effective way for true management use of complex DBMS systems is needed.[19] Research in the economics of DBMS and their staffing is clearly needed. Organizational issues and administrative structuring seem to be a must for expanded research. For those who claim that data is a resource the last two issues are clearly a critical area to justify that position. Finally, the impact of the new technologies on the DBA must be a constant concern.

REFERENCES

1. ANSI/X3/SPARC, "Study Group on Data Base Management Systems," Interim Report 75-02-08, *FDT—Bulletin of ACM-SIGMOD,* Vol. 7, No. 2, February 1975.
2. Auerbach, "Data Base Administrator," Part 1, Doc No. 3-06-04, Auerbach Publishing Company, Philadelphia, 1975.
3. Cagan, C., *Data Management Systems,* J. Wiley, Inc., New York, 1973.
4. Canning, Richard G., "The Data Base Administrator's Function," *EDP Analyzer* Vol. 10, No. 11, November 1972.
5. Canning, Richard G., "The Cautious Path to Data Base," *EDP Analyzer* Vol. 11, No. 6, June 1973.
6. CODASYL Programming Committee, "Data Base Task Group Report," ACM, New York, April 1971.
7. CODASYL Systems Committee, "Feature Analysis of Generalized Data Base Management Systems," ACM, New York, May 1971.
8. Comba, Paul G., "Needed: Distributed Control," *Proceedings of the International Conference on Very Large Data Bases*, ACM, New York, September 1975.
9. Date, C. J., *An Introduction to Data Base Systems,* Addison-Wesley, Reading MA, 1975.
10. De Blasis, J. P. and T. H. Johnson, "Data Base Administrators: Review of Current Recommendations vs. Current Practice," Decision Sciences Department Working Paper 76-03-03, The Wharton School, University of Pennsylvania, Philadelphia, 1976.
11. Diebold Research Program, "Organizing for Data Base Management," The Diebold Group Inc., Doc. S16, New York, December 1974.
12. Guide-Share Data Base Requirements Group, "Data Base Management System Requirements," Share Inc., New York, November 1970.
13. Guide-Share, "The Data Base Administrator," Data Base Administration Project, IBM DB/DC Systems Center, Palo Alto CA, November 1972.
14. Johnson, T. H. and J. P. De Blasis, "Data Base Administrators: A Report From Survey," Decision Sciences Working Paper 76-04-04, The Wharton School, University of Pennsylvania, Philadelphia, 1976.
15. Johnson, Thomas and J. P. De Blasis, "Very Large Data Base Administration: Organizational Implications From a Survey," Decision Sciences Working Paper 76-04-06, The Wharton School, University of Pennsylvania, Philadelphia, 1976.
16. Katzan, Harry, *Computer Data Management and Data Base Technology,* Van Nostrand, New York, 1975.
17. Lyon, John K., "The Role of the Data Base Administration," *Data Base* Vol. 3, No. 4, ACM-SIGMOD, New York, Winter 1971.
18. Martin, James, *Computer Data Base Organization,* Prentice-Hall Publishing Company, Englewood Cliffs NJ, 1975.
19. Morgan, Howard L., "Report on The Very Large Data Base Conference," *Communications of the ACM,* New York, November 1975.
20. Nolan, Richard L., (Editor), *Managing the Data Resource Function,* West Publishing Company, 2074.
21. Sanders, Donald H., *Computers and Management,* McGraw-Hill, New York, 1975.
22. Sanfield, Stuart H., "Data Base Administration: One Approach," Internal Report, Wyeth Laboratories, Philadelphia, 1976.
23. Secrest, Richard, "The Data Base Administrator," in *The Information System Handbook* by F. W. McFarland and R. L. Nolan (Ed.), Dow Jones, Irwin, Homewood IL, 1975.
24. Weldon, Jay-Louise, "Data Base Administration: Theory and Practice," NYU Working Paper 75-75, New York, 1975.

Data dictionary—More on the impossible dream

by MICHAEL EHRENSBERGER

Cincom Systems, Inc.
Cincinnati, Ohio

ABSTRACT

This paper discusses the role of Data Dictionary software in the computing function. It specifically discusses the evolutionary process which brings about the need for a Data Dictionary. It goes on to describe the major components and requirements of Data Dictionary software along with its interaction with data base and the data base administration function.

The advantages, benefits, and potential drawbacks from the misuse of this software are also examined.

INTRODUCTION

Since the initiation of computing, the data processing industry has been preoccupied with hardware selection, programming techniques, project management, and the like. Computing, like other emerging disciplines, tended to orient its management philosophies around the physical hardware rather than the function the hardware performs. Recently, this concept of computer management has taken on a new meaning. The computing industry is no longer preoccupied with computing; rather, it is concerned with the management of a vital corporate resource—data. For that reason, whenever we speak about management in the context of data processing or computing, it's important that we realize that data is the thing that we're managing.

This paper will deal with the acceptance of a new tool used to control that data—The Data Dictionary. At the same time, it will try to address managerial techniques whereby we can more effectively manage the data within our own department.

Over the last four to five years the whole concept of data base and data communication systems has emerged. This particular philosophy has been adopted by a majority of the data processing installations throughout the United States. Because of the tremendous acceptance of data base and data communication systems, the amount of processing required of a computer has expanded geometrically. At the same time, data, standards, procedures, programs, systems, reports, and personnel within data processing operations have also expanded at a geometric rate. The management of the computer operation itself is an increasingly challenging job and one which calls for new and different skills. A new position has been established to deal with the management of data—that of the data base administrator. A key tool of a data base administrator is the Data Dictionary. It is the intent of this paper to try to deal with the role of the Data Dictionary in the management of the computing function. In order to achieve that, the following topics will be covered:

1. The evolution of the need for Data Dictionary
2. Management needs to be addressed by the Data Dictionary
3. The advantages and benefits of the Data Dictionary

THE EVOLUTION PROCESS

Computing and data processing is not unlike the growth of other disciplines in business today. You might reflect back on the emergence of such disciplines as production control, manufacturing management, and the like. It was only after the intensive studies of people like Frederick W. Taylor and Henri Fayol that manufacturing management gained wide acceptance throughout industry in the U.S. At the same time, it was not until the introduction of double entry accounting systems that consistent accounting controls were established. Computing and data processing, like other management disciplines, is now recognized as an important part of the organization. This recognition has not been sudden; rather, there have been various stages of development. These stages have been described by Dr. Richard Nolan of Harvard University. The question of "when Data Dictionary" can be answered by determining what stage of development a particular data processing operation has achieved.

According to Dr. Nolan, there are four, possibly five, development stages. The first stage of growth is the initiation stage when we first acquired the computing power within our organization. During this initial stage, the applications are primarily oriented towards accounting; the personnel we hire are oriented towards the effective use of a particular piece of hardware; and the management itself, typically, is management contained within the functional area which approved that piece of computing power. During this initiation stage, the computer is an under-utilized piece of capital equipment, probably only used 50 percent of the time. This under-utilization provides the rationale for entry into the second stage. During the second stage, we begin to expand. Applications are proliferated in all areas; the personnel within data processing become highly specialized in programming languages. With respect to management, we are oriented towards selling computing services. It is obvious at this stage that we are managing a piece of hardware—the computer itself. There is also a dramatic rise in the budget associated with data processing. This increasing cost led us to the third stage.

The third stage is the stage of control. Typically, no new applications are done; rather, existing applications are rewritten in a native mode or to be more efficient. From a personnel point of view, this is the age of the operating systems. Emphasis is placed on systems programming and the fine tuning of hardware/programs. From a managerial point of view, there is tremendous upheaval, i.e., reorganizations, centralization, de-centralization, etc. It is also during the stage of control that people recognize the vital importance of data processing within an organization. It is at this point that users begin to ask data processing to modify existing systems, to enhance systems, etc. Conversion is fast becoming a way of life as users recognize the importance of data. The integration of data, systems, and programs brings about the fourth stage and the need for new, sophisticated tools. It is at this point in time that some new concepts are introduced, i.e., the concepts of data base/data communications. Along with these concepts comes a dramatic change in the role of computing within the organization. Management of the computing function is now a key job in the organization. This is certainly true when you take a look at the effect that this particular function can have on the overall profit and loss of a corporation. Companies who are cognizant of this are well into the fourth stage. The fourth stage is characterized by a recognition that the responsibility of data processing managers is the management of data, not management of the computer. It is not unlike the basic recognition of the production manager that his role is not the management of machines; rather, it's the overall management of production.

THE NEED FOR A DATA DICTIONARY

The Data Dictionary is designed to manage the data within the data processing department. Like the story of the cobbler who made shoes for his family only after he had satisfied the needs of the villagers, the Data Dictionary provides to data processing the same needs that data processing has historically provided to user departments, i.e., the management of their data.

The onset of integrated data management brought with it a more complex environment and an environment which required new and different tools. Data Base provides functional integration while the Data Dictionary provides the control of that Data Base. Data Dictionary like Data Base is a prerequisite for evolution into the fourth stage. The Data Dictionary should contain information about, and the relationships among the entities within the realm of data processing. These include:

Data Files
Data Fields
Programs
Data Bases
Reports
Systems
Users
Departments
Projects
Transactions
Personnel
Standards
Source Documents
Security Levels

The Data Dictionary should provide utilities to automatically generate information about the above. Utilities which scan existing programs or libraries could be used to generate a good deal of the required information. Other information will need to be researched and/or established with human intervention.

The Data Dictionary should also play an active role in the day to day operations. For that reason, a Data Dictionary should include the following features:

Automatic program set-up

A programmer should not be concerned with coding a "data division" or "I/O" areas within a program. This information should be stored within the Dictionary and automatically copied or invoked at compile time. This feature substantially reduces programming and maintenance time while insuring security and control over the data available to programs.

DBMS interface to the data definition language

Data Base Management Systems and Data Dictionaries must fit hand in glove. For that reason, the Data Definition required for each data record, data set, and data base should be stored within the Data Dictionary and automatically generated on request. Once again, this feature is a necessity for effective data base administration.

On-line data dictionary access

The trend toward cardless systems requires that the Dictionary provide on-line editing and update. Currently on-line editing and validation is built into each on-line program. Access to the Dictionary would all but eliminate this redundant programming effort. The savings are obvious. On-line updates of the Dictionary allow dynamic changes to editing criteria, security levels, passwords, etc. With the advent of tighter privacy legislation, this capability will be required.

Automatic report generation

The Data Base Administrator will need various reports about the Dictionary. Current information and relationships about systems, applications, fields, records, files, programs, users, terminals, etc., are required. The report feature should be an embedded part of the Dictionary. In addition to the standard reports, the Dictionary should be constructed such that non-standard, ad hoc reporting is possible.

Data base documentation support

This rather nebulous sounding characteristic may well be the most important. In essence, it allows a DBA to enforce data base usage and programming standards. This is accomplished through a comprehensive description of a company's data base. It encompasses the following components:

- Complete Attribute Description of All Entities.
  As an example, the attribute description of a data field should include its name, description, alias, programs which manipulate or require the field, source documents and reports which contain the field, editing criteria, field length, usage, decimal point placement, occurs, password, security level, date of last update, program which last updated, data set(s) which contain that field, record type in which field is located, displacement within the record, etc.
- Automatic Relationships Between Entities.
  As an example, the Dictionary should automatically link or relate a data field to programs, data sets, reports, editing criteria, source documents, users, alias.
- Data Base Status and Version Control.
  For security reasons, the Data Dictionary should control the version or view that a program has of the data base.

Security support

The Dictionary should provide for security in two areas. Entity security applies to the protection of data files, fields, programs, etc. Data Dictionary security applies to the protection of the Data Dictionary itself. At a minimum level, entity security should provide for password security by program, terminal location, and individual. Security levels should also be assigned to certain entities; i.e., fields, files, programs, reports. Security of the Data Dictionary is more complex. The Data Dictionary files, the actual code, the execution of Dictionary programs, Dictionary reports, etc., all require a level of security.

Integrity support

Integrity support encompasses edit and validation functions for both batch and on-line, the ability to generate test data and test data bases, and the ability to support distributed data bases. The distributed data base concept is an idea whose time has come. For that reason, support for multiple dictionaries, traffic routing, multiple DBMS installations, and extensive directory features are needed.

DATA DICTIONARY—ADVANTAGES, BENEFITS, POSSIBLE DRAWBACKS

Each of us in data processing has been faced with a user requested change, i.e., the addition of a new field to an existing record, the expansion of an existing field, a change in a report format, etc. Timely response to such requests are a measure of our managerial capability. Not only for ourselves but also from a planning point of view, it is important in the effective management of our data centers that we be able to accurately determine and quantify the impact of user-requested change. A Data Dictionary, as previously defined, provides answers to these questions. Interestingly enough, the ability to accurately assess the impact of change is a key indication of the level of development of any management discipline.

EDP audits are always in vogue. Consistent documentation and standards are an important aspect of such audits. The Data Dictionary eliminates much of the manual labor associated with documentation while substantially upgrading the quality of the documentation. The cataloging of entity attributes provides programming standards, naming standards, data base standards, security, and integrity. It also provides a common repository of data about the data base thus allowing tight control by a Data Base Administrator.

Possibly, the one drawback of the Data Dictionary is in the overhead associated with day-to-day operations. If all access to data requires an additional access to the Dictionary, then the overhead could be substantial. Like many new concepts, the Data Dictionary should be used with enthusiasm but with a measure of discretion. As new technologies emerge, the overhead of accessing data will probably reduce to core-to-core transfers. At that point, the full concept of data base—data dictionary—can and will be a reality.

In conclusion, it is my opinion that data processing had indeed emerged through four stages. In fact, Dr. Nolan has now published a new article indicating the fifth stage of development. The fourth and fifth stages are obviously dependent upon the tools required for effective management of those stages. Data base/data communications are keys to the success and transition through stages four and five. The Data Dictionary is a key ingredient to this success. From all indications, the major corporations in the U.S. will be adopting the concepts of Data Dictionary in the next year. In our opinion, it is a key ingredient to managerial success in the computing function.

REFERENCES

1. Ehrensberger, M. J., "Data Base—The Solution to the Impossible Dream," *Proceedings of the 19th College and University Machine Records Conference,* May 5-8, 1974.
2. Flynn, R., "A Brief History of Data Base Management," *Datamation,* August, 1974.
3. Nolan, R. L., "Managing the Computer Resource: A Stage Hypothesis," *Communications of the ACM,* Vol. 16, No. 7, July, 1973, pp. 399-405.
4. Ferone, W., "Data Dictionary—A Competitive Analysis," unpublished.

Fault tree analysis of computer systems

by C. V. RAMAMOORTHY and G. S. HO
University of California, Berkeley
Berkeley, California

and

Y. W. HAN
Honeywell, Inc.
Minneapolis, Minnesota

ABSTRACT

Fault Tree Analysis (FTA) is a well developed technique for the reliability and safety analysis of complex systems such as nuclear power plants and weapon systems. In this paper, we apply FTA to analyze the reliability and the performance of computer systems. An approach to detect the sequence dependent faults in computer systems is proposed and exemplified. Based on the fault tree analysis, guidelines for up-grading the system can be developed.

INTRODUCTION

Fault Tree Analysis (FTA) has been used extensively to analyze the reliability and safety of complex systems such as nuclear power plants and weapon systems.[1] It identifies faults in a system design that may cause potential accidents and helps to eliminate costly design changes and retrofits. In this paper, we discuss the application of FTA to computer systems. For completeness, a brief summary of fault tree analysis techniques is given in the next section. In order to suit the readers of this paper, Boolean Algebra is used for explanation. In later sections, applications of FTA to analyze the reliability of computer systems are discussed; both hardware and software issues are addressed; and application of FTA to protection in computer systems is discussed. The unique aspects of sequence dependent faults are revealed and exemplified. Examples are used extensively in the paper for illustrating the concepts.

INTRODUCTION TO FAULT TREE ANALYSIS

Fault Tree Analysis was first conceived in 1962 by H. A. Watson of Bell Telephone Laboratories, for an Air Force contract studying the Minuteman launch-control system. Further development and refinement of the technique resulted from the efforts of A. B. Mearns and the Bell study team. They successfully solved the problem of determining the likelihood of the unintentional launching of a missile. The Boeing Company analysts later modified the fault tree technique so that simulation was possible using high speed computers. D. F. Haasl, R. J. Schroder, W. R. Jackson and others contributed to this important development.[9] In the mid-1960's, Fault Tree Analysis had been used extensively in aerospace industries and weapon systems.

To begin the description of FTA, let us introduce the terminology and symbolisms used. *Component state* or, more generally, *basic event* is the failure situation which results when the functions performed by the system element deviate from its specified limits. The *Top Event* which appears at the top of the Fault Tree is the undesired (usually catastrophic) event under consideration. By convention, basic events are represented by circles and the top event by a rectangle. With a given Top Event, the fault tree can be constructed by recursively decomposing the causes of the faults. A simple example will illustrate the concept clearly.

Suppose we have a system as shown in Figure 1a. The system is considered functioning properly if there exists a path from node s to node t that does not contain a bad node. A Fault Tree for the system with Top Event as "Improper Operation" is shown in Figure 1b and Figure 1c. If Triple Modular Redundancy, Figure 2, is implemented in node a, the Fault Tree can then be extended to Figure 3 for more detailed analysis. Extensive analysis techniques for

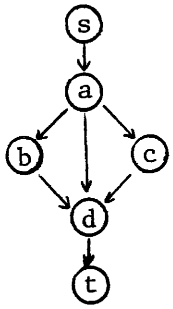


Figure 1a

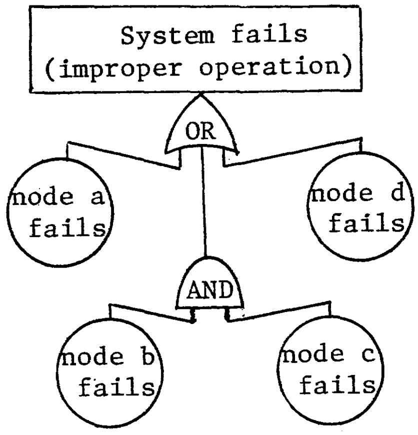


Figure 1b

FTA have been developed. Some important results relevant to this paper are summarized below.

Minimum cut set

A *cut set* is a set of basic events whose occurrence causes the top event to occur.[2] A cut is *minimal* if it cannot be reduced and still insures the occurrence of the Top Event. By enumerating all the min cut sets, the system failure characteristics can be revealed. The weakest aspects of the system can be identified and up-graded. In a complex system, the Fault Tree will be very large and it is hard to generate all min cuts by inspection. In this case, we first formulate the Boolean representation of the occurrence of the Top Event. By expanding the expression into disjunctive normal form and deleting all replicated literals in a product term, all min cuts sets (product terms in the expanded Boolean expression) can be generated. For example, the Fault Tree of Figure 3 can be represented by $[m+(a_1a_2+a_1a_3+a_2a_3)]+bc+d$. After expansion, we have $m+a_1a_2+a_1a_3+a_2a_3+bc+d$. The Fault Tree contains six min cuts, namely m, a_1a_2, a_1a_3, a_2a_3, bc and d. If there are no event replication among min cut sets and basic events are statistically independent, the probability of occurrence of the top event can be computed by

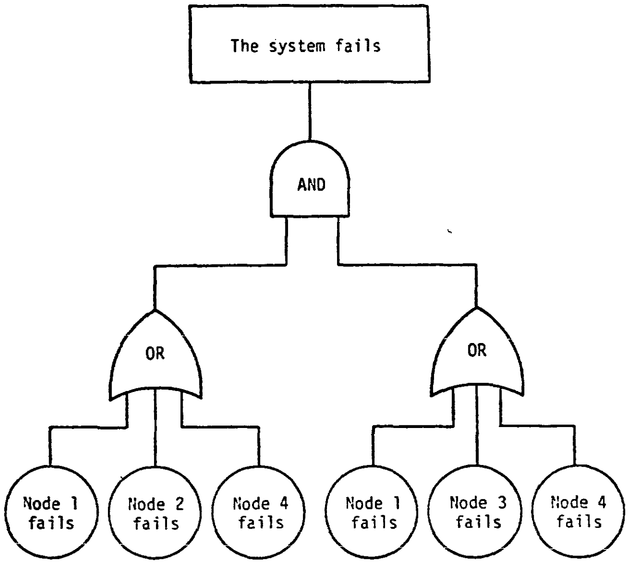


Figure 1c

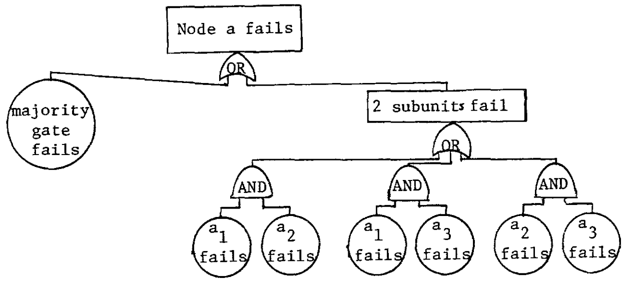


Figure 2

$$P(\text{Top Event})=1-\prod_{s=1}^{r}(1-\prod_{i\in K_s} q_i)$$

where n=# cut sets

K_s=the s^{th} cut set

q_i=probability of occurrence of basic event i

For the Fault Tree of Figure 1c,

Prob(improper operation)

$$=1-(1-q_m)(1-q_{a_1}q_{a_2})(1-q_{a_1}q_{a_3})(1-q_{a_2}q_{a_3})$$

$$\cdot(1-q_bq_c)(1-q_d)$$

In complex systems, e.g., electronic computers, the correct operations of the system require stringent synchronization and co-operation of all the units in the system. The assumptions made above in computing Prob(Top Event) are not likely to hold. In fact, if there are replicated terms in the

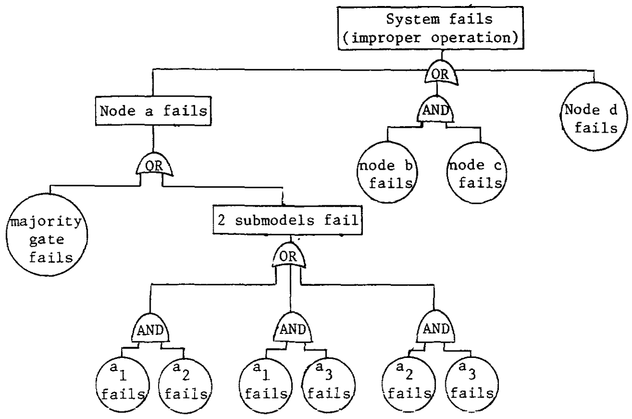


Figure 3

min cut sets, it has been proven that the computation for Prob(Top Event) is NP-complete.[5] Fortunately, fast algorithms giving close bounds on the Prob(Top Event) using the concepts of min cut sets and *min path sets* of the *Dual Fault* Tree have been developed.[1] Using these algorithms, the failure characteristics of the system can be predicted and guidelines for up-grading the system can be formed.

ESTIMATION OF COMPUTER SYSTEM RELIABILITY

Fault tree representation can be considered as a systematic management tool. In comparison with tables and dictionaries, the tree structure is more convenient and natural to express a system hierarchically. We can delineate mixed levels of detail easily and flexibly with a fault tree. If necessary, a basic event can be extended as another fault tree at a lower level as demonstrated in the previous section. As an approach to safety and reliability analyses, FTA is extremely powerful. Usually, a system is designed to work. Using FTA, we view it from the other end: how it may fail. From this angle, failure combinations which otherwise might not have been recognized can be uncovered.

The analysis of computer system reliability shares many common problems with the analysis of the other systems' reliability to which FTA has been applied successfully. Naturally, we may ponder the applicability of FTA to computer reliability.

Application of FTA to hardware faults in a self-repaired computer has been studied by Jack Goldberg[3] and several problems have been identified for this application, namely the difficulties to represent time dependent fault conditions and to estimate the probability that a basic event occurs. In this section, we shall discuss this approach further, especially on the application of FTA to analyze both hardware and software reliability together.

Both the hardware and the software have to be considered in evaluating the reliability of a computer system. People may think software does not fail. But, unfortunately, up to now, there exists no verification or testing method to prove the correctness of large software systems. For instance, a considered-to-be-thoroughly-tested software for the Apollo 14 turned up 18 discrepancies during the 10-day flight, and every release of the IBM OS/360 contains roughly 1,000 bugs.[4] To deal with these hardware, software or both hardware and software reliability problems, we definitely need a systematic management tool. FTA is a well-suited method for analyzing both hardware and software reliabilities involving high interdisciplinary principles. However, we anticipate some problems in using FTA and they will be discussed below.

One major problem is the difficulty to estimate the probability of occurrence of the basic events, that is, to find the failure rates or the error distribution functions of the system units. The failure rate of the hardware is usually obtained through statistical inference based on the circuitry package, the fabricating technology, density, complexity, working environment of the system, etc. Statistical inference is used because by the time sufficient data on a product are collected, the product may have become obsolete. To quantify the reliability of a software program is by no means an easier task than to quantify hardware reliability. Although there exists no rigorous measure of the reliability of a program, based on the length, graph structure, data structure, programming language used, and number of tests passed, we can assign a figure of merit to approximate a program reliability. With a structured, high level language we usually generate programs more reliable than those written in an unstructured, machine level language. Usually business programs with very simple data structures are more reliable than real-time programs which often have complicated data and control structures and therefore are error-prone. With the estimated probability of the basic events, the error behavior of the system can be predicted. Although the analysis of a system using FTA is incomplete, FTA can be very valuable in guiding engineering decisions, analyzing consequences and providing design guidelines.

One major aspect of applying FTA to computer systems is that there are cases in computer systems, in which the consequences depend on the order of events. For example, event A preceding event B may cause a different consequence from the one caused by event B preceding event A, although both events A and B occur. To analyze these order-dependent event sequences, the construction of a fault tree has to include all the scenarios in which the precedence plays a role. The precedence relationships among these events have to be represented in the fault tree. These techniques are illustrated by an example in the next section.

APPLICATION OF FTA TO PROTECTION IN COMPUTER SYSTEMS

Frequently, a computer system is designed and implemented to satisfy certain multiple requirements such as fast throughput rate (good performance), high reliability, high availability, low cost, good protection, and privacy in a resource-sharing environment, etc. FTA can be used as a top-down management tool to handle these problems. In this section, we shall give a simplified, hypothetical example of using a fault tree to analyze the protection mechanism of a computer system. Subsequently, the application of FTA to the performance issues will be investigated. Finally, we shall outline, in general, how to apply fault tree techniques to multiple requirements.

Let us consider a hypothetical protection example which contains order-dependent events, as explained in the previous section.

In a resource-sharing environment, protection is needed to preserve the privacy of every user, as well as to limit the propagation of errors caused by faulty system hardware, software, or users' programs. In a protection system, the information about the rights (called access capabilities) of an active program to the resources which are not open to all users are usually kept in an *access matrix* in Figure 4 a protected memory. The access capabilities here can be classified as: READ (including COPY), WRITE, EXECUTE, DELETE and APPEND. To reduce the overhead, the set of programs having the same capabilities are grouped as a *subject* in the access matrix. Resource can be either virtual or physical: files, privileged instructions, segments of memories or channels, and programs. Similarly, a set of resources is grouped as an *object* if the set can share the same identification as far as protection is concerned.

The system works as follows. Suppose a program requests to READ a certain privileged file. An internal interrupt is generated. The system is switched to the system mode and the system examines the access matrix to check whether the program (subject) has the READ capability of the file (object). If it has the capability, the process will be continued, otherwise the request will be denied.

To handle a chain of requests correctly, we propose tracing the first subject of the chain of requests. We will assume that the capabilities of the first subject determine whether requests of the (N+1)th subject which is also the Nth object, N≥1, can be proceeded. As an example, if A executes B and before the end of this execution, B requests to WRITE on C, then this request should not be allowed. This is because A is the first subject of the chain and it does not have the implied capability.

Now, suppose the software system design has a loop hole. Instead of tracing the first subject of a chain of requests, it only checks the entry determined by the current subject and the current object from the access matrix. We further assume that the system has a watchdog timer for limiting the total time spent by a chain of requests as a check and error detectors for checking the privileged memory which stores the access matrix and logic units which handle the requests. Then, the breaches of the protection can be represented by the fault tree shown in Figure 5. Note that a hardware or software failure may not lead to a protection breach if the detectors can detect the breach.

Using Fault Tree Analysis for performance analysis is a dramatic extension to the current applications of FTA, since it is usually applied for safety and reliability problems. There are many real-time tasks which must fit a stringent performance requirement. Specifically, we may consider the top event as a task which cannot be finished within a prespecified turnaround time. To generate the fault tree, two main factors which determine the turnaround time will be considered: (1) current available resources, and (2) characteristics of the task.

	A	B	C
A	E,R,W	E	---
B	---	E,R,W	W

where E denotes EXECUTE,

R denotes READ

and W denotes WRITE.

Figure 4

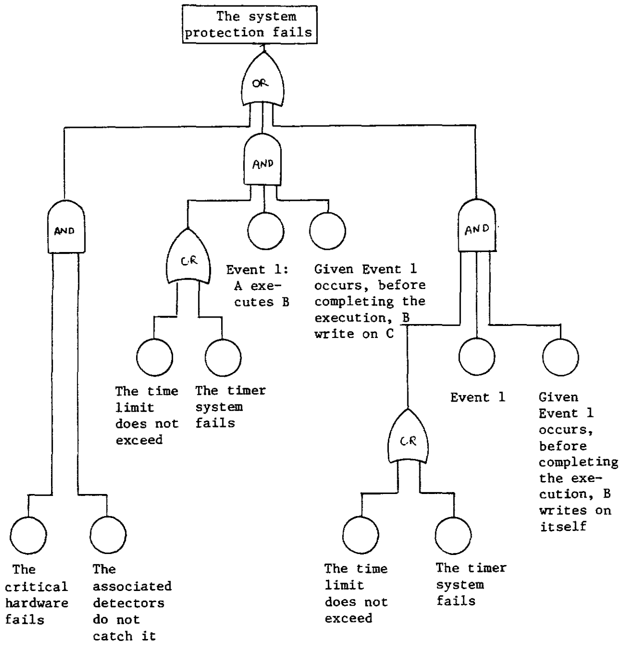

The system protection fails
OR
AND
AND
AND
OR
Event 1: A executes B
Given Event 1 occurs, before completing the execution, B write on C
The time limit does not exceed
The timer system fails
Event 1
Given Event 1 occurs, before completing the execution, B writes on itself
OR
The critical hardware fails
The associated detectors do not catch it
The time limit does not exceed
The timer system fails

Figure 5

The available resources can be hardware buses, processors, memory, or data which can be the results of other computations. In a multiprocessing environment, the situation is further complicated by the problems of resource contention. For a preemptively scheduled system, the turnaround time of a task also depends on the other tasks which will have come before the completion of this task. In short, this factor is determined by the other processes and the host machine. But even with a system dedicated to this task, the turnaround time is still not deterministic in many cases; it may be input data dependent. For example, the number of comparisons needed to sort n elements by Quicksort ranges from 0(nlogn) to 0(n^2), depending upon the ordering of the input elements. As another example, as mentioned previously, there exists a very fast algorithm to calculate the probability of the top event for a fault tree with no basic events replicated, but it can be proven that, in general, there exists a worst case which is very time consuming (i.e., it is NP-difficult).

To perform the overall performance analysis, the top-down management tool (FTA) facilitates the representation and thus the simulation and analysis of the real computer system.

To apply FTA to a system designed to satisfy multiple requirements, fault tree analysis is performed on each requirement. These requirements are arranged in a priority order according to their importance. For example, if a computer is used as a controller for an electric power system, the priority may be in the order of safety of the power system, reliability of the computer, and performance of the computer. The priority list for a computer used in a banking system may be protection of the customers' privacy, reliability of the computer, and performance of the banking system. A fault tree is constructed for each requirement either during the design stage before it is committed to implementation or for updating an existing system to satisfy a more stringent requirement due to a changing environment. From the fault tree, we identify the critical conditions and thus help to redesign the system to meet the requirements.

The sequence of constructing these fault trees follows the order in the priority list, i.e., the trees are constructed and then reconstructed for the first priority, then the second, and so on. If we cannot find a fault tree that satisfies the n^{th} requirement, we shall backtrack to the $(n-1)^{th}$ requirement and construct a new fault tree for it. This suggested process is similar to the depth-first branch and bound method.[8]

CONCLUSIONS

Analysis of the overall behavior of a computer system is a very complicated task. Fault Tree Analysis provides a top-down hierarchical procedure to analyze the reliability of the system at different levels. A basic event, if necessary, can be extended further as the top event of another fault tree. Well-developed algorithms for identifying the crucial conditions (min cuts) of the tree and for calculating the prob(top event) are available. Despite the existence of some difficulties in applying FTA to computer systems, mentioned earlier, we believe it is a good analysis tool to predict the fault behavior of a system and is worthy of research. One direction is the automatic construction of the fault tree of a system. To facilitate the automatic construction, one approach is to represent the hardware system, software system and controlled processes as a labeled graph.[7] The *concurrency and synchronization* of the events can then be indicated dynamically, and all the sequence-dependent behavior can be revealed. However, more work has to be done before such an approach becomes practical.

REFERENCES

1. Barlow, R. E. and J. B. Fussell, "Reliability and Fault Tree Analysis," *SIAM*, 1975.
2. Fussell, J. B. and W. E. Vesely, "A New Methodology for Obtaining Cut Sets," *American Nuclear Society Transactions*, 15, No. 1, 1972, pp. 262-263.
3. Goldberg, J., "A Survey of the Design and Analysis of Fault-Tolerant Computers," *Proceedings of the Conference on Reliability and Fault Tree Analysis*, Berkeley, CA, 1974. It is Ref (1), pp. 687-732.
4. Boehm, B. W., "Software and Its Impact: A Quantitative Assessment," *Datamation*, May 1973, pp. 48-59.
5. Rosenthal, A., "A Computer Scientist Looks at Reliability Computations," *Proceedings of the Conference on Reliability and Fault Tree Analysis*, Berkeley, CA, 1972, pp. 133-152.
6. Mazumdar, M., "Importance Sampling in Reliability Estimation," *Proceedings of the Conference on Reliability and Fault Tree Analysis*, Berkeley, CA, 1974, pp. 153-163.
7. Miller, R. E., "A Comparison of Some Theoretical Models of Parallel Computation," *IEEETC*, Vol. C-22, No. 8, August 1973, pp. 710-717.
8. Garifinkel, R. S. and G. L. Namhauser, *Integer Programming*, Wiley Interscience, 1972.
9. Haasl, D. F., "Advanced Concepts in Fault Tree Analysis," *Proceedings of the System Safety Symposium*, Seattle, 1965 (The Boeing Company, Seattle, 1965).

An overview of fault-tolerant digital system architecture

by STEPHEN Y. H. SU and RICHARD J. SPILLMAN
Utah State University
Logan, Utah

ABSTRACT

With the increasing use of computing systems in such crucial areas as medicine and space, there has come a great need for computers that remain operational in spite of hardware failures. This paper provides a brief overview of several approaches to fault-tolerant computing. Five hardware redundancy techniques are reviewed: static, dynamic, hybrid, self-purging and the reconfiguration scheme. In addition, the advantages and disadvantages of error correcting codes and software fault-tolerant systems are outlined as well as bi-duplexed systems, alternating logic, fail-soft and shared logic systems. It is suggested that perhaps the best fault-tolerant system employ a combination of hardware redundant techniques and software protection.

INTRODUCTION

Since the early 1940's when the first relay computers were developed, the question of how to insure reliable computer operation has been an important one. Today, when computers are used in critical space missions, millions of miles from their human operators, and in biomedical systems where a human life depends on their correct operation, ~~even a small computing error could result in the loss of a~~ life or millions of dollars of equipment and years of research. Under such conditions, the design of computing systems which can operate correctly in spite of hardware or software failures is important. Such systems are called fault-tolerant systems. Specifically, fault-tolerant computing has been defined as the ability to execute specified algorithms correctly regardless of hardware and/or software failures.[1]

The first step towards a fault-tolerant system is to build as much fault-tolerance into the system as possible.[2] Fault-intolerance is the procedure whereby the reliability of the system is increased by avoiding the causes of system failures. This is achieved before the final system is constructed, in the design phase. Only the most reliable components are selected, software is completely tested before it is released, and fault detection and ease of repair considerations are introduced at the beginning of the design process and considered at every succeeding step. Fault-intolerance, however, can only postpone the occurrence of faults, it can not eliminate them entirely. Hence, the second step towards a fault-tolerant system requires protecting the system from faults.

There are several approaches to fault protection including hardware redundancy where extra components are introduced into the system, error correcting codes such as parity checkers, or software fault-tolerant systems where special software procedures are used to recover from an error. In this paper, all three approaches will be briefly reviewed and their advantages and disadvantages outlined. Specifically, sections of this paper will examine five redundancy techniques: masking, standby, hybrid, self-purging, and reconfiguration; will review error correcting codes; software fault-tolerant procedures will be outlined; a number of new approaches to fault-tolerance will be reviewed; and will draw some conclusions and offer some suggestions for future research.

REDUNDANCY TECHNIQUES

The effects of hardware errors can be overcome through the use of protective redundancy.[3] Hardware protective redundancy is defined as the use of additional components which allow the system to continue to operate correctly in the presence of hardware faults. The cost of the extra components was, at one time, a strong argument against the use of redundancy. However, since the advent of LSI and MSI and the reducing cost of digital hardware, redundancy has become an important means of implementing fault-tolerant systems. There are three classifications for the conventional redundancy techniques: static, dynamic, and hybrid. In addition to these three, two other approaches are discussed separately in this paper, self-purging redundancy and reconfiguration scheme redundancy which will introduce different ideas from the previous three.

Static redundancy

Static redundancy involves the use of extra components such that "the effect of a faulty circuit, component, subsystem, signal, or program is masked instantaneously by permanently connected and concurrently operating circuits."[4] It is also called masking or massive redundancy.

The simplest static redundancy technique is triple modular redundancy (TMR). This technique was first studied by John von Neumann.[5] It is implemented using three identical modules operating in parallel as shown in Figure 1. The output of each module passes through a majority voting system whose output agrees with the majority of module outputs. TMR could be expanded to include any odd number of redundant modules to produce an NMR (N Modular Redundancy) system, where N is an odd number.

An NMR can tolerate (N−1)/2 module failures. The three major advantages of static redundancy are:

(1) The corrective action is immediate, the faulty module never effects the circuit.
(2) There is no need for fault detection procedures.
(3) The conversion of a non-redundant system to a static redundant one is easily undertaken. Simply construct (for a TMR system) two new copies of each non-redundant module.

It is important to remember the primary assumption behind the use of static redundancy. That is, a failure in one module is independent of the other modules. This assumption is not valid within an LSI or MSI package and hence, static redundancy is ineffective at the logic gate level within LSI or MSI.

Dynamic redundancy

Dynamic redundancy involves only one unit operating at a time with several spares waiting to replace the unit if a fault is detected. Obviously, this system requires both a method of switching the new module into the circuit to replace the faulty unit and a method of detecting the fault in the circuit. There are two possible switching procedures, logic switching and power switching. In logic switching, all the spares are powered up and operating, when a fault is detected the output of the next spare in line is switched into the circuit and the faulty modules output is switched out. This is equivalent to enabling or disabling a gate between the module's output and the system output. Power switching requires that only the operating unit be powered up. When a fault is detected, the faulty units power is switched off and the next spare in line is switched on. The power switching procedure isolates the spares from the circuit and does not waste power on spares which are not performing useful operations.

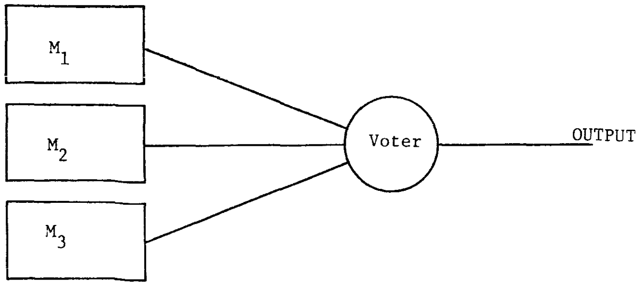


Figure 1—Triple modular redundancy (TMR)

The fault detection procedure is more complicated and could include either concurrent or periodic techniques.[6] Concurrent fault detection is a continuous fault detection technique which utilizes coding and redundant signals. Periodic fault detection requires testing the circuit at certain points in time with a special diagnostic routine. This involves stopping the normal operation of the circuit to conduct the tests, however.

If a working fault detection system can be constructed, there are several advantages to a dynamic redundant system:

(1) All the spares can be used and the system can tolerate as many faults as there are spares.
(2) The number of spares is easily adjusted to allow for later increases in reliability if needed.
(3) If power switching is used, the spares are isolated from the system so the independent fault assumption applies.

Hybrid redundancy

Hybrid redundancy combines the static and dynamic redundancy approaches. Static redundancy, used to provide fault detection, is combined with a set of spares to replace any faulty module in the static system. The basic hybrid redundancy system is the TMR + spares system (Hybrid (3,S) system) shown in Figure 2.[10]

In the Hybrid (3,S) system, if a fault occurs (the output of one of the gates does not agree with the output of the other two) the faulty gate is switched out and one of the spares is

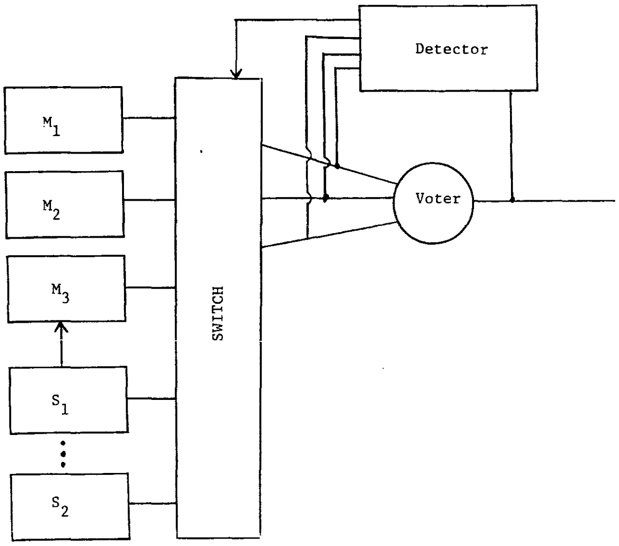


Figure 2—Hybrid (3,N) system

switched in. The system is then reduced to a Hybrid (3,S−1). If all the spares are used up, the system reduces to a Hybrid (3,0) or just a standard TMR.

The hybrid system possess all the advantages of dynamic redundancy without the problem of constructing complicated fault detection procedures. In addition, the voting circuit of a hybrid system masks the effect of any fault. However, the switching system in hybrid redundancy is more complex than either static or dynamic redundancy switching systems. Hence, hybrid redundancy is prone to switching system failure which could bring the entire system down.

Sieworek and McCluskey[8] have suggested the use of an iterative cell switch. They found that an iterative cell switch could save 25 percent and in some cases 80 percent of switch complexity over standard hybrid switching systems. Ogus[26] has also studied fault-tolerant design of iterative cell switches. Use of the iterative cell switch, then, would lead to further improvements in hybrid redundancy reliability.

As with the TMR, a hybrid system with a TMR core can not handle multiple faults.[8] For example, if two active modules in a Hybrid (3,S) are faulty, then the voting gate will incorrectly switch the one fault-free gate out and switch a spare in. The voting circuit will then determine that the spare is in the minority and replaces it with another spare. This process will continue until all the spares are used up and the system crashes. Ramamoorthy and Han[9] have suggested a modification of the hybrid system in which each module's output passes through its own error detector circuit as well as the voting circuit. Called a detector redundant system, this circuit could detect an error and determine which modules are in error. Only the faulty modules would be switched out, hence, this circuit would not be subject to the catastrophic consequences of multiple failures that plague standard hybrid redundancy. However, such error detection circuitry would increase the complexity of the control system.

If just single faults are assumed, the Hybrid (3,S) system performs very well. A Hybrid (3,S) with N units (N−3 spares) can tolerate N-2 single faults using power switching whereas a NMR (a static system with N units) can tolerate only (N-1)/2 module failures. A dynamic system with N units can tolerate (N-1) failures, one more than a Hybrid (3,N-3). However, the dynamic system requires a complicated fault detection procedure, which in general lowers its net reliability. In addition, dynamic faults are not masked. So the gain of tolerating one additional module failure is outweighed in most applications by the loss of flexibility in the dynamic system.

Mathur and Avizienis[10] found that in adding another module to a Hybrid (n,S), a hybrid system with a nMR core and S spares (where N=n+S), the largest gain in reliability is made if the new module is added to the set of spares. That is, the reliability of a Hybrid (n,S+2) is greater than the reliability of a Hybrid (n+2,S). Therefore, in a hybrid system, most of the redundancy should be in the spares and n should equal 3.

The optimum value of S in a Hybrid (n,S) can also be calculated. Cochi[11] assumed that the reliability of the switching system used to switch the faulty module out and the spare in is a function of n+S. He found a method for calculating an optimum value for S, the number of spares. It depends upon the reliability of both the switching system and the individual units.

Self-purging redundancy

Self-purging redundancy[7,12] is sometimes considered a form of hybrid redundancy. However, its approach is different enough from hybrid systems to offer some real advantages in certain applications. In self-purging redundancy, all of the modules are initially connected to a threshold voter through a switching circuit as shown in Figure 3. If an error occurs in a module, its switch is turned off, forcing a logic 0 on its output. In effect, the faulty module is removed from the voting.

The major advantage of the self-purging system over the standard hybrid approach is the simplicity of the switching mechanisms. The self-purging switching circuit only has to turn a faulty module off, whereas the hybrid switching system has to turn off the faulty module, locate an unused spare and then turn the spare on. Because of the simplicity of the self-purging switch, it is possible to include the switch as part of the module forming a modified module.[12] In addition, because of the simplicity of the switching mechanism, it is easy to duplicate the switches such that each module's output passes through two switches. The number of inputs to the voter is doubled. The reliability of the switching system, which is the weak point of any redundancy scheme, is thereby increased through this use of redundant switching. Such redundant switching is only possible because of the simplicity of the self-purging switching system.[12]

In systems where multiple failures are likely, most redundancy schemes break down. For example, consider a self-purging system with 5 modules and the threshold of the voter is 3. The output of this system is given by $Z=l_1(l_2(l_3+l_4+l_5)+l_3l_4+l_4l_5)+l_2(l_3l_4+l_3l_5+l_4l_5)+l_3l_4l_5$. If two modules fail, say l_1 and l_2 then $l_1=l_2=0$ and $Z=l_3l_4l_5$. So, the majority gate becomes an AND gate. Hence, the system can no longer tolerate any stuck-at-0 fault. If the threshold is 2, then the self-purging redundancy system cannot tolerate a double stuck-at-1 fault.[14] How-

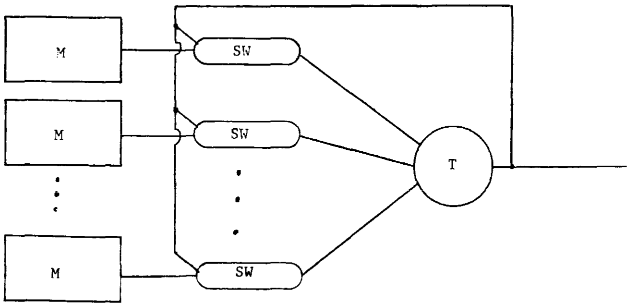


Figure 3—A self-purging system

ever, Losq[13] has developed self-purging systems which have a high probability of recovering from multiple faults, yet Losq's system is not the standard self-purging system shown in Figure 3.

Reconfiguration scheme

Su and DuCasse[14] have suggested a hardware reconfiguration technique for tolerating logic failures. Their scheme begins with a basic 5MR (static redundancy with five modules), it will reconfigure into a TMR under single or double failures. If just a single fault occurs in the 5MR, the system will reconfigure to a TMR with a spare. The equation for TMR can be obtained by substituting any variable, I_i by 0 and any I_j by 1 where $j \neq i$. If the ith module is faulty, Su and DuCasse's scheme forces the output of the faulty module to be stuck at 0 and the (i+1)th module's output to be stuck at 1. This creates a TMR plus a spare. If another fault occurs, the faulty module's output is set to 0 and the spare module, which had its output stuck at 1, is brought back into the voting. The remaining system is a pure TMR. If two simultaneous faults occur in the 5MR, then the system will reconfigure directly into a TMR.

The basic 5MR system can tolerate only two failures. A five module Hybrid (3,2) can tolerate three single failures but any multiple failure will be catastrophic. However, the 5MR reconfiguration scheme can tolerate either three module faults occurring sequentially or a single fault following a double fault. Also, if three faults occur sequentially and the system is reduced to a TMR with one faulty module. Assuming that the probability of a stuck-at-1 is the same as the probability of a stuck-at-0 fault, there is a 50 percent chance of the system tolerating a fourth module failure.[14] Hence, in some applications the 5MR reconfiguration scheme offers a greater fault tolerance than even the hybrid system.

Another advantage of the 5MR reconfiguration scheme over hybrid redundancy is the surprising simplicity of the switching system. Su and DuCasse[14] designed a switching mechanism which will perform the necessary reconfiguration function in the 5MR scheme with only five gates and one flip-flop. It has also been shown that the reconfiguration scheme can be generalized to provide a fault-tolerant system for multiple-valued functions.[14]

It is possible to design NMR reconfiguration systems where N>5. However, the switching system for a NMR reconfiguration scheme (N>5) is no longer as simple as the 5MR reconfiguration switching mechanism. In fact, the complexity of the switching system is a function of N. Hence, as N increases, the reliability of the NMR reconfiguration scheme decreases.

Comparison of redundancy systems

All five redundancy approaches have their own advantages and disadvantages. There always will be some sort of trade-off in selecting one scheme over another. McCluskey, Wakerly and Ogus[6] offer three guidelines for choosing a particular technique:

(1) For extremely short mission times static redundancy is optimal.
(2) For mission times whose duration is of the same order of magnitude as non-redundant system mean life, self-purging redundancy provides the best performance.
(3) For very long mission times, hybrid redundancy is best, unless unpowered modules have the same failure rate as powered modules—in which case self-purging redundancy should be used.

McCluskey, Wakerly and Ogus did not analyze the reconfiguration scheme, so a fourth guideline could be added to the list:

(4) For systems where multiple-faults are likely or where hybrid redundancy is used, the reconfiguration scheme may be optimal.

In the next two sections, error correcting codes and software systems are briefly discussed as alternatives to a hardware fault-tolerant system. In practice, however, it is probably best to use them in conjunction with one of the hardware redundant schemes.

ERROR CORRECTING CODES

Error correcting codes refer to a set of transformations on the digital representation of data and an associated set of checking and correcting algorithms. The transformation could be adding redundant bits or actually changing the form of the data depending upon the nature of the checking and correcting algorithms.[15] These codes provide concurrent fault diagnosis and correction. In a sense, they perform the same masking function of static redundancy with, however, some loss in processor time. There are a number of such codes and reviewing them all is beyond the scope of this paper. Instead a few brief comments on the general nature of error correcting codes will be made.

The encoding process introduces redundant bits, requiring additional hardware in the form of longer word lengths. This also requires additional hardware in the memory, processor, and data transfer systems. In addition, the encoding process is performed by a given algorithm which may be fairly complex. Thus the encoding process increases processor time and hardware requirements.

The checking and correction system also requires additional hardware and computing time. Also there is some concern, as with the switching systems of redundancy schemes, for the reliability of the checking and correction hardware. An error in this hardware would be catastrophic for the fault-tolerant abilities of the system.

Thus, the error correcting code process increases the hardware requirements of the system as well as slowing down the computing speed of the system. The speed of the system is decreased since it now has to convert data to the coded form and reconvert it on output as well as check the code for errors. These factors must be considered in comparing error correcting codes to hardware redundancy processes. Yet, these codes are very useful in providing concurrent correction along data paths, especially between peripheral units and the main systems.

SOFTWARE FAULT-TOLERANCE

Another approach to fault-tolerant computing is to build the fault-handling capabilities into the software. In other words, the system software operates in such a manner as to allow the system to recover from hardware failures. Such a software system works best against transient faults. There are two advantages to a software approach to fault-tolerance. First, fault-tolerant capabilities can be placed in the system after the hardware has been designed. Second, the fault-tolerant capabilities of the system are easily changed at a later date. No hardware rewiring is required, the software is just rewritten to allow for changes in fault-tolerant needs. On the other hand, there are several disadvantages to software fault-tolerance. Assuring that the software will operate correctly in the presence of a hardware fault is the major one. In addition, software now represents a much larger percentage of total system cost than hardware. The development costs of additional software for fault-tolerance could be excessive. The additional software would also require extra storage and in some cases redundant storage.

The major software fault-tolerant approach is called rollback and recovery.[16] Sometimes called time redundancy, this system involves some sort of fault detection procedure and program restarts after a fault occurs. In its simplest form, rollback and recovery involves just instruction retry. If a fault is detected after an instruction is executed then the instruction is reexecuted. More complex forms of rollback and recovery require checkpointing. A checkpoint is a point in time when the system stops its normal processing of user jobs and saves all relevant information in the current state of the system. A checkpoint may be static in which the checkpoints occur at the same time every day or it may be dynamic in which the checkpoints occur at different times dependent on system load and other factors. If any error is detected, the system rolls back and starts reprocessing everything done since the last checkpoint.

There are several questions which need to be considered before installing a rollback and recovery system. For example, how many checkpoints are needed? If too many checkpoints are used, the availability of the system decreases since it has to stop processing the normal flow of jobs to complete the checkpoint. If the checkpoints are too far apart, then the recovery time will be high since the system has to reprocess everything since the last checkpoint. Chandy[16] has reviewed several rollback and recovery models and has calculated the optimum intercheckpoint time which depends on the use of the system. O'Brien[17] has also studied a number of different checkpoint insertion strategies.

It is important to remember that with software fault-tolerance, a single permanent hardware fault could cause the system to crash. Hence, software fault-tolerance works best in a hardware fault-tolerant environment as a supplemental system used to improve fault-tolerant reliability. In addition, software fault-tolerant systems work especially well in providing individual protection to large data bases.

NEW APPROACHES TO FAULT-TOLERANT DESIGN

In the last few years a large research effort in fault-tolerant design techniques has been going on.[18] This research has led to the development of a number of promising alternatives to the standard fault-tolerant design procedures already mentioned. This section will briefly examine several of these alternatives.

Bi-duplexed systems

Courtois[19] has suggested that safety may, in some cases, be a more important concept than reliability. Safety is the probability of not sending incorrect data. This is especially true in biomedical systems. In such systems it may be better for a computer to stop operating rather than to take an incorrect action or give out incorrect data on a patient. Courtois found that a bi-duplexed (BDR) redundant system has better safety than hybrid systems and it is less complex than hybrid systems. The BDR operates with 4 units whose outputs pass through exclusive-OR gates which stop the operation of the incorrect modules. In Figure 4, the stop signal is generated only if both exclusive-OR gates are 1. Otherwise, the output box selects the output from the modules which have an exclusive-OR output equal to one.

Alternating logic

Systems designed using alternating logic design proposed by Reynolds and Metze[20] possess built-in fault detection capabilities. Hence, such systems, while not in and of themselves fault-tolerant, could easily fit into dynamic redundancy or any of the other redundancy schemes already reviewed. The fault detection is attained through a

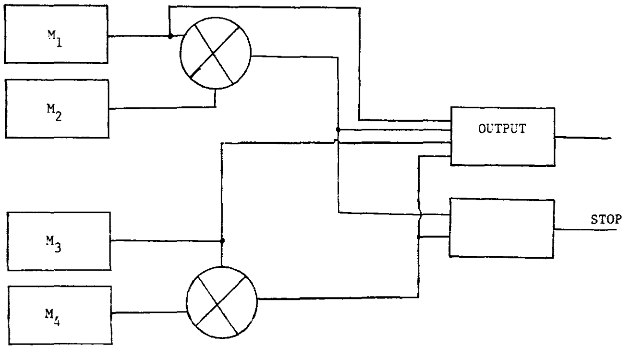


Figure 4—Bi-duplexed system

redundancy in time by successive execution of the function and its dual. Reynolds and Metze have shown that any combinational circuit can be realized using alternating logic design techniques. Since every input to a circuit is an alternating binary sequence of the form $(d,\bar{d})$, the circuit operation is slower. It must process both d and $\bar{d}$ which represent the same information. A fault is detected if the output is not an alternating sequence.

Fail-soft

A concern for high availability of a computing system prompted the development of fail-soft approaches to fault-tolerance. A fault in a fail-soft (graceful degradation) system results in a reduction in system performance. The system remains available to users. An example of a fail-soft system is the PRIME computer system.[21] PRIME has a distributed architecture with four module types: intelligence modules, interconnection modules, memory modules and storage modules.[22]

The University of California at Irvine also has a fail-soft system called the Distributed Computing System (DCS).[23] The DCS distributes hardware, software, and control of the system over a network. The hardware is in the form of a ring of processors (modules) with a software nucleus in each processor. The rest of the software is spread throughout the system. This DCS can tolerate a module failure and continue operation at a lower performance level, that is with fewer processors. Figure 5 shows a DCS with six processors.

Shared logic

For TMR systems to operate correctly the assumption of fault independence is important. This assumption is violated if any of the modules in a TMR share the same logic gates. Hence, three complete copies of each module must be constructed. Osman and Weiss,[24] however, have developed a means of allowing modules to share the same logic called (n,m,r)-basis realizations. They found that a TMR implemented with logic sharing using (n,m,r)-basis realizations would retain the same fault-tolerant characteristics. Yet, the logic sharing means that three copies of each module do not have to be constructed. This results in a considerable savings in hardware.

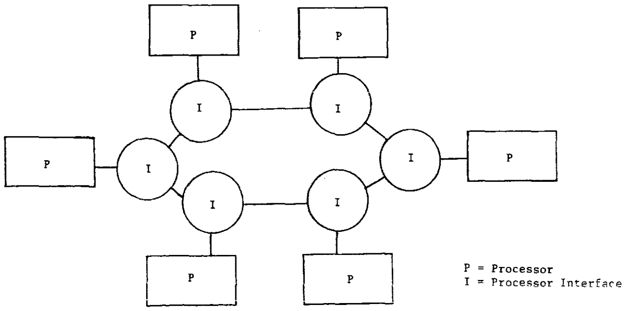


Figure 5—A DCS with six processors

Fail-safe

In the discussion of the bi-duplexed system the concept of safety was introduced in which a system will stop operating rather than generate incorrect data. Fail-safe systems operate under a similar concept. They take into account the fact that an incorrect output of one value may be worse than an incorrect output of another value. That is, the damage caused by outputting an incorrect 1 may be greater than the damage caused by outputting an incorrect 0. In such a case a fail-safe system would produce a 0 output whenever a failure occurs. Hence, an output of 1 would always be correct, an output of 0 may or may not be correct. Such a system is called a 0 fail-safe system.[27] A 1 fail-safe system could be defined in a similar manner. Fail-safe systems are important in biomedical applications where an error resulting in an unnecessary call for a doctor's assistance is not as harmful as not calling for a doctor when the patient has an urgent need for medical treatment. Also, in military systems, an incorrect signal resulting in the launching of a missile can do more damage than the failure to launch a missile when needed. Fail-safe systems for biomedical and military applications would be designed such that if a failure occurs the least harmful output is produced. Hence, a fail-safe system is defined as a system that produces safe-side outputs when failures occur.[28]

Fail-safe systems are subject to problems under multiple fault conditions. A "masked" fault could occur which is a fault that does not affect the output but which could break the fail-safeness of the circuit when another fault occurs.[29] Mukai and Tohma[29] have developed a method of designing asynchronous circuits in which "masked" faults do not occur. In addition, Chuang[30] has also developed a new fail-safe asynchronous design procedure. Chuang's system can handle multiple-input changes which ordinarily could cause a fail-safe machine to produce non-safe-side outputs under certain conditions.

A fail-safe system usually requires twice as much hardware as a non-fail-safe system. This raises some questions on circuit reliability since the more components in a system the higher the chance of a circuit failure. However, Swain[31] has examined this problem and produced a design procedure that will reduce the amount of hardware required to achieve a fail-safe system.

CONCLUSIONS

The need for fault-tolerant computing cannot be understated, especially in light of the role computers now play in medicine, space, and other highly critical areas. This paper has briefly reviewed several methods of achieving fault-tolerance. They all have their own advantages and disadvantages. Perhaps the best overall approach to a truly fault-tolerant system would be some combination of the techniques reviewed in this paper.

In fact, several computing systems have been proposed or are in the process of being constructed which utilize a combination of the fault-tolerant procedures suggested in this paper. For example, the STAR (Self Testing And Repairing) computer at the Jet Propulsion Laboratory uses the following methods of tolerance:[32]

(1) Dynamic redundancy technique with unpowered spares is used. Replacement is implemented by power switching.
(2) Error correcting codes are used for all data and instruction words with concurrent fault detection.
(3) Fault detection, recovery and replacement are achieved with special purpose hardware.
(4) Software recovery techniques are used in the event of a memory failure.
(5) The processor is protected with hybrid redundancy with three active units.

The design of the STAR computer is constantly being updated but it always employes several different fault-tolerant techniques to achieve optimum fault-tolerant performance. Hopkins[33,34] has proposed a fault-tolerant computer for space vehicles which also uses several fault-tolerant techniques. He suggests a multi-processor system for fail-soft behavior. Each processor unit consists of duplicated processors for error detection plus a triplicated scratch pad memory. The memory unit is protected with an error detection and correction code. Several different redundancy techniques are also used in the system.

Wensley, Levitt, and Neumann,[35] after studying several fault-tolerant architectures, concluded that it is economically feasible to build systems using several different fault-tolerant techniques and achieve a mean time between failures greater than 10 years. For the same cost as duplication they also concluded that it was possible to achieve a mean time between failures in excess of 100 years. Yet, they did find significant deficiencies in the state-of-the-art of fault-tolerant design. Hence, additional research into fault-tolerant computing is vital. Several areas need further investigation. For example, hardware redundancy systems will reject a module even if the detected fault was only a transient. Retry procedures need to be developed so that otherwise "good" modules (modules with just transient faults) will not be switched out of redundant circuits. Goldberg[25] has suggested several other areas for future research. Such as the study of systems that can tolerate faulty human operators, or the application of artificial intelligence to achieve highly flexible fault-tolerant systems.

REFERENCES

1. Avizienis, A. "Fault tolerant computing-An overview", *IEEE Trans. Compt.*, vol. 4, January 1971, pp. 5-8.
2. Avizienis, A. "Architecture of fault-tolerant computing systems," *Digest of the Fifth International Symposium on Fault-Tolerant Computing*, Paris, France, June 1975, pp. 3-16.
3. Avizienis, A. "Design of Fault-Tolerant Computers," *AFIPS Conference Proceedings*, Vol. 31, pp. 733-743, AFIPS Press, Montvale, New Jersey, 1967.
4. Carter, W. C. and W. G. Bouricius, "A Survey of Fault-Tolerant Computer Architecture and Its Evaluation," pp. 9-16, *Computer*, Jan/Feb. 1971.
5. von Neumann, J. "Probabilistic logics and the synthesis of reliable organisms from unreliable components," *Automata Studies*, C. E. Shannon and J. McCarthy, eds., pp. 43-98, Princeton University Press, New Jersey, 1956.
6. McCluskey, E. J., J. F. Wakerly and R. C. Ogus, "*Center for reliable computing: Current Research,*" Stanford Digital Systems Lab Technical Report 100, Oct. 1975.
7. Chandy, K. M., C. V. Ramamoorthy and A. Cowan, "A framework for hardware-software tradeoffs in the design of fault-tolerant computers," *AFIPS Conference Proceedings*, Vol. 41, pp. 55-63, AFIPS Press, Montvale, New Jersey, 1972.
8. Sieworek, D. P. and E. J. McCluskey, "An Iterative Cell Switch Design For Hybrid Redundancy," *IEEE Trans. Compt.*, Vol. c-22, March 1973, pp. 290-297.
9. Ramamoorthy, C. V. and Y. Han, "Reliability analysis of systems with concurrent error detection," *IEEE Trans. Compt.*, Vol. c-24, Sept. 1975, pp. 868-878.
10. Mathur, F. P. and A. Avizienis, "Reliability analysis and architecture of a hybrid-redundant digital system: Generalized triple modular redundancy with self-repair," *AFIPS Conference Proceedings*, Vol. 36, pp. 375-383, AFIPS Press, Montvale, New Jersey, 1970.
11. Cochi, B. "Reliability modeling and analysis of hybrid redundancy," *Digest of the Fifth International Symposium on Fault-Tolerant Computing*, pp. 75-80, Paris France, June 1975.
12. Losq, J. "A Highly Efficient Redundancy Scheme: Self-purging redundancy," *IEEE Trans. Compt.*, Vol. c-25, June 1976, pp. 569-578.
13. Losq, J. "Redundancy scheme for optimum multiple fault-tolerance," *Technical Note no. 33*, Digital Systems Laboratory, Stanford University, Stanford California, Jan. 1974.
14. Su, S. Y. H. and E. DuCasse, "A reconfiguration scheme for tolerating multiple failures in digital systems," *Proceedings of International Computer Symposium*, 1975, pp. 216-222.
15. Avizienis, A. "Arithmetic error codes: Cost and effectiveness studies for application in digital system design," *IEEE Trans. Compt.*, Vol. c-20, Nov. 1971, pp. 1322-1331.
16. Chandy, K. M. "A survey of analytic models of rollback and recovery strategies," *Computer*, May 1975, pp. 40-47.
17. O'Brien, F. "Rollback point insertion strategies," *Proceedings of The Sixth International Symposium on Fault-Tolerant Computing*, Pittsburgh, Pennsylvania, June 1976, pp. 138-142.
18. "Special issues on Fault-Tolerant Computing." *IEEE Trans. Compt* Nov. 1971, March 1973, July 1974, May 1975, June 1976.
19. Courtois, B. "On Balancing Safety and Reliability of Hybrid and Biduplexed systems," *Proceedings of the Sixth International Symposium on Fault-Tolerant Computing*, Pittsburgh, Pennsylvania, June 1976, pp. 53-57.
20. Reynolds, D. and G. Metze, "Fault Detection Capabilities of Alternating Logic," *Proceedings of the Sixth International Symposium on Fault-Tolerant Computing*, Pittsburgh, Pennsylvania, June 1976, pp. 157-162.
21. Baskin, H. B., B. R. Borgerson and R. Roberts, "PRIME—A modular architecture for terminal-oriented systems," *Proceedings of the Spring Joint Computer Conf.*, 1972, pp. 431-437.
22. Borgerson, B. and R. Freitus, "A reliability model for gracefully degrading and stand by-sparing systems," *IEEE Trans. Compt.*, c-24, May 1975, pp. 517-525.
23. Rowe, L., M. Hopwood and D. Farber, "Software methods for achieving fail-soft behavior in the distributed computing system," *Record of the 1973 Symposium on Computer Software Reliability*, New York, April 1973, pp. 7-11.
24. Osman, M. and C. Weiss, "Shared logic realizations of dynamically self-checked and fault-tolerant logic," *IEEE Trans. Compt.*, Vol. c-22, March 1973, pp. 298-306.
25. Goldberg, J. "New problems in fault-tolerant computing," *Digest of the*

Fifth International Symposium on Fault-Tolerant Computing, Paris, France, June 1975, pp. 29-34.

26. Ogus, R. "Fault-tolerance of the iterative cell array switch for hybrid redundancy," *Digest of the Third International Symposium on Fault-Tolerant Computing,* Palo Alto, California, June 1973, pp. 107-112.
27. Mine, H. and Y. Koga, "Basic properties and a construction method for fail-safe logical systems," *IEEE Trans. Compt.,* Vol. EC-16, June 1967, pp. 282-289.
28. Tohma, Y., Y. Ohyama and R. Sakai, "Realization of fail-safe sequential machines by using a k-out-of-n code," *IEEE Trans. Compt.,* Vol. C-20, November 1971, pp. 1270-1275.
29. Mukai, Y. and Y. Tohma, "A masked-fault-free realization of fail-safe asynchronous sequential circuits," *Proceedings of the Sixth International Symposium on Fault-Tolerant Computing,* Pittsburgh, Pennsylvania, June 1976, pp. 69-74.
30. Chuang, H. Y. H. "Fail-safe asynchronous machines with multiple-input changes," *IEEETC.* Vol. C-25, June 1976, pp. 637-642.
31. Swain, D. H. "Fail-safe synchronous sequential machines using modified on-set realizations," *Digest of the Fourth International Symposium on Fault-Tolerant Computing,* pp. (3-7)-(3-12), Urbana, Illinois, June 1974.
32. Avizienis, A., G. C. Gilley, F. P. Mathur, D. A. Rennels, J. A. Rohr, and D. K. Rubin, "The STAR (Self Testing And Repairing) computer: An investigation of the theory and practice of fault-tolerant computer design," *IEEE Trans. Compt.,* Vol. C-20, November 1971, pp. 1312-1321.
33. Hopkins, A. "A fault-tolerant information processing concept for space vehicles," *IEEE Trans. Compt.,* Vol. C-20, November 1971, pp. 1394-1403.
34. Hopkins, A. and T. B. Smith, "The architectural elements of a symmetric fault-tolerant multiprocessor," *IEEE Trans. Compt.,* Vol. C-24, May 1975, pp. 498-504.
35. Wensley, J. H., K. N. Levitt, and P. G. Neumann, "A comparative study of architectures for fault-tolerance," *Digest of the Fourth International Symposium on Fault-Tolerant Computing,* Urbana, Illinois, June 1974, pp. (4-16)-(4-21).

The use of passwords for controlling access to remote computer systems and services*

by HELEN M. WOOD
National Bureau of Standards
Washington, DC

ABSTRACT

The widespread use of remote computer resources has made the problem of personal authentication most urgent. This paper examines the use of passwords for controlled access to these resources. Password techniques, ways of protecting passwords, and attendant cost considerations are discussed. Similarities between passwords and data encryption keys are noted and general recommendations for the use of passwords are presented.

INTRODUCTION

With the growth of timesharing and other forms of computer networking, the use of remote computers and their resources has become widespread. However, with this ease of access have come increased operational risks.

Systems without adequate access controls are vulnerable to threats including theft, fraud, and vandalism. Potential losses range from unauthorized use of computing time to unauthorized modification or access to massive amounts of data. Perpetrators of such abuse may be otherwise honest individuals wishing to play a few computer games, or ~~sophisticated corporate spies, hoping to learn trade secrets~~ or acquire the list of a competitor's top ten accounts. (See Reference 1 for a more complete treatment of computer abuse perpetrators.) Current privacy legislation and increased public concern with the integrity and protection of data in computer systems have made the problem of personal authentication most urgent.

AUTHENTICATION

Typically when a user wishes to access resources on a remote computer system, he or she states a claimed identity, perhaps through typing a user identification number. The user is then required to authenticate the claimed identity. This latter process is referred to as personal authentication.

There are three basic methods by which a person's identity may be established for the purpose of allowing access to a remote computer system:

- something the person *knows*
- something the person *has*
- something the person *is*.

The first category includes techniques and items such as passwords and lock combinations. Badges, ID cards, and keys are among the items falling into the second category; while "something a person *is*" includes physical and behavioral characteristics such as one's appearance, voice, fingerprints, signature, and hand geometry. The advantages and limitations of these types of authentication techniques have been discussed extensively elsewhere.[2–4]

The actual authentication techniques selected for a given system should be determined by a risk analysis. This is a consideration of potential threats,[5] the probability of these threats occurring, and the expected losses resulting from a successful penetration of the system.

Password systems cost less at present than many of the other techniques for personal authentication. Consequently, it appears that passwords, perhaps in combination with other techniques such as badges or keys, will continue to be heavily utilized for some time.

The technique of using passwords to authenticate a user to a resource sharing computer system is well-known. However, the simple use of passwords is not sufficient to guarantee system security from unauthorized users. Cotton and Meissner suggest that passwords for most timesharing systems are "notoriously easy to obtain."[2] There are, however, a number of ways to improve the resistance of password-based security systems to penetration. The intent of this paper is to consider the generation of passwords and their effective application to the problem of controlling access to computer resources.

* This work is a contribution of the National Bureau of Standards and is not subject to copyright. Certain commercial products are identified in this paper in order to adequately specify the procedure. In no case does such identification imply recommendation or endorsement by the National Bureau of Standards, nor does it imply that the material identified is necessarily the best available for the purpose.

USES OF PASSWORDS

Personal authentication may be required at any number of points along the path to accessing data. Such points include

- entry to building
- entry to terminal room
- enabling terminal
- encryption interface unit
- login
- file access
- data item access

Physical devices (e.g., cards, keys) are commonly used at the first three access points. While passwords, alone or in conjunction with other techniques, may be used at any of these points, the primary emphasis of this paper is the use of passwords in an on-line environment. Thus we shall only consider the utilization of passwords at login, file access, or data item access time.

Data encryption keys and the banking community's Personal Identification Number (PIN) are equivalent to passwords. An encryption key controls the algorithmic transformation (encryption) performed on data symbol-by-symbol. The PIN is typically a four-to-six-digit number assigned by the bank or selected by the cardholder. It is used in conjunction with a magnetically encoded card. Throughout this paper analogies will be drawn among encryption keys, PIN's, and passwords.

PASSWORD SCHEMES

Password schemes differ according to

- selection technique
- lifetime
- physical characteristics
- information content

In this section the types of password systems are discussed along with the threats they are most effective against. Examples are presented.

Password selection

A password may be chosen by the system user or assigned. User-selected passwords usually are less secure since people tend to pick words or numbers that have some personal meaning (e.g., birthday, child's name, street address) and consequently are easy to guess.[6] Of course the primary advantage of a user-chosen password is ease of recall, alleviating the need for writing the password down.

Passwords may be assigned to users by the system security officer or by the computer system itself. Although they are generally more secure than user-selected codes, the benefits of assigned passwords may be nullified if they are written down by the user, taken from a master list which is discovered,[7] or generated by an algorithm that is deducible.[8]

Johnson examined the use of pseudorandom numbers as passwords and discovered that various "logistically attractive" periodic password generation systems are in fact vulnerable to simple number-theoretic analysis. The generating systems he considered were of the type

$$x_{n+1}=ax_n+b(\mathrm{mod}\ 2^u), \quad u\cong 40,$$

where a and b are selected constants and x_n is the nth password generated. This type of generating system would be considered attractive, for example, in a large databank system in which it is not practical to use complex password schemes.[8]

Another example of a computer-generated password scheme is the random word generator developed to run on Honeywell's Multiplexed Information and Computer System (Multics).[9] The random word generator forms pronounceable syllables and concatenates them to create words. This system was developed to enhance the security of some Multics installations, such as the Air Force Data Services Center (AFDSC), where classified information is processed. Installations such as these cannot risk the compromise of users' passwords.

In his chapter on principles of computer security, Bushkin includes the following principle:

> No passwords or other user authentication data shall have been or shall be created or generated either by the human user who will use them or by a non-human agent (e.g., a program) of his creation or under his control . . .

Although it is not claimed that such a principle applies to each and every system, it is apparent that such nonhuman generation of passwords is the preferred method for enhanced system security.[10]

Password lifetime

Current password schemes allow password assignments to be used for an indefinite period of time, for fixed intervals of time (e.g., one month), or for a single use only. The length of time that a password remains in effect is called the password lifetime or period.

Passwords that remain effective indefinitely are the most susceptible to compromise. Due to the length of the password period, these passwords are especially vulnerable to exhaustive testing. Making the password appropriately long, locking-out log-on attempts after several (e.g., three) tries,[11] and enforcing time delays between log-on attempts are some deterrents against exhaustive enumeration attempts.[12]

Another shortcoming of passwords with indefinite lifetimes is the difficulty in detecting a successful compromise of the password. Some systems prohibit a user from being logged onto the system from more than one terminal at a time.[6] However, even if such system constraints are present, the odds of a system penetrator and the legitimate user attempting to use the same account at the same time depend upon the frequency of access of each. Thus, when fixed passwords of indefinite lifetime are used, a masquerader can penetrate another user's files over a long period of time with a low probability of detection.

Obviously more frequent password changes are desirable.[7,13] An example of a system which requires password updates at fixed intervals of time is the Air Force Data Security Center. In this system, users are required to change their passwords every six months. The enforcing mechanism is the operating system.

Passwords that can only be used once are recognized as generally providing a higher level of protection.[12-14] Successive passwords may be selected by the system from an internal list,[12] generated by a program,[8,9,15] or selected from lists or cards previously distributed to authorized users.[6,14]

Anderson[13,16] suggests an open, one-time password scheme. He contends that if passwords are changed each time they are used there is "no more risk in writing down the password than in carrying a key to a locked room." Should loss or theft occur, prompt reporting would minimize the risks involved.

As a means of further reducing the risk of carrying a password openly, Anderson mentions the possibility of encoding the new password on a magnetic card.[13] The feasibility of this technique was investigated by Richardson and Potter.[17] The major disadvantage of such a technique is the cost of the magnetic card reader/writer.

Major drawbacks to the use of one-time passwords are the cost and difficulty associated with the distribution of lists to large numbers of users[16] and with the support of users who get "out of step" in a system with a heavy workload.[6]

It has been noted by Petersen and Turn that one-time password schemes alone are not effective against the threat of between-lines or piggyback entry. For protection against these threats, message authentication via attachment of one-time passwords to each message would be required. Encryption at the terminal level is also an effective protection mechanism in this situation.[5]

Physical characteristics

Physical characteristics refer to the size of a password and its makeup (i.e., the "alphabet" from which it is made). The number of different passwords possible in a given scheme is called the password space. For example, given a password of length L that is formed using any of the 26 letters in the English alphabet, there are 26**L possible words that could be generated. Enlarging the alphabet increases the password space accordingly. Including nonprinting characters in the alphabet not only increases the password space, but provides some additional protection.[11] Of course, when conditions such as pronounceability are added to the scheme, then some fraction of the total number of possible words would comprise the password space.

Meissner[4] emphasizes that, in order to adequately assess the security of a given password scheme, one must consider the number of *allowable* combinations for valid passwords, rather than simply the theoretical number of combinations based upon the size of the alphabet and the generated password.

In Reference 13, Anderson considers passwords generated as random strings of letters or numbers. He presents a formula for determining the random password length required to provide a given degree of protection against systematic testing. The assumption is that tests occur at the maximum line transmission rate, as would be the case if another computer were attempting penetration by exhaustive enumeration. In his formula, the password size is found by solving

$$(R/E)4.39\times 10^4(M/P)\leq A^S \tag{1}$$

for S, where S is the password size in characters. Here, R is the transmission rate of the line in characters per minute, E is the number of characters exchanged in a log-on attempt, P is the probability that a proper password will be found, M is the period over which the systematic testing is to take place, and A is the size of the alphabet from which the password is made.

As an example, Anderson determines the password size drawn from the English alphabet that gives a probability of no more than 1/1000 of recovery after three months of systematic testing. He assumes a line speed of 300 characters/minute, and an exchange of 100 characters during a log-on attempt. The computation is as follows:

$$\frac{300}{100}\times 4.39\times 10^4\times 3\times 10^3\leq 26^S \tag{2}$$

$$3.951\times 10^8\leq 26^S \tag{3}$$

$$26^S=3.089\times 10^8 \quad \text{for} \quad S=6 \tag{4}$$

$$26^S=8.03\ \times 10^9 \quad \text{for} \quad S=7 \tag{5}$$

Therefore, in this example S=7 is the reasonable choice. Note that increasing the alphabet to 128 characters (e.g., for 7-bit ASCII) reduces S to 5.

Information content

The password may provide information in addition to personal authentication. The University of Western Ontario's generalized information retrieval sytem (GIRS) incorporates the use of assigned, functional passwords whose contents reveal the users' authorization levels.[18] In this system an additional password is needed to effect the initial log-on to the computer system. The functional password is only used by the information retrieval system.

Besides imparting authorization information, it has been suggested that passwords could be constructed to contain check digits or some other sort of self-checking code. "Check digitry" is already being successfully used in other environments, as discussed in a series of articles by Alan Taylor.[19-21] Techniques such as these, combined with some elementary analysis, could help more sophisticated password systems discriminate between entry-errors (such as transpositions of digits) and actual penetration attempts, especially attempts via exhaustive testing.

This idea is similar to that embodied in Kaufman and Auerbach's general model of an electronic funds transfer (EFT) system. Their system incorporates the use of cryptographic check digits derived *from* the PIN.[22]

Other schemes

Many of the disadvantages associated with password systems are minimized or cease to exist altogether if authentication is accomplished via the successful execution of an algorithm. Such procedures are often referred to as "handshaking" or "extended handshakes."[6,23] Some of these procedures directly involve the use of passwords; others can only marginally be considered password schemes.

The ADEPT-50 time-sharing system incorporates a handshaking scheme.[12] In order to gain admittance to the system, the user must supply information items including user identification, passwords, and accounting data. The terminal identification is also compared against the terminal id list for which the user was franchised.

In several systems handshaking is accomplished by a dialog between the system and the user. In such procedures the user may be required to answer personal questions (e.g., child's name, brand of mouthwash) asked in a semi-random fashion, or to supply additional passwords and/or account information.[24]

In another variation, credited to Les Earnest by Hoffman,[25] the system presents the user with a pseudorandom number and requires that the user perform a mental transformation T on that number. The result is then sent back to the computer, which performs an appropriate transformation and compares the results. Thus, the user has performed T on a number x and transmitted y=T(x). Consequently, an eavesdropper monitoring the transmission would at most see x and y.

Hoffman asserts that even simple T's such as

$$T(x)=[(\sum_{i \text{ odd}} \text{digit } i \text{ of } x)^{3/2}]+(\text{hour of the day})$$

raise the work factor in breakinฺg the scheme significantly.

PASSWORD PROTECTION

The previous section has been concerned with the selection of a password scheme that, in addition to being convenient to use, is secure from discovery through guessing or exhaustive enumeration. However, regardless of the password scheme implemented, protection of the password is vital.

The three times during which the password must be protected, are during

- initial distribution
- storage
- transmission.

In this section we shall address the requirements for guarding the passwords against potential threats that might occur at such times.

Initial distribution

The initial distribution of passwords to users is a special case of password assignment, selection, and transmission. Two items must be considered in this situation:

- user identification
- distribution method.

It is usually the practice that first-time users of a system make application in person for authorization to to use the system resources. At that time a temporary password, assigned by the system security officer, can be given to the user. The user then has the responsibility for logging onto the system and changing the password to one known only to him.

In another form of password distribution, more useful when users are great distances from the computing facility, the password is transmitted by mail to the user. PIN's are normally distributed in this manner. If more assurance of receipt is required, registered mail or special messengers can be used.

Initial distribution of encryption keys could be handled in a similar manner, with the magnetic stripe card bearing the first key being sent via registered mail.

Password storage

Passwords may be stored in the computer system (e.g., on a disk), or by users (e.g., on paper or magnetically encoded on a card). Dangers inherent in writing down passwords have already been addressed. The storage of passwords on the system and on cards is considered in this section.

Most password schemes employ the use of tables or lists which contain the current password for each authorized system user. (A notable exception would be the user-transformation scheme described above.)[25] As these tables and lists are perhaps the most vulnerable part of a password system, efforts should be taken to protect them. Internal lists from which one-time passwords are assigned should likewise be guarded.

R. M. Needham is credited with being the first to recognize the vulnerability of password lists. An encipherment algorithm attributed to him has been implemented at Cambridge, England. The cipher produced by this algorithm is a "one-way cipher." This is a cipher for which no simple deciphering algorithm exists. In such a scheme, the user's password is encrypted as soon as it is received by the system, and the transformed password is then compared with the encoded table entry.[26]

A discussion of Needham's system and the merits of various others can be found in Reference 27. Purdy[28] also describes the Needham scheme, discusses the selection of good one-way ciphers, and suggests the use of polynomials over a prime modulus.

There are still potential threats involved in such schemes. One is the interception of passwords prior to encryption, and another is the selection of a poor cipher. The first problem will be dealt with in the next section.

An example of a poor cipher would be one that is highly degenerate (i.e., one in which many combinations encrypt to the same value).[29] Under such a scheme the simple exposure of the encrypted list could give enough information to a would-be penetrator to allow him to, if not break the algorithm, at least access the files of any users whose passwords in their encrypted form were identical to his.

As a part of their Multics vulnerability analysis the Air Force considered the threat of exposure of password files.[30] Their report asserts that accessing the system password file is of minimal value to a system penetrator. Assuming that the password file is the most highly protected file in the system, anyone who succeeded in accessing this file could conceivably penetrate *any* other file in the system! Furthermore, if the password list were enciphered, then it would be much easier to simply ignore it than to attempt to decode it.

For completeness the Air Force study did analyze the "non-invertible" encipherment scheme used at that time by the Multics system. In a report soon to be published the details of their successful penetration of that scheme will be detailed.[31]

In some systems using magnetic stripe cards the PIN itself is stored on the card in an encrypted form. When discussing the threats to bank card systems presented by the underworld,[32] Industrial National Bank Vice President Ernest Northup described the components of a card-based electronic funds transfer system and noted that the "use of a standard PIN scrambling technique or algorithm for bank interchange would require that its elements be widely known, at least among equipment vendors. This increases its vulnerability."

In Reference 22, Kaufman and Auerbach present a comprehensive set of security principles for EFT systems. Concerning the storage of PIN's they state that "there should be no way to derive the PIN from information on the card," although they observe that many current schemes are based upon such risky techniques.

Password transmission

Passwords are vulnerable to several threats during their transmission from terminal to computer. Potential threats include wiretapping, electronic eavesdropping, and piggyback infiltration. The password may also be discovered later in the trash if a hardcopy terminal was used, or observed on a CRT screen immediately after entry. These latter two problems are usually dealt with by masking (the over-printing or under-printing of a series of characters) or echo-suppression. However, as pointed out by Carroll and McLellan,[33] in general the "use of a mask affords no protection to users on CRT visual display terminals." Another method sometimes used incorporates non-printing characters as a part or all of the password.[4,11]

In a discussion of piggyback infiltration, Carroll and Reeves described a situation in which unsuspecting terminal users could be "exploited by a process which mimics the real system long enough to obtain a password. . . ."[34] Of course, echo-suppression and masking are of no help in countering this type of threat. If a more intelligent device than a terminal is used to intercept the conversation, then non-printing characters also lose their effectiveness.

The user-transformation schemes described by Hoffman[25] and Carroll[35] are one way of effectively shielding the password in transit. Here the user, when presented with a random number, performs a pre-determined transformation on it and transmits the result back to the computer for verification. The incorporation of a date-time group into this transformation is recommended to provide additional protection against piggyback infiltration.[35]

Optimal protection of the transmitted password, as with any data, can be realized by encryption of the communications link during the entire conversation.[15,36] The NBS encryption algorithm would be suitable for this purpose.[37]

Brandstad notes that encryption keys and authentication codes may be in effect the same item. In his proposed network access control machine, these keys are never transmitted through the network, but rather are loaded simultaneously by interface units into a primary encryption device. Thus, authentication can be considered complete at that level (at least) if a message can be encrypted, transmitted, and correctly decrypted.[36,38]

In a recent master's thesis on encryption-based protection protocols, Kent addresses encryption key distribution protocols.[39] He identifies two basic transmission techniques:

- chained key changes
- two-level key distribution systems.

Under the chained key system, each new key is enciphered using the last key issued. This new key is then used until another change occurs. Under the two-level distribution system, a special key is used solely for transmitting new keys to remote users. Kent describes protocols for using these schemes and considers the use of magnetic stripe cards for distribution of keys.

COST CONSIDERATIONS

The costs of a given password scheme are those incurred

- by the protector
- by the intruder.

These costs must be considered in conjunction with the value of the information to be protected. (See Reference 40 for a discussion on the value of personal information in qualitative terms.)

The costs to the protector include not only the hardware and software costs involved, but also the effect on overall system performance. Processing time required and communications channel loading may result in severely degraded system response time.

Password schemes have been described which involve authentication to the file or data item level. Turn observes that the "costs of access control operations reflect themselves in increased processing time and storage space requirements."[41] He relates the results of a study of these costs which revealed a 22 to 140 percent processing time increase in file access operations, depending upon when access controls are applied (e.g., at file open time, or data item access time). Such implementation costs in a computer networking environment are considered in Reference 42.

The cost to the system intruder includes the investment in time and equipment (i.e., the work factor) necessary to determine the password or password-generating algorithm. Risk can also be considered part of the penetration cost.

As an example, consider the intruder's cost of acquiring passwords through wiretapping. These could range from the cost of recording equipment (a few hundred dollars), to the cost of a minicomputer and associated software development (several thousand dollars). Risks include possible legal prosecution.[40]

As aptly stated by Petersen and Turn,[5] "the level of work factor which is critical for a given information system depends, of course, on an estimate of the magnitude of threats and of the value of the information." They suggest that a work factor of one day of continuous computation required to break a single encryption key might be adequate against low-level threats.

Of course, the cost of the system utilized in the penetration effort must also be considered. For example, Diffie and Hellman have suggested a configuration consisting of 1,000,000 chips and associated controlling and power supply hardware costing around $20,000,000. They assert that such a system could search the key space of the proposed NBS encryption algorithm in about a day, assuming the possession of a matching block of encrypted and unencrypted text.[43]

CONCLUSIONS

Passwords can be a highly effective form of personal authentication when care is taken in their selection and protection. We have categorized the types of password schemes, identified their capabilities and limitations, and indicated the points at which password protection mechanisms are needed.

The exact password scheme appropriate for a given system depends, of course, upon the required level of security as determined by risk analysis. Cost is also a factor in the selection of the "right" password system.

Until other forms of personal authentication become more cost-effective, the password will remain the most widely used means of controlling access to remote computing systems and services.

REFERENCES

1. Parker, Donn B., "Computer Abuse Perpetrators and Vulnerabilities of Computer Systems," *Proceedings of the National Computer Conference*, AFIPS Press, Montvale, N.J., 1976, p. 65-73.
2. Cotton, Ira W. and Paul Meissner, "Approaches to Controlling Personal Access to Computer Terminals," *Proceedings of the 1975 Symposium Computer Networks: Trends and Applications*, IEEE Computer Society, 1975, p. 32-39, 19 refs.
3. Browne, Peter S., "Computer Security—A Survey," *Proceedings of the National Computer Conference*, AFIPS Press, Montvale, N.J., 1976, p. 53-63, 134 refs.
4. Meissner, Paul, *Guideline on Evaluation of Techniques for Automated Personal Identification*, National Bureau of Standards, Washington, D.C., 1977 [in press].
5. Petersen, H. E. and R. Turn, "System Implications of Information Privacy," *Proceedings of the Spring Joint Computer Conference*, AFIPS Press, Montvale, N.J., 1967, p. 291-300, 14 refs.
6. Beardsley, Charles W., "Is Your Computer Insecure?" *IEEE Spectrum*, January 1972, p. 67-78, 16 refs.
7. Winkler, Stanley, and Lee Danner, "Data Security in the Computer Communication Environment," *Computer*, February 1974, p. 23-31, 7 refs.
8. Johnson, S. M., *Certain Number Theoretic Questions in Access Control*, Rand Corporation, Report R-1494-NSF, January 1974.
9. Gasser, M., *A Random Word Generator for Pronounceable Passwords*, The MITRE Corporation, Bedford, Mass., AD-A017 676, November 1975, 183p., 3 refs.
10. Bushkin, Arthur A., *A Framework for Computer Security*, System Development Corporation, McLean, Va., AD-A025 356, June 1975, 158p.
11. Held, Gilbert, "Locking Intruders Out of a Network," *Executive Guide to Data Communications*, McGraw-Hill Publications Co., New York, 1976.
12. Weissman, C., "Security Controls in the ADEPT-50 Time-sharing System," *Proceedings of the Fall Joint Computer Conference*, AFIPS Press, 1969, p. 119-133, 20 refs.
13. Anderson, James P., "Information Security in a Multi-user Computer Environment," *Advances in Computers*, Vol. 12, 1972, Academic Press, Inc., New York, p. 1-36.
14. Peters, Bernard, "Security Considerations in a Multi-programmed Computer System," *Proceedings of the Spring Joint Computer Conference*, AFIPS Press, Montvale, N.J., 1967, p. 283-286.
15. Baran, Paul, *On Distributed Communications: IX. Security, Secrecy, and Tamper-free Considerations*, Rand Corporation, August 1964, AD-444 839, 39p.
16. Anderson, James P., *On Centralized Distribution of One-time Passwords in Resource Sharing Systems*, James P. Anderson and Co., Fort Washington, Pa., August 1971, 8p.
17. Richardson, Mark H. and James V. Potter, *Design of a Magnetic Card Modifiable Credential System Demonstration*, Electronic Systems Division (AFSC), Hanscom Field, Mass., MCI-73-3, December 1973, 65p.
18. Carroll, John M., Robert Martin, Lorine McHardy and Hans Moravec, "Multi-dimensional Security Program for a Generalized Information Retrieval System," *Proceedings of the Fall Joint Computer Conference*, Vol. 39, 1971, p. 571-577, 5 refs.
19. Taylor, Alan, "Darmstadt System Eliminates Check-Digit Loopholds," *Computerworld*, September 17, 1975, p. 13.
20. Taylor, Alan, "Deeds Check-Digit Method Possibly Valuable DP Tool," *Computerworld*, October 22, 1975, p. 11.
21. Taylor, Alan, "Statistics Improving State of Art in 'Check-Digitry'," *Computerworld*, February 23, 1976, p. 17.
22. Kaufman, D. and K. Auerbach, "A Secure National System for Electronic Funds Transfer," *Proceedings of the National Computer Conference*, AFIPS Press, 1976, p. 129-138, 6 refs.
23. Campaigne, Howard and Lance J. Hoffman, "Computer Privacy and Security," *Computers and Automation*, 22:7, July 1973, p. 12-17, 6 refs.
24. Lupton, William Lloyd, *A Study of Computer Based Data Security Techniques*, Naval Postgraduate School, Monterey, California, AD-765 677, 1973, 77p., 141 refs.
25. Hoffman, Lance J., "Computers and Privacy: A Survey," *Computing Surveys*, 1:2, June 1969, p. 85-103, 69 refs.

26. Wilkes, M. V., *Time Sharing Computer Systems,* American Elsevier, New York, 1975.
27. Evans, Arthur Jr. and William Kantrowitz, "A User Authentication Scheme Not Requiring Secrecy in the Computer," *Communications of the ACM,* 17:8, (August 1974), p. 437-442, 8 refs.
28. Purdy, George B., "A High Security Log-in Procedure," *Communications of the ACM,* 17:8, August 1974, p. 442-445, 8 refs.
29. Fletcher, J. G., *Software Security in Networks,* Lawrence Livermore Laboratory, University of California, 1975, 17p.
30. Karger, Paul A. and Roger R. Schell, *Multics Security Evaluation: Vulnerability Analysis,* Electronic Systems Division (AFSC), Hanscom AFB, Mass., ESD-TR-74-193, Vol. II, June 1974, 156p., 33 refs.
31. Downey, Peter J., *Multics Security Evaluation: Password and File Encryption Techniques,* Electronic Systems Division (AFSC), Hanscom AFB, Mass., ESD-TR-74-193, Vol. III, in preparation.
32. Northup, Ernest H., "Bank Cards Vs. the Underworld," *Banking,* 67:9, September 1975, p. 66, 68, 70, 73.
33. Carroll, John M. and P. M. McLelland, "The Data Security Environment of Canadian Resource-sharing Systems," *INFOR, Canadian Journal of Operational Research and Information Processing,* 9:1, March 1971, p. 58-67, 17 refs.
34. Carroll, John M. and Paul Reeves, "Security of Data Communications: A Realization of Piggyback Infiltration," *INFOR, Canadian Journal of Operational Research and Information Processing,* 11:3, (October 1973), p. 226-231, 2 refs.
35. Carroll, J. M. and P. M. McLelland, "Fast 'Infinite-key' Privacy Transformation for Resource-sharing Systems," *Proceedings of the Fall Joint Computer Conference,* AFIPS Press, 1970, p. 223-230, 12 refs.
36. Branstad, Dennis K., "Encryption Protection in Computer Data Communications," *Proceedings of the Fourth Data Communications Symposium,* IEEE Computer Society, October 1975, p. 8-1-8-7, 2 refs.
37. National Bureau of Standards, "Proposed Standard Encryption Algorithm for Computer Data Protection," *Federal Register,* 40:52, August 75, 12134-12140.
38. Branstad, Dennis K., "Security Aspects of Computer Networks," *Proceedings of AIAA Computer Network Systems Conference,* American Institute of Aeronautics and Astronautics, New York, N.Y., April 1973, 8p.
39. Kent, Stephen T., "Encryption-Based Protection Protocols for Interactive User-Computer Communication," (Master's Thesis), Massachusetts Institute of Technology, Cambridge, Mass., AD-A026 911, May 1976, 122 p., 42 refs.
40. Turn, Rein and Norman Z. Shapiro, "Privacy and Security in Databank Systems—Measures of Effectiveness, Costs, and Protector-intruder Interactions," *Proceedings of the Fall Joint Computer Conference,* AFIPS Press, Montvale, N.J., 1972, p. 435-444, 26 refs.
41. Turn, Rein, *Privacy Protection in Databanks: Principles and Costs,* The Rand Corporation, Santa Monica, California, AD-A023 406, September 1974, 21 p., 19 refs.
42. Lientz, Bennet P. and Ira R. Weiss, *On the Evaluation of Reliability and Security Measures in a Computer Network,* Office of Naval Research, Arlington, Va., AD-A002 996, December 1974, 28p., 19 refs.
43. Meissner, Paul, *Report of the 1976 Workshop on Estimation of Significant Advances in Computer Technology,* National Bureau of Standards, NBS-IR 76-1189, August 30-31, 1976, 70p., [in press].

A microprocessor selective encryption terminal for privacy protection

by JOHN H. CARSON, JOHN K. SUMMERS and JAMES S. WELCH, JR.
The MITRE Corporation
McLean, Virginia

ABSTRACT

This paper reports on an experiment performed by the METREK Division of The MITRE Corporation from November 1975 through August 1976. The purpose of the experiment was to determine how effectively a low cost microcomputer could provide privacy protection to a user of a time-shared computer system. The microprocessor was used to selectively encrypt information entered at the terminal before transmission to the host computer so that the protected information never appears in plaintext form at the host computer. The advantages and disadvantages of this approach along with the operational details of the experimental system are presented.

INTRODUCTION

The recent advances in microcomputer technology are having a strong impact on the capabilities of conventional time-shared computer systems. By using microcomputers to construct very intelligent terminals, new features and capabilities can be added to existing systems with the intelligent terminal, rather than the host computer system, providing the new functions. Privacy protection is one such function. Privacy protection requirements vary widely for users in the diverse user communities commonly found in large-scale computer systems. The use of a programmable intelligent terminal provides a means for tailoring both the capability and cost of privacy protection to the individual user's needs.

This paper reports on an experiment in which the NBS proposed Data Encryption Standard[1] was used by a microprocessor to selectively encrypt and decrypt portions of the transactions with the host time-shared computer. This encryption terminal can interactively build, maintain and retrieve from databases in which selected fields are encrypted to provide a level of protection sufficient for most privacy requirements.[2]

By performing encryption at the terminal, privacy protection is maintained *independent of the host system.* At no time does the host see the encryption key, the encryption algorithm or the plaintext form of the protected text. Furthermore, this form of protection requires no modification to the host's software. Through terminal encryption, the party held responsible for protection of the data retains control over the protective mechanism with no dependency on the geographically and organizationally remote host computer's operations group.

THE PROBLEM

Privacy protection of computerized databases requires that the data be protected both from accidental disclosure and from unauthorized penetration by a determined intruder.[3,4]

Privacy protection is easily attained when the owner of the data controls the host's operations and processing privileged information is a major part of the installation's job load. Privacy protection is difficult to achieve when the database is loaded on an easily accessed time-sharing computer whose system and operations staff are organizationally independent of the owner of the data.

This situation is further complicated when only portions of the database need protection and the remaining portions of the database should be available to selected user groups. For example, in a medical database the patients' names must be carefully guarded while it is desirable to permit researchers and administrators to study the remainder of the database.

TECHNICAL APPROACH

Figure 1 shows the encryption experiment hardware configuration. A microcomputer, which is inserted between the modem and the terminal, is programmed with the

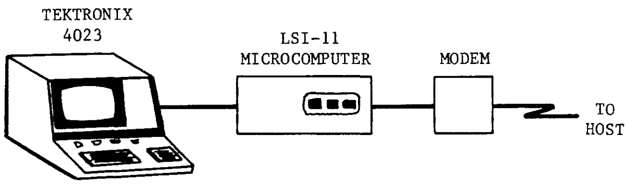


Figure 1—Experimental configuration

encryption/decryption algorithm plus the logic necessary to detect when these processes are to be invoked.

In the METREK experiment, the microcomputer was a Digital Equipment Corporation LSI-11 with 8K words of memory and was attached using serial interfaces to a Tektronix 4023 CRT terminal and an Anderson Jacobson A242 acoustically coupled 300 baud modem. (The LSI-11 software actually required only 5K words but only increments of 4K were available during the experiment.)

The LSI-11 software operates in two modes, plaintext and encryption. In the plaintext mode, the LSI-11 acts as a relay between the terminal and host. Each character from the terminal is relayed, individually, to the host and *vice versa.* All control characters such as carriage return, line feed, insert character, delete character, delete line, break, etc., are relayed to the host for processing.

The terminal user signals the start of a protected phrase by typing an open bracket "[" which puts the LSI-11 into the encryption mode. All subsequent incoming characters are diverted to an internal buffer until the phrase is terminated by a close bracket "]" or a carriage return/line feed. Character deletes are handled by backing up the buffer one character. The phrase terminator causes the collected buffer to be encrypted and transmitted to the host. The transmitted encrypted message is surrounded by brackets "[ ]" to identify it upon return as an encrypted phrase.

Although any encryption algorithm could have been used, the NBS Proposed Data Encryption Standard[1,5] was used as the encryption process for this experiment. The algorithm accepts an 8 byte key (input from the terminal) and an 8 byte block of plaintext data. It produces an 8 byte block of encrypted data. The entire plaintext buffer is fed through the encryption process, 8 bytes per entry. The final block is padded with filler bytes to make it a full eight bytes.

The encryption algorithm is a 16 step block cipher process involving bit permutations and "exclusive or" operations with the key and plaintext block. The characters produced by the algorithm can take on any of the possible 256 8-bit patterns. Unfortunately, most computer systems will not accept all of the 256 possible 8-bit characters.

For example, there are often special system control characters such as carriage return, character delete, line delete, etc., that are intercepted by the operating system and not passed through to the application database software. In addition, there may be control characters for database management systems which cannot appear in data being input. Finally, many computers do not even recognize the full 256 character input alphabet. For example, the METREK IBM 370/145 operating under VM/CMS uses a translation table for converting the ASCII input to EBCDIC. This table is implemented as a many-to-one mapping which has an output alphabet of less than 256 characters.

Since the character identities within the encrypted message serve no meaning when the message is in the host computer, the entire message may be treated as a stream of binary information. The 64 character ASCII subset described in Figure 2 is used to transmit this "binary" message to the host. This is accomplished by dividing the message into 6-bit entities, each of which is mapped into one of the 8-bit characters in the selected ASCII subset. (On return from the host, this process is reversed.)

Upper Case Alphabetics	26 values
Lower Case Alphabetics	26 values
Numerics	10 values
Two special characters (upward ↑ arrow and underline ____)	2 values
Total	64 values

Figure 2

This ASCII subset is free of symbols that have any special meaning to the host. Thus, the encrypted phrase, surrounded by brackets to serve as sentinels, will pass unmolested through the host, to the application database as just another plaintext character string.

This process results in an increased message size which can be computed as follows: If n is the number of bytes in the plaintext phrase, then let N = the integer portion of $(n + 7)/8$, (i.e., the smallest multiple of 8 bytes that can contain the message). Then $8 \cdot N$ is the number of bits in the message and $8N/6$ is the number of ASCII characters required to represent the message. Two sentinel characters must be added to the message bringing the total number of characters transmitted to $4N/3 + 2$.

This approach makes it quite easy to adapt the selective encryption terminal to different host computers. By selection of the proper ASCII subset, the terminal system can be used with most contemporary computer systems.

THE VIEW FROM THE TERMINAL

During the plaintext exchanges, the selective encryption terminal behaves as a conventional "teletype compatible" terminal. Plaintext messages include system commands and responses, and application commands and responses.

Any part of a message can be encrypted by enclosing the segment in square brackets [ ]. For example, the database transaction

```
NAME=[WASHINGTON IRVING],
OCCUPATION=AUTHOR, AGE=194
```

when entered from the selective encryption terminal would actually transmit

```
NAME=[ORDQfq53VKFJNYSOFqisKL],
OCCUPATION=AUTHOR, AGE=194
```

to the host computer where it would create and store a record in the host's database management system. In order to retrieve that record using the NAME field, the request

```
FIND NAME=[WASHINGTON IRVING]
```

would be entered into the terminal resulting in the transaction

FIND NAME=[ORDQfq53VKFJNYSOFqisKL]

being sent to the host computer. The record would be retrieved and displayed at the terminal in its original plaintext form. If the record were to be retrieved either through a conventional terminal or the selection encryption terminal without the proper password, the name field would not be properly decrypted when the record was displayed.

The user sets his encryption key prior to sending the first encrypted phrase. The key governs the encryption procedure and can be used to vary the content of the encrypted phrase. Any phrase that is encrypted using two different keys will produce two different encrypted phrases.

The terminal is notified of a key change by enclosing the new key in brackets within an encrypted message segment. The key is not echoed at the terminal and remains in effect until a new key is entered. For example, the encryption key can be set to "OLD" by typing

FIND NAME=[[OLD]WASHINGTON IRVING]

at the terminal.

FIND NAME=[[ ]WASHINGTON IRVING]

will be displayed at the terminal while

FIND NAME=[ORDQfq53VKFJNYSOFqisKL]

will be transmitted to the host computer.

DEMONSTRATION DATABASE

A database was constructed to demonstrate the operational feasibility of this privacy protection approach. The demonstration database consisted of 100 medical records, selected from a much larger operational database. Each record described an accident victim who was treated at a midwestern general hospital. The records contained 18 fields, including patient identification, patient profile data (age, sex, religion, etc.), accident description, (time, place, injury, how it happened, etc.) and treatment (emergency room facilities, speciality care, survival).

The demonstration database was loaded on the host from magnetic tape, with the patient identification omitted. Patient identification was added later through the selective encryption terminal. Demonstration tests have shown that the host's data management system retained most of its functional capability. The selective encryption terminal is able to update the database, retrieve by patient name (an encrypted field), retrieve a list of patient names sharing some common characteristic (e.g., patients with abdominal injuries) and perform statistical analysis and summary reporting on the collected data. As expected, the host data management system was not able to report a sorted list of patient names nor was it able to perform range type retrieval on patient name (e.g., list all patients whose last name begin with B).

The host data management system was unaware that part of its database was encrypted and no modifications were made to any host programs to perform this demonstration.

VULNERABILITY

In the selective encryption process the same encryption key is used for all values of each encrypted data field and the encryption process is restarted in the same manner for each encrypted value. Furthermore, a constant is used to pad messages to the next higher increment of eight characters. This consistency is necessary for situations where retrieval on the encrypted field is required. The price paid for this consistency is redundancy; everywhere the plaintext field contents are the same, the encrypted values are also identical. Redundancy is an attack point for any determined intruder.

If this redundancy presents a severe problem it can be eliminated by employing random padding characters rather than a constant. The contents of the filler characters affect the entire message and this could be employed to provide an invertible one-to-many mapping. However, employing this technique makes retrieval by protective field impossible.

The decision to employ either random or constant filler must be made by considering the value of a retrieval by protected field value versus the amount of protection required. The intent of this approach is to make it more difficult to obtain the protected information from the computer system than by more traditional methods (e.g., bribery, theft, etc.).

IMPLEMENTATION DIFFICULTIES

The encryption of selected fields of a large database presents two major problems. The first problem appears when the data are loaded into the DBMS or when an existing database is to be used with the encryption terminal. The next problem appears when it comes time to re-encrypt the database because of possible key compromise or because periodic re-encryption under a new key is sound policy.

In either case, a database of any magnitude will rule out the manual data entry approach employed in the medical demonstration database described above. Some sort of automated support must be sought.

Alternatives begin with the transporting of the impacted fields *via* tape or demountable disk, to the microcomputer for encryption and back to the host *via* the same medium. This is expensive since an additional hardware component is required for the microcomputer.

A second alternative is to set up a dialogue between the microcomputer and the host. Each piece of data is sent to the microprocessor, re-encrypted and returned. Although no human participation is required at low speeds (300 baud), the process will be time consuming.

A third alternative is to implement the encryption procedure on the host and perform the encryption during dedicated time when physical security could be assured. This is a risky procedure since the key, the plaintext and the encrypted text are all present in the host system.

CONCLUSION

Selective encryption at a time-sharing terminal is a viable approach to privacy protection of data stored on unsecured host computer systems. By constructing such a terminal and operating it on a demonstration database, MITRE has shown the approach to be technically feasible.

The configuration and component costs of the system used to support the experiment is:

1 LSI-11 microcomputer with 4K words RAM and 1 serial interface	$2,495
4K words additional RAM	625
1 additional serial interface	235
Total	$3,355

The LSI-11 was selected to provide a hardware/software breadboard facility for a range of experiments. The system was not designed to either minimize cost or achieve high performance for the encryption project. Significant cost reductions could be realized in the development of a production system. If the NBS proposed Data Encryption Standard is adopted, it seems probable that a low-cost ($100) LSI version of the algorithm will quickly appear on the market.[6]

Based on this assumption, the following configuration cost implementation is conjectured:

1 microprocessor	$250
4K bytes of memory (mixed ROM and RAM)	140
2 serial interface devices	100
1 power supply	50
1 LSI encryption chip	100
Total	$640

These component cost estimates are intentionally conservative. Additional savings could be realized by building the encryption system into a terminal and taking advantage of the facilities (e.g., power supply, buffer areas) already in the terminal. This would clearly decrease the cost of the implementation and if the terminal were microprocessor based (as many are), the implementation cost could be dropped to around $200.

REFERENCES

1. "Encryption Algorithm for Computer Data Protection," National Bureau of Standards, *Federal Register,* Vol. 40, No. 52—Monday, March 17, 1975.
2. Carson, J. H., W. R. Flury and J. S. Welch, "The Selective Encryption Terminal: A New Approach to Privacy Protection," M76-56, The MITRE Corporation, METREK Division, September 1976.
3. Corasick, M. J., "Protection of Computer-Based Information Systems," M75-235, The MITRE Corporation, May 1976.
4. *The Privacy Act of 1974,* Public Law 93-579.
5. "NBS Standard Puts Encryption in DP System," *Computerworld,* June 7, 1976.
6. Sykes, D. J., "Protecting Data by Encryption," *Datamation,* August 1976, Vol. 22, No. 8, pp. 81–85.

NOTES

1. John H. Carson is now with RLG Associates, Inc., Reston, Virginia.
2. NBS proposed Data Encryption Standard has been adopted (1977).

MIDAS—A compositional modeling system

by J. R. WARNER
Colorado University Computing Center
Boulder, Colorado

ABSTRACT

MIDAS (A **M**odular **I**ntegrated **D**igital **A**nimation **S**ystem) is a portable FORTRAN-based system for defining two- and three-dimensional graphic compositions. Hierarchical modeling functions are coupled with dynamic storage allocation providing extensible storage for easily defined structural and transformational primitives. This modeling system is interfaced to a modular device-independent graphics package that utilizes a segmented virtual display file. Alternative graphics systems may be interfaced.

Typical application imagery includes architectural modeling, visualization of mathematical functions, two- and three-dimensional simulations, and animation. This paper discusses the general system configuration.

INTRODUCTION

A graphic composition may be defined as any spatial model formed from *basic dimensional primitives*. Points, lines, arcs, and alphanumeric characters are common dimensional primitives. An *object* is defined as a named set of contiguous dimensional primitives. A polygon, for example, is defined from connected line segments; a text string is created from alphanumeric characters.

Typically, compositions are developed from "instances"[8] of these prototype objects. Complex compositions are built from groups of object instances, called assemblies, and groups of assemblies. Assemblies may extend to any level, forming a *compositional hierarchy* built, at the lowest level, from simple dimensional primitives.

The compositional hierarchy exists in a finite two- or three-dimensional environment. The units of this environment (e.g., inches, meters, light-years) are determined by the composition designer as a function of the physical or abstract phenomenon being modeled.

A *compositional modeling system* must provide straightforward techniques for defining prototype objects, multiple primitive instances, and hierarchical groups. Further, facility must be made for easily transforming any assembly, instance, or object in the compositional hierarchy. Simple, yet extensible, techniques for displaying all of the composition, or modular components, onto a virtual graphics device must be incorporated into the modeling system.

While flexible display techniques are vital to the modeling function, the enactment of these techniques should be reserved for a modular, device-independent graphics system, interfaced to the composition creation and display procedures.

Graphics modeling systems have been designed and successfully implemented based upon hierarchical data structures.[4,15] Often, however, these approaches go beyond the modeling and display functions, interweaving the low-level computer graphics display routines into the compositional modeling system.

While this approach is effective on stand-alone computing systems supporting a single graphics device, it often obviates graphics independence and precludes system portability among computers.

MIDAS has been designed as a machine-independent, device-independent compositional modeling system. Hierarchical modeling and transformation functions allow arbitrarily complex compositions that are easily transformed in the user's dimensional environment. Simple display conventions provide an understandable mapping from the compositional environment to an idealized, virtual display device.

The compositional design and virtual display modules of MIDAS are independent of the actual graphics display module. Presently, MIDAS is interfaced to a modular graphics system based upon segmented virtual display file techniques.[9,10] Figure 1 defines the general system structure.

This paper discusses the compositional modeling and virtual display capabilities of MIDAS. The organization of the hierarchical data structure ("workspace") is detailed.

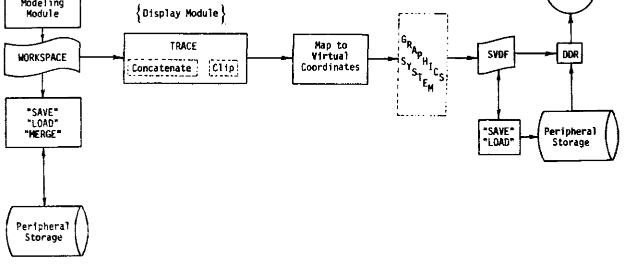


Figure 1—General system structure

The graphics system requirements are briefly discussed. The interface between virtual representation of the composition and final display on a graphics device is presented. Overall system portability is discussed as machine-dependent and machine-independent modules. Finally, a brief review of compositional applications is given.

HIERARCHICAL MODELING IN MIDAS

All hierarchical design packages have similar modeling traits. These include:

- basic dimensional primitives (e.g., points, lines, arcs, character set)
- identification of a contiguous set of elements as an object (e.g., rectangle, polygon, text string)
- definition of object instances from prototype objects
- grouping of objects and object instances into assemblies
- hierarchical assemblies
- transformation capabilities at any level of the hierarchy.

This section details MIDAS' approach to each of these operations.

MIDAS maintains a standard set of dimensional primitives for defining modular objects. Most objects are formed as a contiguous set of two- or three-dimensional points. "Attribute values" are associated with each point to define visible (pen down) or invisible (pen up) line segments. Other attribute values reflect dimensional characteristics of the object (e.g., convex polygon, concave polygon) useful in hidden-line processing. This facility is discussed below.

Special function objects include conics, parametric 2-D curves, text strings, and polygonal shading. Shading is generated as evenly spaced parallel line segments with optional cross-hatching. Other special function objects (e.g., parametric 3-D patches, parallelpipeds, etc.) could easily be incorporated into this basic set.

Actual 2- or 3-D coordinates are either explicitly provided by the user or read from a graphics input device.

For reference purposes within MIDAS, each user-defined object is given a *name*. The user explicitly provides the object name at the time of object definition. Names may be either integer constants or alphanumeric literals at the preference of the user.

New named objects (or *instances*) may be created as spatial transformations (e.g., translation, scaling, rotation) of existing objects.

A group of objects is referenced as an *assembly*. Assemblies also are given names. New assemblies may be created as instances of existing assemblies. Groups of assemblies are also called assemblies. Assemblies may extend to arbitrary levels in the hierarchy. For simplicity in subsequent discussion, objects and assemblies will both be called entities.

Spatial transformations are mathematically represented as 4-by-4 matrices. In MIDAS transformations are named. Instances of entities ("new entities") associate a named transformation with the new entity name. Existing entities are redefined by associating a named transformation to the entity name. Multiple redefinitions are enacted by matrix concatenation. Transformations used to create new entities or redefine existing entities are called *defining* transforms. They are implicitly assigned unique literal names by MIDAS and are, effectively, transparent to the compositional designer.

Another class of transformations, called *attach* transforms, may be explicitly named and linked to entities by the user. Whereas defining transforms enact immediate change in the compositional hierarchy, attach transforms are only applied at entity display time. Defining transforms, implicitly built and concatenated by entity creation or redefinition, permanently update the data structure. Attach transforms allow the application of temporary or iterative transformations without changing the data structure.

Attach transforms permit straightforward programming of animation sequences requiring simple or compound motion (e.g., planetary motion). At the end of the animation, the attach transform is dissociated (*detached*) from the entity and the data structure remains unaltered.

THE DATA STRUCTURE

The MIDAS data structure (or *workspace*) consists of dynamically allocated *nodes*. These nodes maintain information on all created entities and transforms and the interrelationships among these hierarchical components.

Figure 2 provides the data structure schematic of an hierarchical composition. In this example, the composition spans four levels in the hierarchy. At the lowest level, objects and instances of objects reference blocks of dimensional primitives (e.g., points, lines, text). Advanced levels in the hierarchy contain assemblies. An assembly or assembly instance may reference any set of objects or assemblies that are lower in the hierarchy.

The layout of the entity node is shown in Figure 3. Entity nodes are maintained in a doubly-linked queue, called the *entity directory*. As shown in the figure, all information parcels in the entity node are 16-bits long, except for the 32-bit name. This format simplifies implementation on 16-bit machines. The actual information parcels are defined below:

NAME	the 4-character literal or 32-bit integer name of the entity
PLINK	pointer to the predecessor entity in the directory
SLINK	pointer to the successor entity in the directory
TYPE	object/assembly switch
COLOR	7-bit color
INTENS	8-bit intensity
COMPTR	pointer to the entity's component block (discussed below)

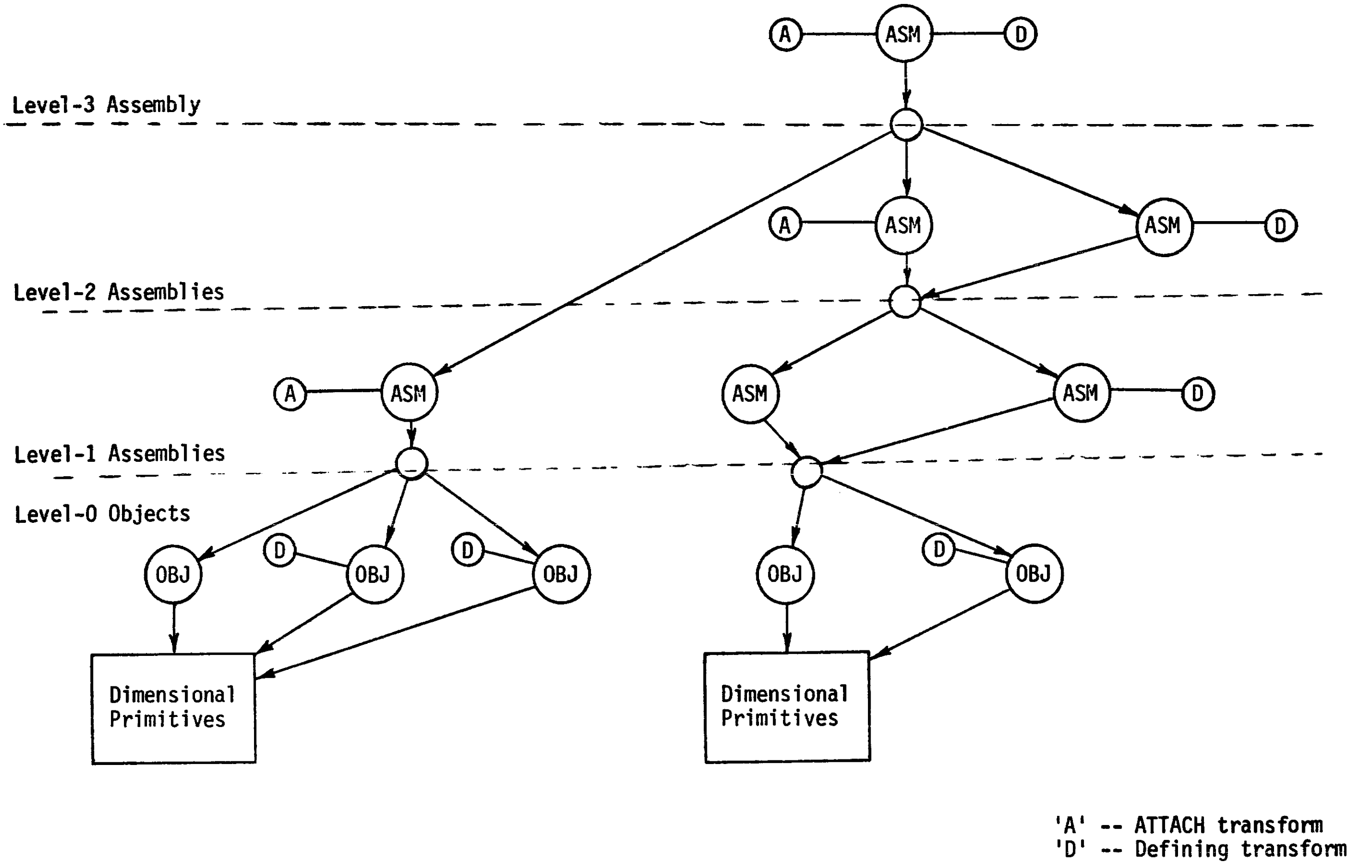


Figure 2—Data structure schematic

DEFPTR pointer to defining transform (if any)

ATTPTR pointer to attach transform (if any)

(optional)SEGPTR address of display file segment for the entity if it is currently in the display file

(optional)SEGCNT number of successive segments in the display file representing this entity
(object: SEGCNT=1; assembly: SEGCNT≥1).

32-bit NAME			
PLINK			SLINK
TYPE	COLOR	INTENS	COMPTR
DEFPTR			ATTPTR
SEGPTR			SEGCNT

Figure 3—Entity node

The SEGPTR and SEGCNT are optional information parcels that directly reference the graphics system virtual display file. They are useful in providing selective update to the display file image without full retrace of the composition data structure. The efficiency savings is vital on high performance, refresh CRT's.

These parameters bind the application data structure to the segmented virtual display file graphics approach. As a result, system modularity may be eroded in favor of display update efficiency. The inclusion of these parcels is left to the discretion of the implementor, to be based upon the resident graphics system (if any) and the probable graphics devices to be supported.

The *component block* of an entity defines the basic dimensional elements of an object or the sub-entities of an assembly. The assembly component block, shown in Figure 4, is always the same. A different format is used for each type of object component block. Typical object component blocks are shown in Figure 5.

Every entity component block contains the component type and a continuation pointer. The five-bit component type (TYPE) allows up to 31 different object forms (e.g., X-Y pairs, X-Y-Z triples, conics, text, etc.). The type permits a parameterized and, subsequently, compact definition of mathematical objects, text strings, and other special type

TYPE=31	SUB-ENTITY COUNT (N)	CONTINUATION POINTER
POINTER to SUB-ENTITY-1		POINTER to SUB-ENTITY-2
POINTER to SUB-ENTITY-3		POINTER to SUB-ENTITY-4
	⋮	
POINTER to SUB-ENTITY-(N-1)		POINTER to SUB-ENTITY-N

Figure 4—Assembly component block

TYPE=5	NUMBER OF SEGMENTS	CONTINUATION POINTER
	RADIUS	
	X_0	
	Y_0	

Figure 5B—Circle object

objects. At recovery time, separate processing routines convert each parameterized object into 3-D line segments that are mapped into the display file.

While parameterization of objects minimizes workspace storage, it can significantly increase the computation time required for converting regularly accessed objects into display file line segments. To overcome this added computational burden, the user may elect to directly maintain the actual 2- or 3-D point pairs in the application workspace. This space/time tradeoff is primarily a function of the modeling application. Other factors are main memory size and the availability of floating point hardware.

The component continuation pointer links the successive component blocks. When created, an entity needs only one component block. However, it is sometimes desirable to explicitly redefine an entity to have more points (for objects) or more sub-entities (for assemblies). The continuation pointer facilitates this expanded redefinition.

Transform nodes (Figure 6) are also linked in a two-way queue, called the *transform directory*. The TRATYP is the *transform type*. Each bit in the type indicates that a particular transformation function (e.g., X-axis rotation) has been concatenated into the transformation matrix. By knowing the actual transformation contents of the matrix at recovery time (e.g., only scaling, only translation, rotation and perspective, etc.), the computation time of the matrix algebra may be abbreviated.[1]

The storage allocation scheme for the compositional modeling module obtains blocks of main memory from a sequential free storage pool. Some machines will allow dynamic expansion of a user's program space at run time. In such systems the user need not declare the maximum size of the free storage pool. On limited memory machines, where complex compositions could not be maintained wholly in core, the workspace could easily be processed in blocks, paged in and out of peripheral storage.

Note that this workspace configuration does not facilitate run-time deletion of entities or transforms. Deletion is felt to be a graphics function (e.g., deleting a segment from the display file), rather than a modeling function. The only time components of the modeling structure need to be deleted is when they are never referenced, are impractical to explicitly redefine, and free storage is exhausted. By experience in the current implementation, this has rarely occurred.

DISPLAYING ENTITIES

Figure 7 provides a general flowchart of the composition display process. The recovery algorithm traces each branch in the hierarchy of the entity to be displayed. A *combined transformation matrix* (CTM) is built for each branch, reflecting the concatenation of all transformation matrices along that branch. A push-down stack maintains hierarchical ordering of both entities and CTM's allowing pseudo-recursive tracing of the compositional branches.

When an object node is detected at the end of each branch, the CTM is applied to the two- or three-dimen-

TYPE=0	POINT COUNT (N)	CONTINUATION POINTER
	X_1	
	Y_1	attribute value$_1$
	X_2	
	Y_2	attribute value$_2$
	⋮	
	X_N	
	Y_N	attribute value$_N$

Figure 5A—2-Dimensional point pair object

TYPE=8	NUMBER OF WORDS	CONTINUATION POINTER
	X-CENTER OF STRING (X)	
	Y-CENTER OF STRING (Y)	
	WIDTH OF STRING (W)	
	HEIGHT OF STRING (H)	
	The character string is stored 4 characters per word starting in word 5. A sentinel character and special code characters signal italics, case change, end of string, etc. The string is mapped into the rectangular area defined by (X,Y,W,H).	

Figure 5C—Character string object

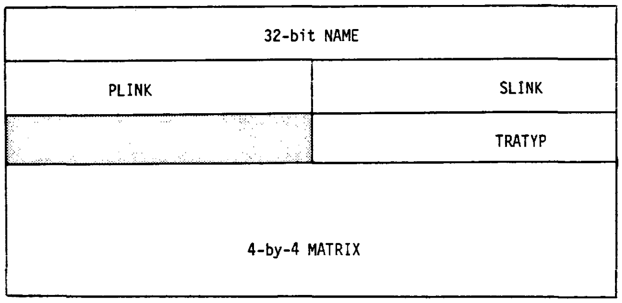


Figure 6—Transform node

sional coordinates of the object. The resultant user-dimensioned coordinates are passed through an interface module for mapping into virtual display coordinates and creation of a virtual display file segment. Special purpose recovery software converts parameterized primitives and text into 3-D line segments before applying the CTM.

If hidden line processing is to be performed, the mapping interface is pre-empted. Instead, the recovered coordinates are saved in dynamically allocated recovery buffers. The hidden line processor (discussed, below) synthesizes the wire-frame composition in these buffers, routing the hidden line output to the interface module.

The interface module performs 2- or 3-D clipping as required and maps the recovered compositional entities into the virtual display space. In the current implementation, each object of the composition defines a separate display file segment.

The user may explicitly define both physical (i.e., user-dimensioned) windows and viewports, and virtual display windows for mapping any compositional region onto any rectangular area of the virtual display.

HIDDEN LINE ALGORITHM

The hidden line algorithm accepts polygonal objects for processing. Sutherland's re-entrant polygon clipper[6] preprocesses these objects into convex "shells and holes." Non-polygonal objects are detected and treated as "hideable, but may not hide." The shells, holes, and non-polygonal figures are then processed against each other using a subset of the polygon-clipping algorithm to do line clipping. Facility is made for implicit line calculation of intersecting polygons.

The visible lines of the figure are finally returned to the interface module for mapping into the display file.

The algorithm requires space for the convex 3-D representation and normal vector of each polygon. Processing time increases as (n)*(n−1) where n is the number of convex representations.

THE GRAPHICS SYSTEM

The graphics system is organized around a segmented virtual display file.[10,13] This display file is geared toward an idealized virtual terminal[20] and, as such, is graphics device independent. A separate set of device-dependent routines (DDR's)[1] is loaded at execution time for each graphics device that is supported.

As in the MIDAS workspace, the segmented virtual display file (SVDF) is implemented for 16-bit words dynamically allocated from a separate free storage pool. Segments are allocated sequentially as the display file is built.

Figure 8 defines the actual segment block. Segments do not have names. Instead, SVDF routines use the segment address as the identifier returned to the calling routine. PLINK, SLINK, COLOR, AND INTENS are analogous to the same functions defined for entity nodes. COUNT is the number of (X, Y) coordinate pairs in the segment. PCKSNS is a switch indicating if the segment is pick-sensitive, a mandatory feature for devices allowing graphic input. POSTS and PAINTS are switches indicating the current status of the segment on the display device.[13] A segment is *posted* if it is to be displayed during the next display file trace (refresh). An *unposted* segment is not to be displayed during the next trace. A *painted* segment is currently displayed on the display device. An *unpainted* segment is not currently displayed.

XMIN, XMAX, YMIN, YMAX provide boundary information on the segment in virtual display coordinates, thereby speeding up the picking operation.

The virtual display coordinates are 14-bit integers, allowing a virtual display region of 0≤X≤16383; 0≤Y≤16383. As shown, a two-bit opcode precedes both the X and Y coordinates. This design preserves the 16-bit word criterion. The opcodes, interpreted by the DDR's, provide point plot, move/draw, absolute/relative information for each coordinate pair.

This particular implementation allows very simple DDR's. The only required primitives that must be written for each device are:

BEGIMG	—initialize a new image
ENDING	—terminate image (screen erase, frame eject)
SETCLR	—set color
SETINT	—set intensity
MOV/DRW-ABS	—move/draw absolute
MOV/DRW-REL	—move/draw relative
PNTABS	—point absolute
PNTREL	—point relative

The SVDF assumes that all coordinates are in the domain of the virtual screen. Transformation, clipping, and windowing are considered application program/system functions (e.g., MIDAS). All character output is generated by software.

This SVDF structure simplifies reference to complete images (frames), buffering in and out of peripheral storage (e.g., SAVEDF, LOADDF), and transfer of graphic data files among computing installations. Increased sophistication could be incorporated into the DDR's to allow virtual image windowing and clipping before mapping from virtual to screen coordinates. Similarly, software text generation

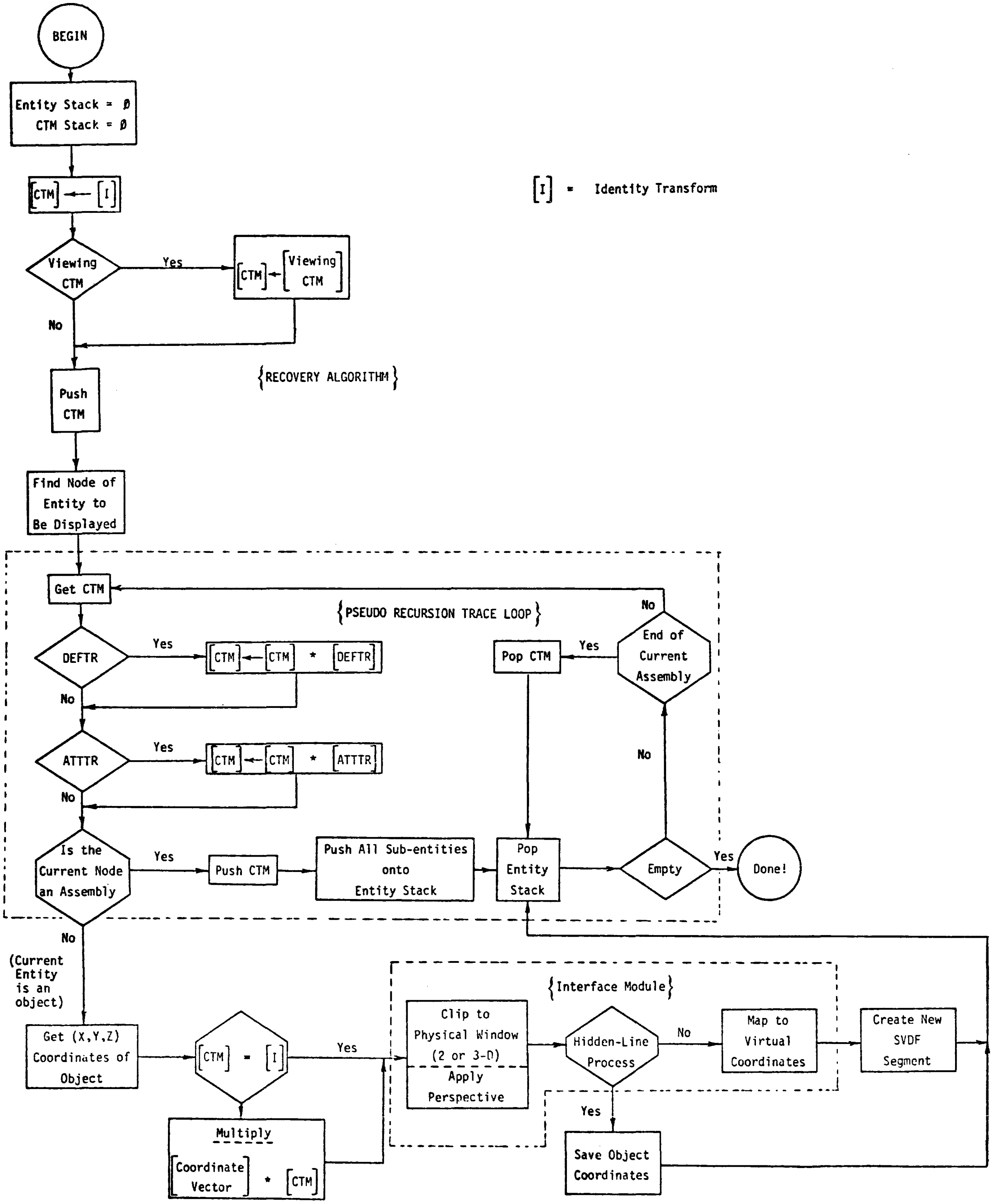


Figure 7—Composition display process

PLINK		SLINK			
COLOR	INTENS	COUNT	PCKSNS	POSTS	PAINTS
XMIN		XMAX			
YMIN		YMAX			
op code	X_1	op code	Y_1		
op code	X_2	op code	Y_2		
⋮					
op code	X_N	op code	Y_N		

Figure 8—SVDF segment block

and polygonal shading could conceivably be done at the DDR level, freeing up SVDF storage requirements.

These and other "enhancements" are feasible and, indeed, practical for computing installations with specific graphics applications. Unfortunately, as the SVDF segments become more parameterized, the DDR's must assume more of the computational burden of generating the image. The increased complexity at the DDR level deters both graphics system modularity and the transfer of SVDF files to different machines.

Though not as elegant or potentially efficient as more customized approaches,[3] the described SVDF permits straightforward DDR display on all vector-based graphics devices and uncomplicated transfer of imagery among computing installations.

APPLICATIONS

The user accesses MIDAS as a library of FORTRAN-callable subroutines. Typically, application programs are written in FORTRAN, with CALLS to the MIDAS routines. The MIDAS routines build, update, and display the composition.

Animation

In 1973, MIDAS was conceived as an animation system. While it has evolved into a compositional modeling system, many of the original animation features remain.

Animation with MIDAS is not real-time. The software transformation algorithms, combined with a frame by frame hierarchical trace, preclude real-time image update on high-performance refresh displays. At this time, real-time vector-based animation requires transformation hardware. As a result, existing real-time systems[2] are entirely device-dependent.

Animation using MIDAS is produced in stages that parallel conventional animation techniques. Working from an action scenario, or *storyboard*, user-written modeling programs define the compositional imagery to be used in the animation sequence. This imagery is maintained on peripheral storage as one or more MIDAS workspaces. Compositional editing programs may be invoked interactively to load, preview, edit, and update the animation workspaces.

When the static compositions are correct, the actual animation program(s) are invoked. The MIDAS display routines simplify the specification of animation parameters such as varying focal point, virtual camera position, and motion along a path. The ATTACH transform facilitates iterative motion sequences. Coupled with hierarchical constructs, the ATTACH transform simplifies compound motion sequences. Other special function routines allow overlaid imagery, pan, zoom, tilt effects, and key-frame animation.

When generating the actual animation sequence, the display file of each frame is saved in compacted form on disk. An interactive preview program can then be used to view and analyze selected frames of the animation sequence.

When both the imagery and motion are deemed correct, the compacted animation sequence is routed to a hardcopy display device. Normally, this would be a microfilm recorder or storage-tube terminal synchronized with a 16-mm camera.

A complete discussion of microfilm-based animation techniques using MIDAS is planned.

DESIGN

MIDAS is currently used in a variety of design and simulation disciplines including architecture, chemistry, and psychology.

Most design applications involve an interactive monitor program that invokes MIDAS routines for on-line composition definition and editing. As the modular design is being built, intermediate views may be either saved on disk or routed to a hard-copy display medium. Utility workspaces maintain regularly-used objects and assemblies (e.g., cubes, circles, cylinders, icosahedra).

Hidden-line processing on polygonal compositions enhances the displayed image. Figures 9 and 10 show hidden-

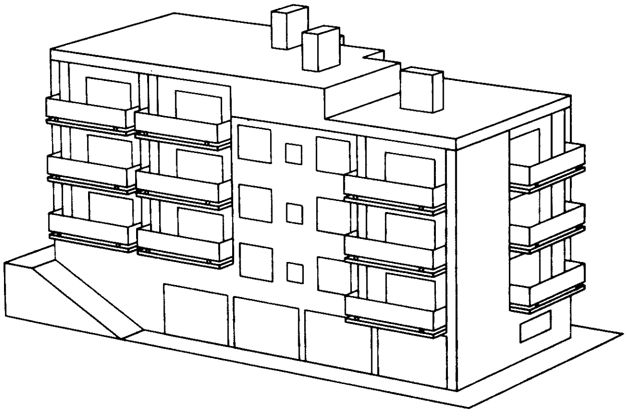


Figure 9—Perspective view of structure

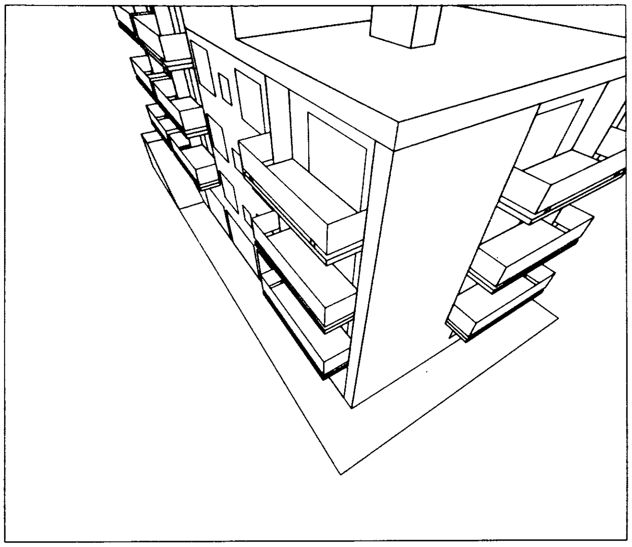


Figure 10—Perspective view of structure

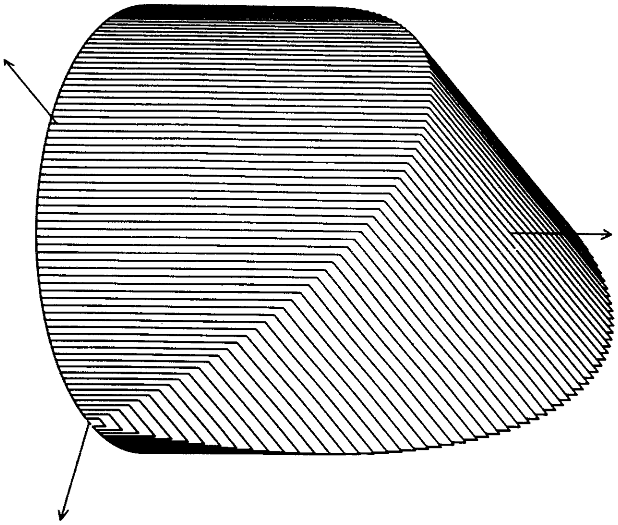


Figure 11—Solid built from parallel plane cross sections

line processed design imagery, varying both the focal point and virtual camera position.

The compositional modeling and display modules of MIDAS do not easily support the graphic input capabilities inherent in truly interactive computer-aided design systems. Symbolic entity reference and compositional transformation within interactive MIDAS programs are normally enacted by keyboard input. Menu selection and dynamic 2-D transformations via light-pen, cursor, mouse, etc., while feasible, are unwieldy within the hierarchical constructs. These operations are considered graphics system functions requiring explicit user reference to the segmented display file. MIDAS does allow direct user interaction with the segmented display file via subroutine calls to the graphics system routines. However, this implies user understanding of the SVDF structure and the interface between the SVDF and the compositional workspace.

Mathematical imagery

This area is actually a subset of design, relying heavily on the viewing and hidden-line algorithms. Three-dimensional solids are built as parallel-plane cross-sections (Figure 11). Solids of revolution are built from concentric disks (Figure 12). Certain 3-D surfaces may be simulated (Figure 13). Polygonal shading hilights 2-D areas (Figure 14). Overlaid color transparencies can be used to identify critical components of a mathematical model. Animation is a straightforward progression from these static displays.[16]

Teaching computer graphics

MIDAS can provide an easy introduction to the rudimentary mathematical concepts of computer graphics. These include concepts in coordinate geometry, transformation matrix algebra, matrix concatenation, and clipping algorithms.

PORTABILITY CONSIDERATIONS

As described above, MIDAS is primarily coded in FORTRAN-IV. FORTRAN is presently the only viable language for non-business applications with 'universal' implementation. Only this universality justifies FORTRAN for hierarchical modeling applications. It has neither the structured procedures nor recursive programming facilities of either PL/1 or SAIL. The matrix notation and algorithm specification is clumsy compared to APL. The representation and manipulation of character data is syntactically awkward. Perhaps most importantly, FORTRAN is difficult for 'non-scientists' to learn. Collectively, these drawbacks may doom the FORTRAN language to gradual extinction.

Acknowledging this structural rigidity and staged obsolescence, FORTRAN remains the most portable medium for scientific application systems. Utilizing modular programming techniques, with prudent use of machine language primitives, can significantly improve the code, enhancing both readability and portability.

Within MIDAS, machine-dependent code is selectively used for those utility functions requiring optimum efficiency (e.g., low-level data structure routines). Machine word characteristics (e.g., bits per word, characters per word, words per floating point value, etc.) are stored in global COMMON blocks and are used to calculate addressing and shifting values at run time. Certain regularly accessed routines (e.g., matrix multiply, clipping, dot product) are coded in FORTRAN to improve portability at a sacrifice of efficiency. Implementors are encouraged to rewrite these routines in their local machine language.

Character I/O is a function of word size and characters

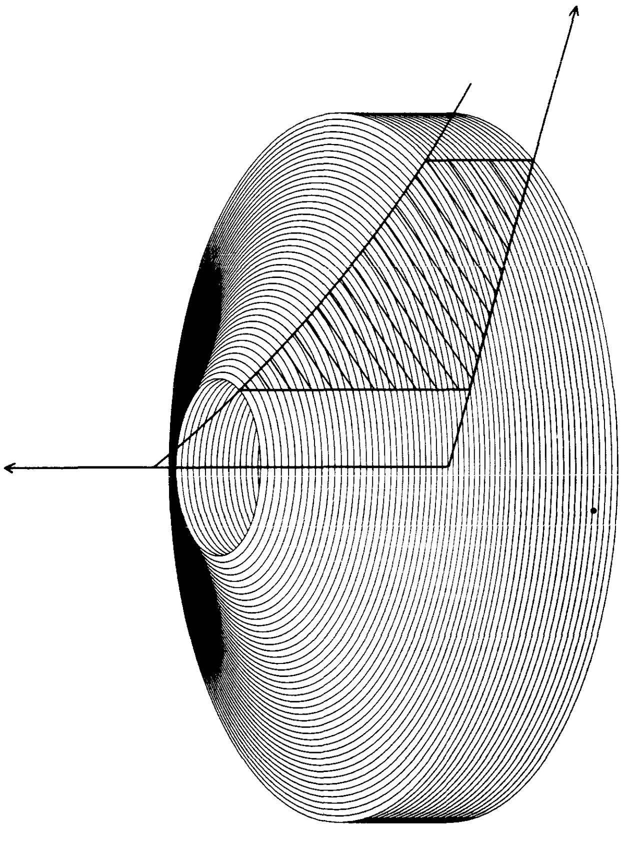


Figure 12—Solid of revolution

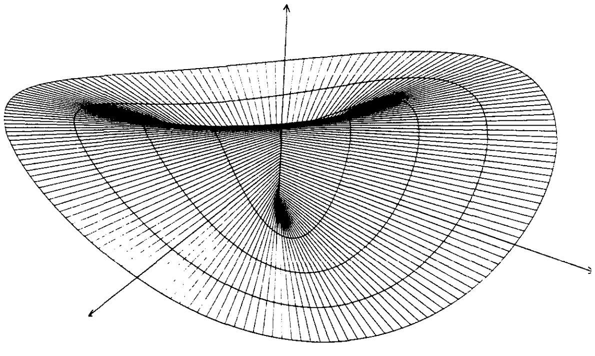


Figure 13—Shading to hilights a 2-dimensional area

per word. Print formats are maintained on disk and loaded at run time. As a result, WRITE statements use variable formats.

Application program debugging and error messages are optionally available from all modeling and display routines. By default, the extent of commentary is implicitly controlled by the user's mode of access; batch or interactive. The user may explicitly control the verbosity of comments, from full program tracing to only writing fatal run-time diagnostics.

All FORTRAN code has been reviewed using the PFORT Verifier developed at Bell Labs.[12] PFORT performs stringent portability verifications against a subset of ANSI-FORTRAN. Wherever practical, MIDAS conforms to the more stringent PFORT standards.

The Graphics System Module (GSM) supporting the SVDF also conforms to the PFORT standard wherever practical. The MIDAS interface to the GSM is modular and well-defined. This allows implementors with an existing graphics system easy interface to the MIDAS composition and display modules without major recoding.

The 16-bit display file format presented here is encouraged, but is not inflexible. Facility is made for easy redefinition of segment parcels within the SVDF segment block to optimize word usage (e.g., packing two X, Y coordinate pairs into a single CDC 60-bit word). Similarly, the workspace node formats may be redesigned for improved efficiency in storage and/or execution.

The DDR's described here are written in machine language. Each graphics device at a given computing installation will have nuances inherent to both the device and the installation. If the display file format is kept simple, the implementor should be able to code device-dependent routines for a single device in half a day. Any parameterization of imagery within the display, other than points and lines (e.g., hardware text, shading, circles and arcs), will burden each DDR with specialized image generation software. The tradeoff is increased SVDF memory versus DDR complexity.

MIDAS should not be viewed as a 'general purpose graphics system.' Several existing packages[5,19] use this catchall phrase as an alluring description of their capabilities. The implication is that all graphics applications, from 2-D data display through real-time hidden-line computer-aided design, are easily realized on all graphics display devices.

Most such systems are founded on a requisite set of graphics primitives that maintain some form of display file

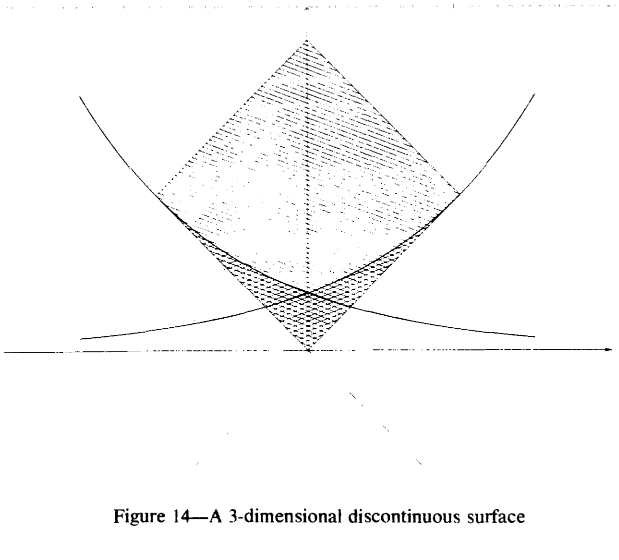


Figure 14—A 3-dimensional discontinuous surface

image. Often, however, special-purpose enhancements (e.g., single-call graph generation subroutines, complete with tick marks and annotation) front-end the display processor. Other systems interweave an application-oriented picture structure into the graphics routines. Neither these hybrid approaches nor the MIDAS modular approach implies generality.

The general purpose graphics system is a marketing idiom offered as the conditional panacea of computer visualization. The general purpose graphics *application* system is non-existent.

Only the general purpose graphics *display* system purports credibility. The 'display' qualifier limits the scope of such a system to display manipulation functions and graphic I/O. Virtual display file constructs and localized transformation utilities are also within the definition of a general display system. Clipping and windowing are the highest level functions of such a system, providing a clean interface to the user's application program or the structured application system.

While a general purpose graphic display system should be algorithmically portable among machines, it is often impractical to require strict code portability.

To capsulize, MIDAS is a structured compositional modeling system that is readily interfaced to a graphics display system. Neither the modeling module nor the interfaced SVDF graphics display system are considered general purpose. Only the segmented virtual display file system could be expanded into a general graphics display system.

SUMMARY

This paper has discussed both the modular configuration and certain internal characteristics of the MIDAS modeling system. A separate graphics system, interfaced to MIDAS, and based on segmented virtual display file techniques, was presented. Compositional applications were suggested in animation, design, mathematical simulation, and computer graphics instruction. Graphics system generality was qualified in a discussion of modular systems configurations, device-independence and portability.

ACKNOWLEDGMENTS

The structural simulations (Figures 9 and 10) were generated by Mr. Imre Komaromi of Budapest, Hungary. The mathematical images (Figures 11-14) were generated by Mr. Ken Joy of the University of Colorado Mathematics Department. Mr. Joy has also done all the development on the hidden line processor implemented within MIDAS.

REFERENCES

1. Blinn, J. F. and A. C. Goodrich, "The Internal Design of the IG Routines—An Interactive Graphics System for a Large Time-Sharing Environment," in *SIGGRAPH '76 Proceedings*, pp. 229-234.
2. Csuri, C., "Computer Animation," *SIGGRAPH '75 Proceedings*, pp. 92-101.
3. De Fanti, T., "The Graphics Symbiosis System," in *Real Time Film Animation*, Computer Graphics Research Group, Ohio State Univ., Columbus, Ohio, January 1973.
4. Herzog, B., "DRAWL in MTS," *Compgraph*, Ann Arbor, MI, Oct. 1971.
5. Hirschsohn, I., P. Preuss, and M. S. G. Repko, "Design and Implementation of the DISSPLA Graphics Language," *Eurocomp '75 Proceedings*, London, 1975.
6. Hodgman, G. W., and I. E. Sutherland, "Reentrant Polygon Clipping," *CACM*, Vol. 17, No. 1, January 1974, pp. 32-42.
7. Kitching, A., "The ANTICS Computer Animation System," *Proceedings of 1975 Eurocomp Conference on Interactive Systems*, Online Conferences Ltd., Uxbridge, England, pp. 465-480.
8. Newman, W. M., "Instance Rectangles and Picture Structure," *Proceedings of the Conference on Computer Graphics, Pattern Recognition, and Data Structures*, IEEE Catalog No. 75CH0981, May 1975.
9. Newman, W. M. and R. F. Sproull, "An Approach to Graphics System Design," in *IEEE Proceedings*, Vol. 62, No. 4, April 1974, pp. 471-483.
10. Newman, W. M. and R. F. Sproull, *Principles of Interactive Computer Graphics*, New York, McGraw-Hill, 1973.
11. Rogers, D. F. and J. A. Adams, *Mathematical Elements for Computer Graphics*, New York, McGraw-Hill, 1976.
12. Ryder, G. G., "The PFORT Verifier," *Software Practice and Experience*, Vol. 4, 1974, pp. 359-377.
13. Sproull, R. F., *OMNIGRAPH: Simple Terminal-Independent Graphics Software*, XEROX Palo Alto Research Center, December 1973.
14. Sutherland, I. E., R. F. Sproull, and R. A. Schumacker, "A Characterization of Ten Hidden-Surface Algorithms," *Computing Surveys*, Vol. 6, No. 1, March 1974.
15. Van Roekel, J., *The Michigan Graphics Interpreter*, Dept. of Aerospace Engineering, University of Michigan, Ann Arbor, MI, 1971.
16. Wahl, M. and J. R. Warner, *Dynamic Functions*, (film), University of Colorado, July 1973.
17. Warner, J. R., "Design Applications of the MIDAS Graphics System," *Computers and Graphics*, Vol. 2, No. 1, 1976, pp. 15-22.
18. Warner, J. R., *MIDAS Programming Manual*, Colorado University Computing Center Library, Boulder, CO, 1976.
19. Woodsford, P. A., "The Design and Implementation of GINO3D Graphics Software Package," *Software Practice and Experience*, Vol. 1, Oct. 1971, p. 335.
20. Wright, T., "A Schizophrenic System Plot Package," in *SIGGRAPH '75 Proceedings*, pp. 252-255.

A system for automatic acquisition of three-dimensional data

by HENRY FUCHS, JOE DURAN and BRIAN JOHNSON
The University of Texas at Dallas
Richardson, Texas

ABSTRACT

This paper presents the design of a three-dimensional data acquisition system based on multiple, single-dimensional optical sensors. The system can operate in any of three modes:

(1) the tracking of multiple, independent, point light sources
(2) the automatic digitization of opaque surfaces
(3) the real-time tracking of an unmarked moving object (e.g., tip of user's hand).

The design offers such advantages as a lensless sensing system, a minimum reliance on analog measurements, an ease of upgrading to higher precision measurements, an ease of portability, an adjustable field of view, and the ability to operate under normal ambient light conditions. A network of microprocessors is incorporated to minimize processing delays and thus increase data acquisition rates. In its initial application the system will digitize the craniofacial surfaces of candidates for reconstructive surgery.

INTRODUCTION*

Recent developments in charged coupled device (CCD) technology have made generally available a variety of IC chips which contain a linear array of light-sensitive elements (Figure 1). Such an array can be treated in the system as a digital shift register in which successive clock pulses cause serial output of the entire array of values. (The sole complication, that the serial output is analog, is easily overcome with a single analog-to-digital converter.) Each of the individual values in the array is a function of the number of photons which have been absorbed by the associated cell in the array since the previous scanning of the shift register's contents. If the CCD chip and the analog-to-digital converter are considered as a single unit, then a sequence of clock pulses will obtain an array of *digital* values which define, as a function of distance along the array axis, the amount of light striking the CCD chip.

To build a one-dimensional sensor for 3-D input, we place in front of the CCD array a knife-edge in a plane parallel to the CCD array face with the line of the knife edge boundary perpendicular to the line of array elements (see Figure 2). If the environment contains a point light source which is brighter than the ambient light level, then the knife edge will cause a shadow to be cast somewhere along the line of light-sensitive elements.

MATHEMATICAL FORMULATIONS

With a number of such detector units placed around the room, the location of a light source can be determined by the taking of a single measurement, h_i, at each linear sensor, i, which can "see" the source. Each h_i measurement defines a plane in which the light must be located. When three such planes are defined, each containing the same light source, the location is uniquely determined (Figure 3). The use of homogeneous matrices considerably simplifies the mathematical formulations (c.f. Reference

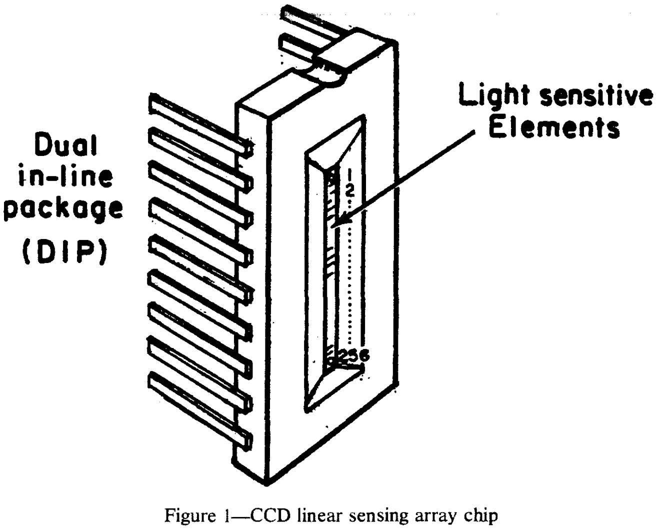


Figure 1—CCD linear sensing array chip

* Appendix A includes a short discussion of previous digitization devices and their effect on the design presented in this paper.

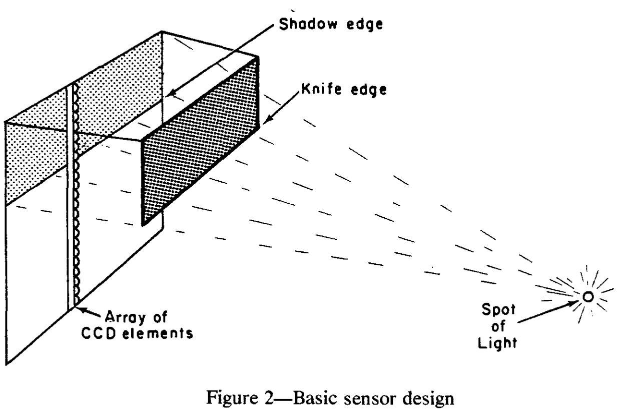


Figure 2—Basic sensor design

11). We show below that these simplifications allow the problem of computing the coordinates of a point light source to be stated as the problem of solving a system of planar equations of the form

$$a(h)X+b(h)Y+c(h)Z=d(h), \qquad (1)$$

where h is the position of the shadow edge (Figure 4). X, Y, Z are our system coordinates, and the coefficients a, b, c, d are linear expressions in h.

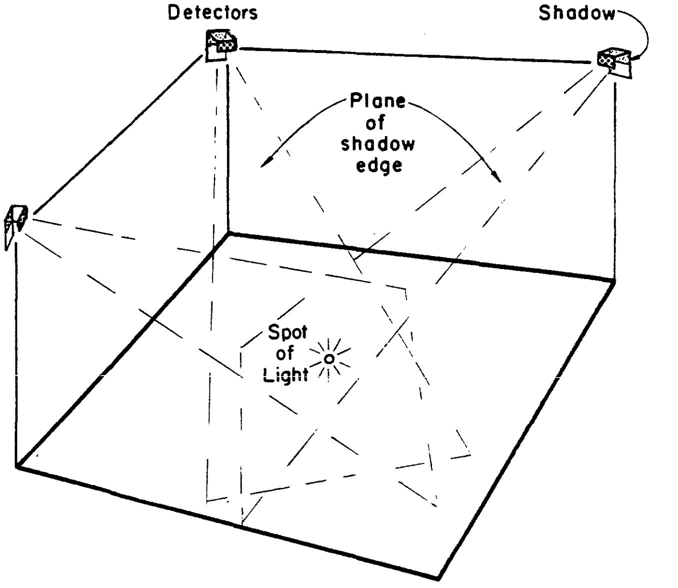


Figure 3—(3-D position from) 3 linear sensors

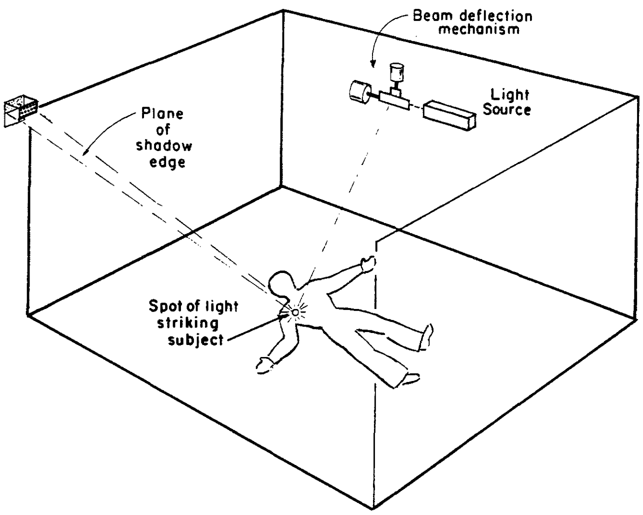


Figure 4—Sensor/room coordinate relationship

It is well-known (c.f. Reference 6) that an appropriate 4×4 matrix can be used to transform a 3-D point between 2 coordinate systems. For a single sensor, i, the transformation from room to sensor coordinates (see Figure 4) can be expressed by the 4×4 matrix, C^i, such that

$$(X \ Y \ Z \ 1)[C^i]=(x_i' \ y_i' \ z_i' \ w_i), \qquad (2)$$

Where $x_i=\frac{x_i'}{w_i}$, $y_i=\frac{y_i'}{w_i}$, and $z_i=\frac{z_i'}{w_i}$ are the actual sensor coordinate values. For the ith sensor, h_i, r_i, and the coordinate components, x_i and y_i, of the light source in the sensor coordinate system are related by the expression,

$$\frac{h_i}{-r_i}=\frac{y_i}{x_i}=\frac{C_{12}{}^iX+C_{22}{}^iY+C_{32}{}^iZ+C_{42}{}^i}{C_{11}{}^iX+C_{21}{}^iY+C_{31}{}^iZ+C_{41}{}^i}$$

Simplifying by collecting factors of X, Y, and Z yields

$$(h_iC_{11}{}^i+r_iC_{12}{}^i)X+(h_iC_{21}{}^i+r_iC_{22}{}^i)Y+(h_iC_{31}{}^i$$
$$+r_iC_{32}{}^i)Z+h_iC_{41}{}^i+r_iC_{42}{}^i=0$$

One of the transformation matrix elements can be eliminated by dividing through by, say $r_iC_{42}{}^i$. This yields

$$(h_ip_{1i}+p_{2i})X+(h_ip_{2i}+p_{4i})Y+(h_ip_{5i}+p_{6i})Z+h_ip_{7i}+1=0, \qquad (3)$$

where $P_{1i}=\frac{C_{11}{}^i}{r_iC_{42}{}^i}$, $P_{2i}=\frac{C_{12}{}^i}{C_{42}{}^i}$, etc.

Eq. (3) is of the form of eq. (1) and is the equation of a plane in room coordinates. The light source lies in this plane. From three or more such planar equations, the unique location (X_l, Y_l, Z_l) of the light source in room coordinates can be calculated, either by standard matrix solution techniques or by least squares methods. If the p_{ji} are known for each sensor i, the h_i values measured for each of three or more sensors will, by eq. (3), uniquely determine the light source location.

Our sensors can be calibrated quite simply by determining the p_{ji}. Given the known position of seven light sources in room coordinates and the h_i value generated by each source at each sensor, we obtain a linear system of seven equations in the seven unknowns $p_{1i}, p_{2i}, \ldots, p_{7i}$ for each sensor. The seven points should be chosen so as not to give an ill-conditioned or singular matrix. The probability of such problems can be reduced by the use of more than seven calibration points, allowing a least squares solution.

Once the p_{ji} are calculated for each sensor, the location of an unknown point can be found by solving the linear system

$$\begin{pmatrix} a_i & b_i & c_i \\ a_j & b_j & c_j \\ a_k & b_k & c_k \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} d_i \\ d_j \\ d_k \end{pmatrix} \tag{4}$$

(where $a_i=h_i p_{1i}+p_{2i}$, $b_i=h_i p_{3i}+p_{4i}$, etc.) for any three sensors i, j, k which see the light source. If more than three sensors see the source, error smoothing via a least squares solution of an overdetermined system is possible. If the values of $a(h_i)$, $b(h_i)$, $c(h_i)$, and $d(h_i)$ for each sensor are stored in advance for each possible h_i value (quite feasible, since h can take on 256 discrete values for our current sensor), then the coordinates of a point may be calculated with about 20 or so multiplications and divisions. The exact number depends upon such factors as the number of precalculations stored and the number of excess sensor equations which may be desired for error smoothing. This processing load is light enough so that the system can operate at sensor speeds with only a minicomputer serving as the central host.

With the use of a computer-deflected light beam, only one sensor needs to detect the light reflected from the surface being digitized, since the position of the light beam source in the environment will be previously determined and its orientation (deflection in two axes) will be under computer control. The plane and line measurements thus determine the 3-D location of the point on the target surface which is illuminated by the narrow beam of light (Figure 5). This might best be implemented by defining the line of the light beam in terms of the equation (in the form of eq. (3)) of two planes which intersect along the line. The sensor measurement thus defines the third necessary plane and the computational method described above may be used. This has the advantage of readily allowing a least squares error smoothing treatment if the use of more than one linear sensor is desired.

With some sacrifice in speed, it is possible to use the reflected light beam mode for interactive input applications. In this case the light beam deflection would be controlled by a real-time program which would acquire and track the object of interest (e.g., the tip of the user's hand) in a manner similar to a stage spotlight following a star performer.

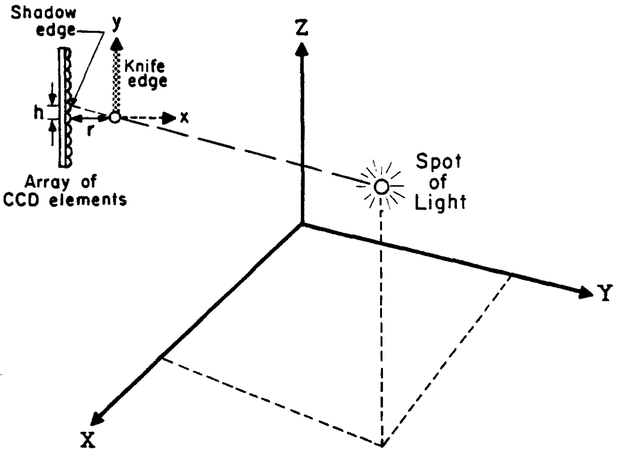


Figure 5—Automatic surface digitization mode

SYSTEM ARCHITECTURE

The current system layout has a microprocessor associated with each CCD array. The microprocessor is used to control the integration time of the CCD array (the amount of time it is allowed to sense), to collect the data from the array, and to compute the element position value (h_i) of the shadow boundary. Since current CCD arrays suffer from background noise level variations from cell to cell, individual cell calibration improves sensor performance. The microprocessor records these variations during an initial calibration phase and uses them to improve the shadow boundary determination.

The microprocessors are all connected to a host CPU, a small minicomputer with a minimal floating point capability (currently an HP 2100). The host computes the source point coordinates by solving the system of linear equations (4).

In the single point (or "wand") acquisition mode, the host stores the computed coordinates and waits for the next input point. It also carries out various control functions such as starting and stopping the process, changing the acquisition rate, etc. With multiple light source input, the host fires the sources at known times and keeps track of the association between coordinate computations and the corresponding sources. This mode allows tracking of not only the position, but also the rotational orientation of a moving object by the tracking of the position of three noncollinear light sources attached to the object.

In the light beam scanning mode, the host also must calculate coordinates for the scanning beam and deflect the beam to the desired orientation. The host polls the sensors, computes the coordinates of the reflected light spot, deflects the beam to the next desired position, then polls the sensors again, continuing in this manner until data acquisition is completed.

SENSITIVITY AND ACCURACY

Our prototype sensor uses a 256-element Fairchild model 110 CCD array. The measured effective sensitivity of the array is about 10^{-5} watts/cm^2. This indicates that a uniformly radiating point source of .1 watt can be sensed up to about 40 cm. (In practice we have been able to do slightly better, even without resorting to individual cell calibration.) We have sensed a 2 watt source at a distance of about 2 meters, in a room which was darkened but with still enough light for reasonable human vision.

The sensing distance for a projected spot depends both on the amount of energy projected onto the target surface and the scattering coefficient of this surface. Preliminary experiments indicate that the current sensor will enable digitization of human heads but not automatic scanning of

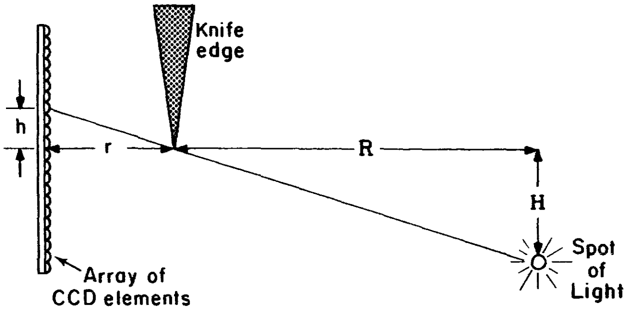


Figure 6—Sensor resolution distances

entire bodies, without resorting to high-powered light sources. We expect continual improvement in CCD sensitivity and can use other sensor technologies, such as arrays of individual photo-transistors or image intensifier devices, if desired.

The sensor's resolution is of course limited by the number of cells in the CCD array. Thus, the 256-element sensors yield a resolution of about .6 cm in a 1 cubic meter working volume. The smaller the working volume, the better the absolute resolution limit. The resolution depends upon the number of CCD elements and the relative distances of the CCD array, knife edge and source (see Figure 6). The absolute resolution in room coordinates is related to CCD element width, Δh, by the relation $\Delta H = \frac{R}{r} \Delta h$.

Accuracy increases can be realized by using sensors with more elements (sensors with over 1000 elements are currently available) or using more sensors to segment a desired working volume into several contiguous, smaller volumes.

CURRENT STATE OF IMPLEMENTATION

A prototype sensor system has recently been completed and interfaced to an HP 2100 computer. Figure 7 illustrates the output from the linear sensor array. A sharper distinction between the outputs of the shadowed and non-shadowed elements can be achieved by substituting a smaller light source.

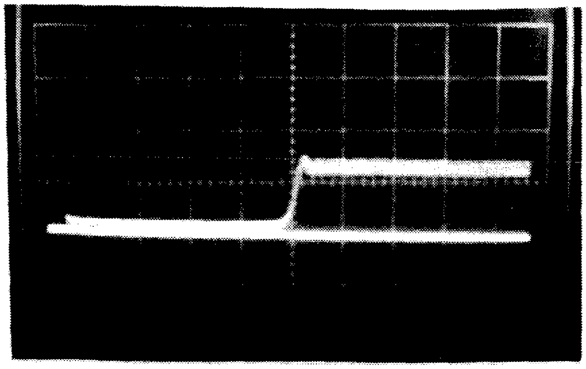


Figure 7—Output from sensor array

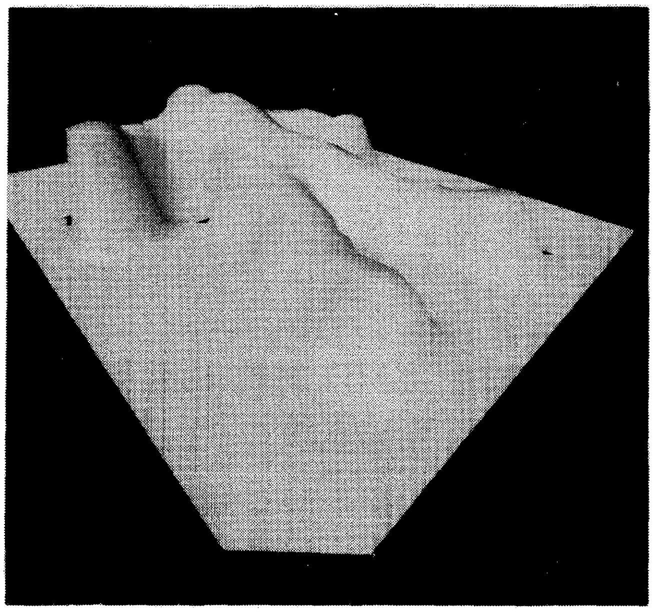


Figure 8—Reconstructed surface from laser scan

The data for Figure 8 was acquired by Fuchs,[3] by scanning a human model with a laser beam. The original Twinkle Box system detected the individual reflected spots of lights and calculated their three-dimensional positions.

CONCLUSION

We have presented a system which allows the use of very simple linear sensors to obtain 3-D information. These sensors involve no optical lensing systems or mechanical movement, and their use in the system leads to simple, tractable mathematical calculations. Although our current system was inspired by and uses linear CCD arrays, any sensor system capable of reporting values of a shadow edge position along one dimension can be used, with little or no changes in the total system. More expensive sensors may be used to increase sensitivity and/or resolution as needed.

It is hoped that this new system will make acquisition of 3-D information significantly faster, easier and less costly, thereby encouraging an expansion of 3-D man/machine interaction.

ACKNOWLEDGMENT

The authors are grateful for ideas contributed by Larry Evans of Evans and Sutherland Computer Corp., Salt Lake City, at the inception of this research project. Gordon McNorton assisted with the sensor/computer interfacing.

REFERENCES

1. Binford, T. O., "Visual Perception by Computer," *Proc. of the IEEE Conf. on Systems and Control*, Miami, Dec. 1971.

2. Burton, R. P. and I. E. Sutherland, "Twinkle Box—A Three-Dimensional Computer Input Device," *AFIPS Conference Proceedings,* Vol. 43, May 1974.
3. Fuchs, H., "The Automatic Sensing and Analysis of Three-Dimensional Surface Points from Visual Scenes," Ph.D. thesis, Univ. of Utah, Salt Lake City, August, 1975.
4. Gara, A. D., R. F. Majkowski, and T. T. Stapleton, "A Holographic System for Automatic Surface Mapping," General Motors Research Labs., Research Publication GMR-1342, Warren, Mich. March 1973.
5. Herron, R. E., "Biostereometric Measurement of Body Form," *Yearbook of Physical Anthropology,* Vol. 16, 1972, pp. 80-121.
6. Newman, W. M. and R. Sproull, *Principles of Interactive Computer Graphics,* McGraw-Hill, New York, 1973.
7. Roberts, L. G., "The Lincoln Wand," MIT Lincoln Laboratory Report, Lexington, Mass., June 1966.
8. Science Accessories Corp., "Graf/Pen Sonic Digitization," Science Accessories Corp., Southport, Conn., 1970.
9. Selective Electronic Company AB, "Selspot System," Molndal, Sweden.
10. Spreight, B. S., C. A. Miles, and K. Moledina, "Recording Carcass Shapes by a Moire Method," *Medical and Biological Engineering,* March 1974, pp. 221-226.
11. Sutherland, I. E., "Three-Dimensional Data Input by Tablet," *Proc. of the IEEE,* Vol. 63, No. 4, April 1974, pp. 453-461.

APPENDIX A—BACKGROUND

The widespread interest in acquiring 3-D information has stimulated the development of a variety of methods and systems. These systems can be loosely categorized into two general classes:

I. those systems which digitize the location of a special position-indicating device ("wand"), which is moved manually to specific points of interest, and
II. those systems which digitize (manually or automatically) an entire surface of interest.

Examples of type I systems include the early Lincoln Wand of Larry Roberts[7] and the current commercial product, the Graf-Pen.[8] Both systems digitize the position of a small ultra-sonic transmitter located at the tip of the wand. More recently, the Twinkle Box of Burton and Sutherland[2] and the Swedish commercial product, Selspot,[9] are both able to digitize a multiple number of points of light concurrently, using time-division multiplexing.

Type II systems, being usually more complex than type I, tend to be laboratory developments oriented toward a specific application: the digitization of automobile bodies using holographic techniques,[4] the measurement of the surfaces of animal carcasses by interference methods,[10] the scanning of small object clusters for automatic scene analysis using optical triangulations,[1] and the digitization of human body contours by classic stereometric procedures.[5]

This last application—the digitization of human body form—seems to be of greatest current interest. In this area, 3-D surface digitization has been used for such diverse studies as anthropomorphic dummy evaluation, spinal deformation measurements, and tumor detection and tumor growth monitoring. However, the presently implemented and proposed digitization methods are incompletely automated, and tend to be cumbersome and expensive, thus requiring much human intervention and effort. Additional processing steps, such as the transfer of images to photographic materials, often cause additional delays and expense.

Wand-like devices are also inconvenient for surface digitization in that the wands have to be moved by the user to every point to be measured. This can be a very time consuming task, sometimes made impossible if the surface to be digitized moves too rapidly. However, wand-like devices are particularly useful when real time tracking of only a few points is needed. For example, in limb mobility measurements, "wand tips" (small light sources) placed on a subject's leg allow leg movement to be tracked in real time.

The most direct predecessor of our system is the aforementioned Twinkle Box. We have attempted to retain most of its capabilities as well as overcome its major flaws. We consider the major assets of the Twinkle Box to be

(1) the digitization of multiple points,
(2) the use of arbitrarily positioned single-dimensional sensors instead of two-dimensional devices, such as TV cameras,
(3) a simple, unified approach to the problem of converting raw sensor measurements into a 3-D position in Cartesian coordinates.

The Twinkle Box's limitations, as reported by Burton and Sutherland, can be primarily attributed to the basic electro-mechanical sensor design. The sensors consist of spinning discs sandwiched between an optical lens and a photomultiplier tube. The system's four sensor units, each with a 22-inch diameter disc spinning at 3500 rpm, create a very noisy working environment, and the 2 h.p. motors needed to spin these discs severely overheat the sensors after about 30 minutes of operation. The disc fluctuations and non-synchronous rotations cause additional difficulties.

Our system eliminates the mechanical and optical problems by using an entirely new sensor design. The system also adds two new features (1) a focused light beam deflection system to enable automatic surface digitization and the tracking of unmarked objects; and (2) a network of microprocessors for signal pre-processing and other parallel computations. These features allow our system to be used both in the "wand" tracking mode of type I systems and also the automatic-surface-digitization mode of type II systems.

Strategies for the successful introduction of computer technology in a mental health care setting—The problem of change

by JAMES H. JOHNSON, THOMAS A. WILLIAMS, RONALD A. GIANNETTI, and L. J. SCHMIDT
University of Utah
Salt Lake City, Utah

ABSTRACT

An innovative approach to mental health care delivery which makes use of on-line computer technology is described. Results of a formal evaluation study of this system are summarized. These results indicate that the computer supported system has the capability of effecting improvements in mental health care delivery. However, it has also been learned that clinical staff have difficulties in accepting innovative approaches to patient care, and furthermore, that their resistance to change limits the effectiveness of the new system. Strategies to effect clinician acceptance of these changes and the positive results of these strategies, are also discussed.

We are no longer able to satisfy patient needs within the constraints of available economic resources with traditional approaches to the delivery of mental health services.[1] The allocation of resources to mental health facilities has not kept pace with the increasing demand for services. May and Cohen have suggested that solutions to this problem await major innovations in the design of hospital admitting systems.[2] They hypothesize that changes in admissions procedures aimed at providing "a highly selective evaluation and distribution operation" (p. 737) will lead to more efficient and effective provision of services. However, it has been impossible to implement such systems using traditional procedures. The primary obstacles have been the time and expensive manpower required to complete an evaluation that is sufficiently comprehensive to permit sensible intake, treatment, and other patient-management decisions. Typically, several days elapse before the mental health team can complete and report the results of the various psychological, social, and medical examinations. Thus, in a traditional system, intake and initial treatment decisions are made without the benefit of a comprehensive data base.

As a potential solution to this problem, the Psychiatric Assessment Unit (PAU) was established at the Salt Lake City Veterans Administration Hospital.[3–5] The PAU was conceptualized as a means for optimizing assignment of patients into the treatment system in order to improve both individual care and resource utilization. The functional design of PAU differs significantly from that of the usual psychiatric admitting system. Comprehensive psychological, social, and physical assessments are performed at the time of application for care. These assessments provide the basis for determining an optimal assignment to an initial treatment program. An on-line computer system that functions in a real-time mode collects patient information and summarizes, interprets, and prints reports that are available immediately for use in clinical intake decision-making. A substantial amount of clinical information is collected on-line, utilizing the patient's responses to inquiries presented on a cathode-ray tube (CRT) terminal. The remainder of the intake data base is collected by mental health paraprofessionals who record data in response to structured examination schedules which are also presented on the CRT. This design, which maximizes utilization of self-report and paraprofessional introduced data, permits implementation of real-time "evaluation" at intake, yet at a cost which is feasible in most mental health settings.

Patient flow through PAU is relatively structured. A receptionist opens the patient's computer file by entering basic identifying data into a CRT. The patient is instructed in the use of a CRT and completes an 18-item measure, the Q1, which predicts the likelihood that the patient can supply valid self-report information.[6] The PAU coordinator, a mental health professional, then interviews the patient and gathers information about the circumstances surrounding the application for care, the chief complaint, and the history of the present illness. If the patient fails the Q1 test and is judged to be in need of immediate inpatient care, he by-passes the regular PAU assessment system at admission and is evaluated later when his clinical condition has improved sufficiently to permit self-report testing. In the usual case, self-report testing is possible at intake, and the comprehensive assessment process begins. First, a health technician performs a computer-prompted screening physical examination. Results are entered into the CRT. The patient then begins self-report testing on the CRT. The standard self-report evaluation consists of the Beck Depression Inventory, the Beck Hopelessness Scale, the Minnesota Multiphasic Personality Inventory, the Briggs Social

History, a problem list, a modification of the Shipley-Hartford test of intellectual performance, and a screen for organic impairment. A PAU interviewer then administers a structured mental status examination and records the data on a CRT. As each procedure is completed, the computer analyzes the responses and prints a narrative report. The PAU coordinator reviews these reports and summarizes the results by selecting problems from a precoded problem dictionary and rating the severity level of each problem.[7] The computer report of this procedure becomes the "initial problem list" for the patient's medical record. Based upon this "initial problem list," the coordinator, in consultation with a psychiatric physician, if needed, determines an optimal initial treatment disposition for the patient.

A formal evaluation study was performed to compare the PAU system with the traditional admitting physician intake procedures on a variety of process and outcome measures.[8,9] While some data analyses are still being completed, the available results strongly support the conclusion that, when the PAU system is compared with traditional hospital intake evaluation approaches, it results in superior assessments which have a positive impact on treatment outcomes and which decrease the cost of patient care. Furthermore, patients report that they prefer the prototype computerized evaluation system better than they do the traditional approach.

The PAU represents a positive innovation in mental health care delivery. On-line computer technology has been integrated into the operation of a comprehensive psychiatric treatment program as one solution to the problem of meeting increased demands for services with limited resources. Evaluation studies have shown that the PAU system is highly successful in this enterprise. However, despite the fact that the PAU has been shown to have a positive impact on mental health care delivery, there have been a number of problems attendant with its implementation. As with any other radical systems innovation, the approach utilized by the PAU has challenged deeply ingrained notions associated with mental health treatment. This has been true for most personnel associated with the Salt Lake City VA Hospital Mental Health Treatment Services, but it has been especially true for those individuals assigned to work within the PAU. The purposes of this paper are: (a) to discuss some problems associated with staff acceptance of a computer-assisted intake system, (b) to review our efforts in working to solve these problems, and (c) to present the outcome of our efforts. It is hoped that this paper will stimulate others to consider problems and solutions in the acceptance of change, especially as related to the acceptance of innovations utilizing computer technology.

THE PROBLEMS OF CHANGE

The PAU was designed and implemented as the central intake point for both inpatient and outpatient mental health care at the Veterans Administration Hospital in Salt Lake City. As such, it became the primary bond between a number of independent treatment units. Some of the ramifications of this position were unanticipated. We soon discovered that the PAU, by virtue of its central position, was expected by many to meet existing needs in other areas of the treatment system. Therefore, PAU's mission quickly evolved to include a number of functions for which it was not designed. These included crisis intervention, initial prescription of medications, liaison with the local judicial system, arbitration of inter-ward transfers, telephone referral service, and completion of routine evaluations which were not related to intake. Rather than merely performing intake evaluations, the PAU became a center for communications and referral for the entire mental health care system, which it was designed to serve as but a portal of entry. Thus, the initial problem of change associated with the implementation of the computer-assisted psychiatric assessment system was one of defining the actual limits of its operation.

A second major problem of change was also unforeseen, although it was addressed in the original design of the system. The PAU evaluation protocol called for an extensive assessment prior to intake triage. Patients were to be examined for several hours prior to treatment decision making. Treatment staff, used to the idea of making quick decisions without comprehensive information, were often impatient when faced with the wait for extensive evaluation. They did not understand how to proceed to decision-making, based upon the large data base now potentially available to them. What they could understand was the need to complete mandatory administrative admission procedures and laboratory tests. The comprehensive intake evaluation was viewed as but a hindrance to completing these other routine but mandatory tasks. As a result, treatment staff began to pressure PAU staff to make triage decisions prior to the completion of the complete evaluation. In an attempt to accommodate to these pressures, PAU staff began making premature triage decisions in order to facilitate processing of the patient for admission to a treatment program. However, once the patient left the PAU area, he was seldom returned to complete his evaluation. Furthermore, even if the patient did return, the concept of making decisions on the basis of a complete data base was rendered ineffective, since the intake triage decision had already been determined.

The difficulties mentioned above resulted in a final major problem: PAU staff began to lose sight of the original goals of the project. On the one hand, they were confronted with demands for a large number of services which were not planned for in the original design of the system. On the other hand, they were receiving pressure from treatment staff members to shorten the evaluation process and move patients into treatment more quickly. This was reflected in a progressively decreasing number of completed evaluations, potentially neutralizing the project's aims.

In summary, there were several problems of change associated with the introduction of the computerized PAU. Many of the ramifications of a unifying systems concept were not forseen. Treatment staff did not fully understand the concepts involved with an intake system which was radically different from that which they had experienced previously. PAU staff lost their commitment to the original goals of the project.

TABLE I—Workload Analysis for Staff Functions of The Computer-Assisted Psychiatric Assessment Unit

Function	Percent	Total
Computerized examination related		
Read chart	1.15	
Talk to pt.	3.69	
Mental status	3.44	
Problem list	1.48	
Physical	3.93	
Review tests	0.57	
Other	0.08	14.34
Noncomputerized examination related	0.98	0.98
Staff monitoring of patient self-report		
Start test	1.80	
Correct error	0.33	
Call and collect reports	2.87	
Update test log	0.74	
Take reports to ward	0.00	
Other	0.49	6.23
Hospital administrative reporting		
10-10M	1.23	
Chart notes	0.16	
Short form admission	0.57	
Referral form	0.66	
Routing slip	0.16	
Chart processing	0.90	
Pt. log	2.05	
Typing	0.49	
Other	5.74	11.96
Consultations with physicians	1.56	1.56
Pt. management		
Emergency treatment and crisis	0.73	
Med. related	0.49	
Advice or info	1.07	
Miscellaneous	0.82	3.11
Communications		
Info to/from SEU staff or personal calls	2.87	
Computer related	0.98	
Processing or status of pt.	8.69	
Paperwork related	1.07	
Staff or resources	0.90	
Patient (clinical)	2.05	
Psych inpt.	0.98	
Psych opt.	0.74	
Community resources	0.33	
Miscellaneous	0.41	19.02
Other	42.73	42.73

PROCEDURES USED IN ATTEMPTING TO SOLVE THE PROBLEMS OF CHANGE

A study group composed of senior clinicians and project leaders was formed to correct the problems which had developed during the initial implementation of the computer-assisted Psychiatric Assessment Unit. To facilitate better understanding of the problem, it was determined to obtain a work sampling of PAU to determine the current functional resource allocation. Twenty-one hours of work time were observed, at the rate of one observation per PAU staff member per minute, for a total of 1220 observations. The results of this analysis is presented in Table I. It was apparent from this analysis that only 21.5 percent of staff time was being allocated to patient assessment functions, while a relatively large proportion of resources was being allocated to initially unanticipated functions such as: (a) VA forms completion (11.96 percent), (b) consultations and patient management (4.67 percent), (c) communications (19.02 percent), and (d) miscellaneous (42.73 percent).

Based on this analysis, patient flow through the PAU was reorganized. Coordination and assessment functions were separated. One person was identified as the coordinator for communications among PAU staff, and between PAU and other components of the VA facility and local community facilities. An additional full-time person was recruited to handle all crisis functions. All other PAU staff members were assigned responsibility for specific assessment duties. Wherever possible, the secondary functions and tasks which PAU had assumed during the initial period of implementation were eliminated. The evaluation procedure was standardized according to the original design. A structured scheduling system was implemented. In this system both evaluation staff time and self-report CRT time were scheduled to increase the efficiency of utilization of these resources. A tracking system was installed in order to monitor the progress of the patient through the comprehensive evaluation procedure. This minimized the possibility that portions of the evaluation would be inadvertently omitted. These steps were taken in order to reorient PAU staff members' attention back to the original goals of the intake assessment concept and, also, to take into account the unforeseen tasks which were encountered during the initial implementation of the prototype central intake triage unit.

An in-service staff education program was initiated in order to deal with the problem of treatment staff acceptance of this PAU concept. Results of the formal evaluation of the PAU system were presented to treatment unit administrators and clinicians. Introductions to the value and use of the various PAU assessment instruments were presented to staff members of the various treatment units. Informal discussions were held to improve communications between PAU and treatment unit personnel. The goals of these undertakings were to increase understanding of the PAU concept and its value and, thereby, to facilitate better clinician-acceptance of the system.

RESULTS OF A STUDY TO EVALUATE PROCEDURES USED TO SOLVE PROBLEMS OF CHANGE

It was hypothesized that if treatment staff acceptance of the PAU concept was improved, and if PAU staff members could be reoriented to the original goals of the project, then there would be an increase in the number of assessment packages completed. A "complete assessment package" was operationally defined as completion, *prior* to triage decision-making, of *all* procedures specified in the original design of the PAU. In order to test this hypothesis the number of complete evaluations was counted for three weeks before, and again for three weeks after, implementation of the changes enumerated in the preceding section of this report. Six evaluations were completed in the three weeks prior to implementation. Thirty-nine evaluations were completed in the three weeks following. This represents a 650 percent increase in the number of evaluations completed. Since the introduction of new operating procedures inevitably results in some initial confusion and error, an even greater gain is anticipated to accrue, as the PAU staff become accustomed to the new procedures.

SUMMARY AND DISCUSSION

In this paper an innovative approach to mental health care delivery is described. This PAU approach involved the introduction of on-line computer technology in order to facilitate an improvement in service delivery. Even though this system has now been shown experimentally to effect positive changes in the care of mental health patients, there have been problems in the acceptance of the system by both treatment and PAU staff members. These problems effected day-to-day operations of the system. An approach to solving these problems, and results of a study aimed at evaluating the success of this approach are also presented.

It is evident from this research that improved systems concepts utilizing computer technology will not necessarily result in an improvement in patient care, even if the proposed concept provides the demonstrated capability of such an improvement. There must be staff acceptance of an innovative concept in order to achieve integration of that new idea into an existing system. Where computer technology is a component of an innovative approach, there appears to be a ready resistance, especially in mental health care delivery settings. Staff members trained and experienced in traditional settings tend to view computerized techniques as dehumanizing to the patient. Presumably, they are, therefore, inclined to view such approaches with suspicion. This suspicion must be recognized and dealt with, if the use of computer technology is to obtain staff acceptance.

Those of us who have been conducting research on the use of computer technology in mental health care settings have become increasingly aware of the fact that the implementation of innovative approaches is impeded more by the problem of clinical acceptance, than by the present state of computer technology. The present technology is sufficient to effect significant improvement in the quality of mental health care delivery. However, technology alone is not sufficient to effect the desired improvement in care without significant staff resistance. It is hoped that this paper will stimulate additional progress in this latter arena.

REFERENCES

1. Lanyon, R. I., "Technological Approach to the Improvement of Decision-Making in Mental Health Services," *Journal of Consulting and Clinical Psychology,* 1972, Vol. 28, pp. 431-438.
2. May, P. R. A. and J. Cohen, "The Mental Health Engineer: an Agent for Institutional Change," *Hospital and Community Psychiatry,* 1974, Vol. 25, pp. 735-739.
3. Johnson, J. H., R. A. Giannetti and T. A. Williams, "Real-Time Psychological Assessment and Evaluation of Psychiatric Patients," *Behavior Research Methods and Instrumentation,* 1975, Vol. 7(2), pp. 199-200.
4. Johnson, J. H. and T. A. Williams, "The Use of On-Line Technology in a Mental Health Admitting System," *American Psychologist,* 1975, Vol. 30, pp. 388-390.
5. Williams, T. A., J. H. Johnson and E. L. Bliss, "A Computer-Assisted Psychiatric Assessment Unit," *American Journal of Psychiatry,* 1975, Vol. 132, pp. 1074-1076.
6. Johnson, J. H., T. A. Williams, D. E. Klingler and R. A. Giannetti, "Interventional Relevance and Retrofit Programming: Concepts for the Improvement of Clinical Acceptance of Computer-Generated Assessment Reports," *Behavior Research Methods and Instrumentation,* in press.
7. Giannetti, R. A., J. H. Johnson, T. A. Williams and C. F. McCusker, "An On-Line Problem-Oriented System for the Evaluation of Mental Health Treatment Service," *Behavior Research Methods and Instrumentation,* in press.
8. Klingler, D. E., J. H. Johnson and T. A. Williams, "Strategies in the Evaluation of an On-Line Computer-Assisted Unit for Intake Assessment of Mental Health Patients," *Behavior Research Methods and Instrumentation,* 1976, Vol. 8(2), pp. 95-10.
9. Klingler, D. E., D. A. Miller, J. H. Johnson and T. A. Williams, "Process Evaluation of an On-Line Computer-Assisted Unit for Intake Assessment of Mental Health Patients," *Behavior Research Methods and Instrumentation,* in press.

Database management for clinical trials

by JOHN M. LONG and JOSEPH R. BRASHEAR
Hyperlipidemia Program
Minneapolis, Minnesota

ABSTRACT

The Authors describe how they used a standard database management system (System 2000) and a computer utility to build a sophisticated medical records system in support of a national multi-clinic clinical trial. Privacy, protocol adherence, quality control and other key elements of an ethical clinical trial were satisfied at a fraction of the development cost for the more traditional approach of building a customized system. The authors feel that old lessons learned in other areas regarding the balance of manual to automated systems and the use of standard software are being re-learned for clinical trials. Neither a medical setting nor a clinical experiment changes the basic issues of good systems design. The possibility of using clinical trials as a test bed for developing medical information systems is also proposed.

In this paper we will describe our experiences in the development of a data processing system for a national, multi-clinic, controlled clinical trial. It is hoped that the description will demonstrate that controlled clinical trials are neither unique nor specialized users of medical information systems. In fact, clinical trials could become a vehicle for testing new concepts in information system design for medical applications.

The clinical trial is a controlled experiment designed to test the effects of a medical treatment on human subjects. It is the last step in the medical research process. It is the test to determine whether a drug, surgical procedure, or medical device is safe and effective for use in day-to-day medical practice, as well as a medical research device to test alternative hypotheses.

A controlled clinical trial requires detailed analysis of the medical well-being of large groups of patients. The subject population is well defined. Major aspects of their health care are carefully controlled. Clinical data is collected in a standard manner according to a pre-defined trial protocol. Patients are followed (indeed pursued) over long time spans to insure complete data collection and statistical reliability.

Our clinical trial, funded by a grant from the National Heart, Lung and Blood Institute, is formally known as the Surgical Control of the Hyperlipidemias, Secondary Prevention Trial. We simply refer to it here as the Hyperlipidemia Clinical Trial or Study.

The Hyperlipidemia clinical trial is a definitive test of the lipid hypothesis. It is designed to seek an answer to the question of the effect of maximal cholesterol reduction on patients with known atherosclerotic heart disease. One thousand patients are being randomly assigned to either a treatment or a control group. The patients will be followed for five years after randomization. Periodic clinic visits are scheduled to ascertain the extent of atherosclerosis and to detect the occurrence of any other medical problems. Trial protocol requires a highly restrictive recruiting and screening process to obtain 1,000 patients for randomization. It is anticipated that clinical data will be collected from at least 10,000 patients during the screening phase of the trial. Twenty-two different forms are used for data collection. They range in size from a single page log-in report to a 31 page medical history and physical form. Obviously both patient management and data management are primary concerns of the trial.

In addition to these physical parameters, we were faced with a set of environmental constraints. Form design and patient recruiting had already started when computer systems personnel were hired. We expected to be inundated with forms at any moment, and certainly before systems and procedures were fully established. Additionally, in the rush to recruit, incomplete attention had been given to the practical day-to-day aspects of how to handle data and readily analyze the end results of the trial. We were forced into early recognition that actual computer processing represented only one of three major data processing systems areas. We were forced by practical circumstances to relearn this old lesson which should not have been forgotten.

Given these parameters and constraints, a system had to be developed that could be operational quickly at reasonable cost. The data processing system is a necessary tool for a clinical trial. As in all of health care, the computer is not the paramount concern and should not become a major expense item. In addition, the entire system had to be responsive and easily modified. A medical researcher simply cannot anticipate all of the effects of his procedures. Medical ethics require constant surveillance for unanticipated events and possible procedure termination.

Other constraints on system design were no different than those facing a systems analyst in any application. Privacy and security of data must be considered whenever data is collected about individuals. Economics dictate that novel developments in supporting disciplines cannot compromise the integrity of the user and his application. Neither a medical setting nor a clinical experiment change the basic issues of good system design.

Trial management was faced with a classical computer system dilemma. There are four obvious alternatives in building a data processing system: develop a custom system, still a tantalizing alternative for a systems analyst; share customized hardware and software with other similar applications; develop a generalized hardware/software system specifically for clinical trials, or use a computer utility and available standard software. The Hyperlipidemia Clinical Trial chose the fourth alternative.

We are using a shared CDC Cyber 74 facility at the University of Minnesota and System 2000 as our database management system. This approach has proven a good one. Starting in May of 1975, we had control of documents and protocol adherence via time-sharing terminals by the end of that year. The basic system was fully operational in May of 1976. By this we mean that we were able to enter clinical data in a controlled operation, provide data accountability, and retrieve reports based on that data. Improvements have continued and some work remains to be done in data analysis and statistical reporting, but data processing, including programming or systems analysis, is not responsible for any delays. We have been able to generate all reports very shortly after they have been defined. The basic system was developed in less than one year at a cost of less than $60,000. Of course, some of these savings are lost in higher operating costs which we will discuss later.

An interesting and significant body of work has been done in peripheral areas as well, ranging from data security and privacy to the social psychology of medical data processing. We have also been able to be truly responsive to our users. A great deal of time has been devoted to gaining input and support regarding reporting formats. The generality of standard software has permitted changes with little disruption to the automated portions of our system. Overall, we feel that we have been able to emphasize those areas where emphasis rightly belongs. Rather than expend all our resources on computers and software, we have emphasized data content and editing, meaningful data analysis, and effective computer generated reports.

Our medical records database design was based on three primary concerns. First, we needed ready access to large volumes of clinical data. Secondly, we are required to monitor adherence to the trial protocol by every patient from each clinic. And finally, we had to control data flow through a complex editing and certification process. These concerns led to consideration of three separate System 2000 databases. The Main, or scientific, database would contain all of the clinical information collected for every patient, that is, the patient's individual medical record. The Administrative database would hold the information needed to control data collection and adherence to trial protocol. A Locator database would be used to monitor the flow of data forms through the entire system.

These functions are analogous to those found in a traditional circulating library. The Main database is equivalent to the card catalog. We use it to determine what data we have on each subject. The Locator database is the analogue of the checkout or circulating system, tracing the current location or user of individual holdings.

Function or method of use dictated the structure of the three databases and resulted in three different designs within this rather loose restrictions of System 2000. The simplest final structure was that of the Locator database. The automated version was abandoned. The checkout and circulation control function appears to operate better as a manual procedure.

The Administrative database is on-line and is accessed through a time-sharing terminal. Information is added or updated as forms are received. An entry is made for every data form which arrives from a clinic. This is a fundamental step in imposing control over forms flow and protocol adherence. The database has a tree structure with three hierarchical levels. All data elements, except a text memo, are inverted. That is, they are all key elements and may be used to qualify retrieval or access operations. Complete inversion has given us the capability to retrieve formatted administrative reports with minimal programming effort.

The Main database, holding the medical records, is also maintained on-line, but is updated periodically in a batch processing mode from punched cards. Although extensive editing and consistency checks are made in the update programs, it is expected that manual control of the input data will make it relatively error free. Entry into the Main database is the last step in the manual editing and checking chain. This database is also structured as a three level, hierarchical tree but the structure is unique. Our approach was dictated by storage costs and limitations, rather than access modes. You will recall that the clinical data is submitted on a large number of complex forms. A straightforward approach would be to define a data set or repeating group with unique data elements for each form. This approach rather quickly leads to realization of System 2000 overhead costs and limitations on numbers of unique elements. The design used was to designate the third level as a two element data set containing a question number and an answer. Access to information in the database requires detailed knowledge of each form and explicit definition of data by question number. This has not been a severe restriction and is offset by reduced computer operating costs. In fact, requiring knowledge of the data form has turned into an advantage. Requests for reports have been unusually brief and rational.

By using a standard software package, System 2000, we were able to minimize development time and cost. But, obviously, the system is not an optimal system for this particular application. For example, the ratio of total database size to actual information is greater than five to one. We are paying four indexing characters for each information character. Part of this cost is attributable to full inversion, but a fair amount is pure overhead. For example, the cost to index a date is totally exorbitant in System 2000.

First, dates must be represented as ten character data elements. Further, if a date is a key element, 20 characters are required to index each unique value. System 2000 also imposes a second indexing scheme for multiple occurrences of each key value. An application requiring data set selection with date of birth as one of the criteria becomes very expensive. The Administrative database has five dates designated as key elements. Given that there is a need to index dates, we could devise a coding scheme which would minimize file overhead. But we would have to give up the straightforward System 2000 natural language capability to manipulate dates on storage or retrieval. We do not believe the increased storage efficiency would offset the resulting increase in programming costs.

We could cite a number of similar instances of balancing or optimization which have been considered. There are undoubtedly many more that have not been recognized as yet. The major point is, that by selecting a software package, we have the time now to consider optimization. The trial data system is operational. We enter data and produce reports on a production basis. We have a happy customer and the resulting leisure to consider system enhancements.

An additional benefit from use of System 2000 is reduction in continuing programming staff and costs. Reports are produced using combinations of System 2000 natural language and FORTRAN or COBOL. Data elements for a report are selected and retrieved from the database using a simple "LIST" command. The selected data sets are ordered and written onto a sequential, FORTRAN compatible, temporary file. The final report is produced either by a simple formatting program or by a package such as SPSS. The "LIST" command is easily constructed, and complex selection criteria can be readily changed. All report programming to date has been a part-time activity for a senior programmer. Report production costs have averaged less than 50 cents per page.

So far our decision to use a computer utility and standard software appears to be a good one. The bottom line, total cost or net profit, has been our measure. However, we have encountered enough problems along the way to confine our appraisal to one of cautious optimism. We have alluded to some of these problems earlier. We would now like to share them further.

Many of our development and operational problems can be grouped and labeled as environmental. We do not and cannot fully control our hardware or software environment. The trial is only one user of a university computer system and an indirect customer of the software house which produced System 2000. Even though a clinical trial might become the largest single user of this particular computer utility, it would still be a single user.

Production schedules for the Hyperlipidemia Trial are now largely based on University classroom assignments. Updating the Administrative database is referred to as a conversational mode operation. During the summer months or winter holidays, this is factual statement. Response times to individual entries or inquiries have been typically less than five seconds and often only one or two. However, during mid-quarter or final week, response times are measured in minutes.

The batch mode update to the scientific database requires over 110K of memory and as much as 60 seconds of CPU time. The priority scheme at the University optimizes through-put for student FORTRAN jobs. Obviously, a major update run by our project requires enough resources to reduce run priority considerably. An update run submitted in the morning during final examination week will test the computer system's mean time between failures.

These response time problems are a major frustration to the trial systems staff and management. But the bottom line is our major measure. We pay only for time used. The trial does not have the fixed expense attendant to a dedicated computer system. We should not expect the side benefits of a dedicated system. We have however, relearned another old lesson. Update runs are scheduled according to system loading. Batching data for a scheduled update appears to relieve the perceived pressure to rush individual forms to the terminal. We are experiencing better human editing and checking of data forms and a lower rate of rejection by the computer system on heavy system load days.

The elapsed time problems with batch updates have forced us to give close attention to file backup procedures. We are also looking into the use of update and suspense files as an adjunct to the central database. We should be able to reduce our system resource requirements and vulnerability considerably, by reviving these techniques. They appear to be compatible with our System 2000 database approach, and we do not anticipate major reprogramming costs for implementation.

So far we have discussed our implementation of one alternative approach to construction of an information processing system for a clinical trial. The use of standard computer resources and application of tested systems analysis techniques permitted attainment of trial goals in minimum time with minimal cost. This approach is certainly not new nor is it unique to clinical trials. Also, there is no doubt that another alternative could have been implemented to produce another optimal system balance. Indeed, other clinical trials have taken alternative routes to system implementation and have succeeded according to their measures.

The variety of paths taken to attain a common goal by different clinical trials points to a major side benefit which could be significant to all medical information processing. Clinical trials require systems to handle clinical records for ambulatory care, longitudinal analysis of clinical data, computerized medical record summary systems, and archival storage of clinical data. In fact, they need most of the tools we are developing for health care applications. There is no doubt in the minds of the computer community or the medical community that they are necessary tools. There is however, an undercurrent of doubt or concern over introduction of these tools into an existing system of health care delivery.

We contend that the controlled clinical trial is an ideal environment for development and testing of these new tools. A clinical trial, by definition, is a controlled environment. The patient population is stable, the trial has an endpoint in time, and the necessity for data collection is established. Further, installation of new procedures in a clinical trial would be a non-disruptive, parallel operation. A trial is not in the main stream of day-to-day health care delivery. A trial does, however, require most of the procedures used in the main stream.

Finally, a clinical trial is a research procedure. Both the patient population and the medical personnel are conditioned to accept new practices. Clinical trial medical staff are usually research oriented and more receptive to use of computers. This last may be the most important point. A prudent health care practioner must be a conservative when his patient is involved. This attitude leads directly to conflict when something new is proposed for patient care. This type of conflict can become irrational and irreconcilable and lead to rejection of the new idea. Unfortunately, the psychological process described is evident throughout the health care industry today.

This potential test-bed facility, the controlled clinical trial, is not a new practice. Drug trials have been run by the pharmaceutical companies for years. The National Heart, Lung and Blood Institute has funded at least six major trials in the last fifteen years. These have required from 1500 to over 10,000 patients for each trial. Anticipated trial duration ranged from three to six years or more. Of more importance to the computer community, the clinical trial is here to stay. Recent legislation requires that all new medical devices, such as pacemakers, undergo a clinical trial before being released to the public.

We seldom find a potential testing ground for computer development which has broad generalized needs, is truly non-disruptive, and is securely financed. Controlled clinical trials meet all these criteria and more. We must stop regarding trials as different or unique experimental procedures and begin to exploit them as a resource.

Data management for clinical research

by W. L. SIBLEY, M. D. HOPWOOD, G. F. GRONER, W. H. JOSEPHS and N. A. PALLEY
The RAND Corporation
Santa Monica, California

ABSTRACT

This paper discusses a prototype system intended for the personal use of physicians engaged in clinical research. In particular, the prototype is a highly integrated, interactive, minicomputer based data management and analysis system. The facilities offered by the system allow the clinical investigator to store, recall, analyze, and display his research data without resorting to computer programming. Modern data base techniques are available to the physician as aids in organizing, storing, and retrieving his data. The data base concepts are expressed in terms that are familiar to a clinical researcher.

PURPOSE

The purpose of this paper is to discuss a prototype system intended for the personal use of physicians engaged in clinical research. In particular, the prototype is a highly integrated, interactive, minicomputer based data management and analysis system.

THE CLINFO PROJECT

The CLINFO prototype data management and analysis system described in this paper has been developed as part of the CLINFO project, a scientific inquiry sponsored by the Division of Research Resources (DRR) of the National Institutes of Health (NIH). The goals of the project are to identify and characterize the information analytic tasks and the information flows in clinical research, and to develop methods for facilitating these tasks and flows. The project is being conducted by clinical investigators at the Baylor College of Medicine, the University of Washington, the University of Oklahoma, and Vanderbilt University; by information scientists at the Rand Corporation; and by staff members of the DRR.

We have thus far (1) interviewed clinical investigators both formally and informally., (2) characterized their information processing needs, (3) identified data management and analysis as major problems, (4) examined existing computer systems aimed at satisfying those needs, (5) designed, built, and installed three copies of a prototype system, and (6) started the analysis of instrumentation data and user interviews to help us evaluate the system.

References 1, 2, 3, and 4 discuss various aspects of the foregoing points.

CLINICAL STUDY DATA VOLUMES AND ACTIVITIES

A General Clinical Research Center (GCRC) provides facilities which clinical research personnel can use in conducting a variety of medical studies. Each study tends to be separate and distinct from other studies and provides a natural focal point for data collection and analysis. A large study involves approximately 75 subjects (patients) and a total collection of approximately 67,000 data values.[1]

Generally, a clinical study involves the following activities:

- During the study design the investigator decides what data are to be collected, at what rate, and in what volume.
- As the study progresses, data are collected and recorded in a central data file. Considerable care is taken to ensure the validity of those data.
- As the study progresses, the investigator reviews and summarizes the data. Part of the process of summarization involves transcribing the data into a form suitable for statistical analysis as well as the preparation of plots, graphs, and reports.

The CLINFO prototype is designed to accommodate 25 such studies.

SYSTEM OVERVIEW

The prototype is implemented on a commercial minicomputer (Data General Eclipse S/200) using multi-user extended BASIC.[5] Each study has its own portion of a 25 million character disk. When an investigator (or a member of his staff) uses the system, he interacts with the CLINFO software (primarily by responding to prompts) and is prevented from using BASIC procedures to affect his data.

Figure 1 illustrates the main features of the CLINFO

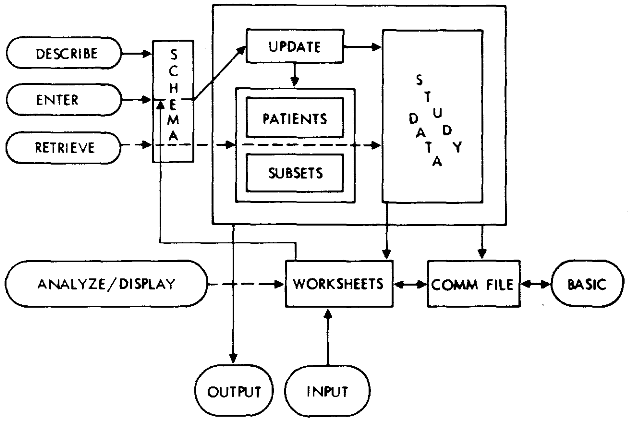


Figure 1—Command(- - -) and data (——) flow in the CLINFO prototype for a single protocol

prototype. The ellipses encompass the three general activities which were mentioned above. The rectangles indicate the various ways in which data (or descriptions of data) are stored in the prototype. The diagram encompasses a single study (or research protocol). This paper is concerned primarily with the activities labeled "DESCRIBE," "ENTER" and "RETRIEVE" and the data contained in the "SCHEMA," "UPDATE," "PATIENTS," "SUBSETS," and the "STUDY DATA" structures. The "ANALYZE/DISPLAY," "INPUT," "OUTPUT," and "BASIC" activities and their associated structures will be described, but in less detail.

THE DATA BASE

The organization of a CLINFO study data base is predicated on three observations:

- The fundamental unit of analysis is usually the patient.
- Data are collected about that patient over time.
- The data have natural groupings in time (e.g., the vital signs of a patient are sampled at essentially the same time).

The time-oriented nature of clinical research data has been discussed by Fries.[6] Dr. Fries' ideas have been further reinforced by our observations of the current manual techniques for recording that data. For example, data are recorded in patient flowsheets (one for each patient) which are two-dimensional arrays with the rows representing the data items to be measured and the columns containing the values of an item for different times of measurement. We have further observed that some of the rows can be grouped naturally into related items (examples of such groups are vital signs, blood serum tests, urine analyses, etc.). However, the groupings differ widely from study to study. The CLINFO term for a grouping is "PANEL." The concept parallels records or groups in the CODASYL Data Base Task Group (DBTG) report.[7]

PATIENTS/SUBSETS

The key structure is the PATIENT file which contains an eight character abbreviation for each patient (assigned by the clinician when the patient is added to the study) and an internal numeric key (assigned by the system). This file can contain up to 392 different entries. The entries are ordered alphabetically on the patient abbreviation to allow for efficient searching.

A subset is a named file (e.g., LOWBLOOD) with the same structure as the PATIENT file but containing patients with particular properties. No particular patient ordering is maintained for subsets. There can be as many subsets as space allows.

The numeric key associated with each patient references a block of data in a PATIENT DIRECTORY (not shown). That block contains some redundant identification data and two tables. The first table contains pointers to the patient's set of panels (see above) in the STUDY DATA file and counts of the numbers of instances of the panels. The second contains data concerning the state of the various EVENTS (to be described later) for this patient.

DESCRIBE/SCHEMA

The SCHEMA[7] provides the means for adapting the CLINFO data base structure to the needs of a particular investigator. It is a description of the panels and their contents. The schema exists in two forms: the external textual form and the internal compiled form. The former is for human interaction, the latter is for computer processing. The DESCRIBE activity is in essence an interactive editor which allows the clinician to create, edit, and compile his schema.

The schema describes three main entities: ITEMS of data, PANELS of items, and EVENTS triggered by the occurrence of certain values for particular items.

PANELS have eight character names, and references to them by name imply reference to all of their items. There are two types of panels, numeric and textual. The type of the panel determines the types of the data items included in that panel. Each panel can contain at most 30 items, all of which are either 32-bit floating decimal numbers, or character strings at most 70 characters in length. A sample panel entry in the schema might be

```
PANEL  1 hist   , patient ID & history,numeric;
```

Reading from left to right, this is the first panel (PANEL 1), its name is "hist   ", it contains "patient ID & history," and its items are all numeric.

ITEMS also have eight character names which are used to reference the data associated with an item in a wide variety of circumstances. Again, an item may either be numeric or textual. In addition, there are parts of the item description that serve to screen, validate, and possibly encode the value associated with that item. For example:

```
ITEM 9 HT ,Height    ,inches    ,num,range,(55,80),no;
```

Reading from left to right, this is the ninth item in the schema (ITEM 9), its name is "HT ", its values represent height in inches, its values are numeric and must lie in the range 55 to 80 inches inclusive, and its value need not be considered confidential ("no"). "Height" and "inches" are purely descriptive and do not imply automatic units checking by the system.

The sequence "num,range,(55,80)" illustrates one option available for data screening. Other options for items in numeric panels are:

- date,range(01031976,12251980)
  the item value is a date in the indicated range (1/3/1976 to 12/25/1980 inclusive).
- time,range(1300,1330)
  the item value is a time in the indicated range (13:00 to 13:30 inclusive).
- num,check(4),(7,10,6,5)
  the item value is numeric and one of the listed 4 values.
- char,code(3),(yes ,no ,unk)
  the item value is externally one of the listed 3 character strings and is carried internally as one less than the ordinal position of the string in the list (i.e., 0, 1, or 2).
- the data types date, time, and num may require no validating, e.g., num,none.

The textual panels and items are of the form:

```
PANEL  2,demo   ,demographic data,text;
ITEM   20,name  ,patient name,text(30);
```

The item "name" will then be allocated 30 characters in each occurrence of a "demo" panel.

EVENTS are time markers maintained by the CLINFO prototype system to aid the user in the retrieval of his data. It is common in clinical research to relate the response of a patient to a procedure by referencing some events which marks the application of that procedure. For example, blood sugar levels are measured at regular intervals after the ingestion of glucose. Events provide the means for dealing with relative time. A sample event is:

```
EVENT 3 test  ,glucose ingest,gluc  ,FIRST;
```

Reading from left to right, this is the third event in the schema, its name is "test," it relates to glucose ingestion, and it records the time at which the item "gluc" FIRST takes on a value. Time is taken here to mean a combination of date and time.

Events may be triggered by the "FIRST," "LAST,""MAX," and "MIN" values of its governing item. In addition, an event may be triggered at the first or last time the item takes on a particular value.

When the clinician is satisfied with his description of the data he intends to collect, he compiles the schema (the system provides appropriate feedback about errors in the compilation). He then requests that disk space be allocated to accommodate his study. He assists the allocation by estimating the number of patients he expects to include in his study and an approximate number of occurrences of each panel type for the average patient.

He is then ready to start entering patients and their data into the data base.

ENTER

The main function of the ENTER activity is to process data (entered interactively) in the light of the data screening specifications contained in the schema. The data that are accepted are not passed directly to the STUDY DATA file, but are recorded in an UPDATE file whose contents are merged with the STUDY DATA file at some convenient time. This buffering of the input data allows a simple structure for the STUDY DATA file, reduces the exposure of the system to computer malfunctions, and provides a means of reviewing the input data before it is actually recorded in the STUDY DATA file.

Again, a CLINFO data base is patient- and time-oriented. In order to enter data, a "context" must be established for that data. That is, the data must

- Belong to a patient already in the PATIENT file.
- Have an associated date and time of day at which the data are assumed to have been sampled.

After a context has been established (by interactive prompts and responses) the clinician may then enter values for items by either

- Typing the name of an item followed by its value or
- Typing the name of a panel, whereupon the system prompts for each of the items in that panel.

The context stays in force for all data entered subsequently until the clinician desires to change it. The context may be changed at any time by typing "patient:," "date:," or "time:" followed by the corresponding value.

As each value (including patient abbreviations, dates, times, and panel and item names) is entered, that value is checked for reasonableness. The patient abbreviation must be in the PATIENT file; dates and times must have the correct format; panel and item names must exist in the schema; the value for an item must satisfy the criteria in the schema. Incorrect values are refused and re-prompted for by the system. An exception is the case of an out-of-range numeric value which may be forced upon the system by entering the initials of the data enterer. Incorrect but reasonable values that have been accepted by the system may be corrected by re-typing them (perhaps after re-establishing the correct context).

ENTER has three additional functions:

- Adding patient abbreviations to the PATIENT file.
- Reviewing data in the UPDATE file.
- Copying and screening data from a worksheet into the UPDATE file.

As an alternative to item-by-item data entry, an entire two-dimensional array (i.e., a "worksheet") of data may be entered. A worksheet is a two-dimensional array of numeric entries extended to include 8-character labels for the rows and columns as well as a worksheet name, title, date of creation and date of last modification. A worksheet is stored by the system as a file with the same name as that of the worksheet. Worksheets provide the data organization required by the ANALYZE/DISPLAY activities. The row and column labels can be used to reference patients, relative times, and items. These labels are used to direct the data entry process. Data entry via worksheets provides the facility for entering "derived" data (i.e., data produced as the result of the analysis of raw data) into the STUDY DATA file. It also provides another data entry route that may be more convenient in some circumstances.

As indicated above, the clinician can interactively review the contents of the UPDATE file and edit it to the extent of deleting blocks of data.

When he is satisfied that the data are correct, the clinician can request that the UPDATE file be merged with the data already in the STUDY DATA file. The newly entered data are organized automatically into panels, those panels are marked with the internal patient identifier, the panel number, the date and time of sample, the date and time of data entry, and the initials of the data enterer, and the results are merged by date and time of sample. The merge process checks for existing instances of panels with the same date and time as those being merged and for item values previously entered in those instances.

The investigator is now in a position to retrieve his data from the STUDY DATA file and to format it in a variety of ways: into worksheets, as a display at the terminal, or as a file to be passed to an externally generated applications package (Such a file is a "communication" file in CLINFO terminology.).

RETRIEVE

The data in the STUDY DATA file may be viewed as values associated with points in a three dimensional space. Figure 2 illustrates that space. The axes are labeled "PATIENT", "TIME," and "ITEM" to reflect the parameters that uniquely identify a data value.

The three dimensional space is sparsely populated with data points, especially if the TIME axis is taken to represent absolute dates and times. Moreover, not all data are collected about all patients for all of the expected times.

It is the intent of the RETRIEVE activity to identify a planar slice of the data space and to abstract part of that slice into a two-dimensional form (i.e., a worksheet). Selecting a slice as illustrated in Figure 2 results in what was referred to above as a flowsheet for that patient. A slice parallel to the PATIENT-ITEM plane produces an array of items for a group of patients at a particular time. That time may be either an absolute date and time, or it can be a time relative to an event (e.g., two hours after glucose ingestion). Finally, a slice parallel to the PATIENT-TIME plane results in a worksheet containing the time history of a particular item for a group of patients (again either absolute or relative time).

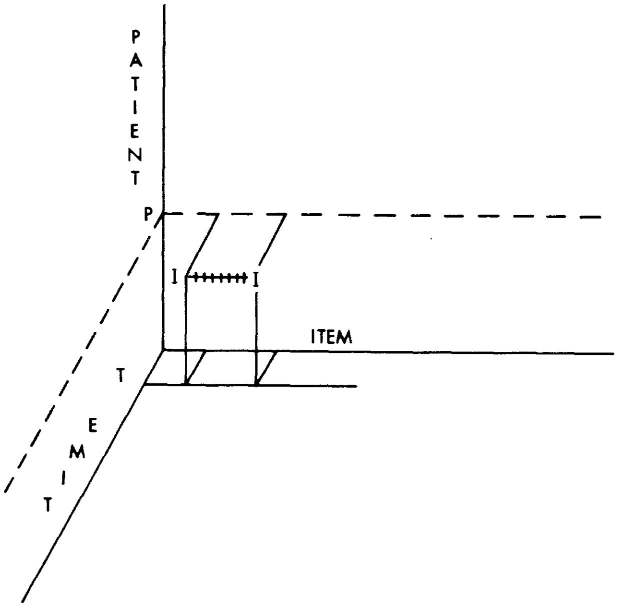


Figure 2—A three-dimensional representation of a time-oriented data file. I-I is a panel of items for patient P at time T

The clinician constructs his retrieval request interactively under the control of the schema. In particular, coded items (e.g., sex may be either "male" or "female", but is encoded as either 0 or 1) are referred to in terms of the external form ("male," "female") while the system deals with them as numeric values (0 or 1). The values of items play a role if the clinician wishes to restrict the panels being retrieved by specifying conditions that the values must satisfy.

When he completes the specification, the system surveys the appropriate parts of the STUDY DATA file and produces a worksheet with the rows and columns appropriately labeled.

The retrievals discussed above are limited to numeric (or coded) items. However, textual panels may be retrieved and displayed at the terminal or passed in their entirety to a communication file. This facility allows the clinician to review his notes about a patient or to pass demographic data to a BASIC program that, for example, produces mailing lists.

For those retrievals in which a list of patient abbreviations is appropriate (e.g., a slice parallel to the PATIENT-ITEM plane), the investigator may provide that list by giving the name of a SUBSET.

SUBSETS

As mentioned above, subsets are files containing patient abbreviations and numeric keys. The files have the same structure as the PATIENT file.

Subsets are created in a variety of ways:

- By listing patient abbreviations.
- By the usual set operations (e.g., intersection and union) among existing subsets.
- By requiring that a patient's data in the STUDY DATA file satisfy a set of conditions.
- By requiring that a patient's data in a worksheet (e.g., the values in a row labeled with the patient's abbreviation) satisfy a set of conditions.

The sets of conditions mentioned in the last two members of the above list are made up of lists of either conjunctions or disjunctions (but not both) of range restrictions on the values of items. A sample set of conditions is:

$$\begin{array}{rrll} \text{If} & 30 \leq & \text{age} & \leq 65 \\ \text{and if} & 120 \leq & \text{systol} & \leq 140 \\ \text{and if} & & \text{sex} & = \text{male} \end{array}$$

The last line illustrates the use of a coded item which takes on one of a set of discrete values.

In addition to the conditions placed on item values, the time at which a sample was taken can play a part in the selection. For example, the above conditions might be required to hold in the third hour after the ingestion of glucose.

In any case, the usual result of a retrieval using either a subset or all of the PATIENT file is a worksheet.

ANALYZE/DISPLAY

The CLINFO approach to data analysis is to abstract the desired data from the STUDY DATA file (which can cause very intensive accessing of the disk storage) into a worksheet which fits into the user's working space in main memory. The worksheet can then be manipulated efficiently, especially since its contents need not be re-abstracted from the STUDY DATA file each time a new question is asked of those contents.

A sample worksheet has the format illustrated in Figure 3.

```
WORKSHEET CLINICAL
TITLE Clinical Characteristics of Diabetic Patients
CREATED 12/17/75    MODIFIED 12/17/75
# OF ROWS    7
# OF COLS    4

  ROWS/COLS       1          2          3          4
   LABELS        age        sex      %idl wt    dur diab
   1 case 1       27        male       100         12
   2 case 2       24       female       98         11
   3 case 3       32        male        98         15
   4 case 4       34        male        94         20
   5 case 5       21        male        96          9
   6 case 6       23        male       . . .      . . .
   7 case 7       28       female       99         10

              (Note: . . . indicates missing values)
```

Figure 3—A sample worksheet

A worksheet can hold approximately 2200 entries and its dimensions can vary within that restriction. In the case above, the row labels are patient abbreviations and the column labels are item names. These labels and the contents of the worksheet would ordinarily be supplied by the retrieval process. In addition, Figure 1 indicates an INPUT facility that allows the direct creation of worksheets, the labelling of their rows and columns, and the entry of data into those rows and columns. This facility allows the investigator to use CLINFO's analysis and display features independently of the STUDY DATA file.

CLINFO maintains a "current" worksheet so that the same worksheet can be used in a variety of situations without the user respecifying it as the worksheet of interest. Most of the worksheet analysis and display functions apply to the current worksheet. The main worksheet manipulation facilities are:

- Select a worksheet as the current one by typing its name. If there is no worksheet by that name, create one as specified by the user.
- Display a worksheet at the terminal (see Figure 3).
- Label worksheet rows or columns.
- Enter data into a worksheet by row or by column or by individual cell.
- Discard (destroy) a worksheet.
- Sort a worksheet by rows or columns.
- Edit a worksheet by adding, deleting, or moving rows or columns.
- Copy a sub-array of a worksheet, with labels, into the same or another worksheet.
- Print a worksheet on the system printer.
- Display a list of all the worksheets in this study.

Worksheets created by the retrieval process or by direct means can be used by the analysis facilities. In general, the analyses can be made to apply to sub-arrays within the body of a worksheet. The results of some analyses may be stored in worksheets to minimize transcription and to make those results available for further analysis.

The CLINFO analysis facilities include:

- Descriptive Statistics (means, standard deviations, etc.).
- T Test.
- Chi Square Test.
- Linear Regression (simple and multiple).
- Analysis of Variance.
- Cross Tabs.
- Scatter Plots and Bar Charts.
- Histograms.
- Frequency Distributions.
- Normality Test.
- Non-parametric Paired Tests.

In addition, special calculations may be performed on a worksheet, and the result of these calculations are stored in the worksheet. This facility replaces in part the need for special purpose computer programming. These calculations are:

- Generate values in a row (or column) as the result of evaluating a single algebraic-like expression involving other rows (or columns) as variables.
- Generate truth values (0, 1) on the same basis.
- Perform special calculations such as cumulative sum, time differences, etc.

SUMMARY

The foregoing discussion does not describe the properties and facilities of the CLINFO prototype in their entirety. The intent is rather to outline those properties and facilities and to emphasize the idea of a personal, integrated, highly interactive system placed in the hands of personnel whose primary interest is to use the system as a tool.

At least one CLINFO prototype has been in daily use by medical personnel since January of 1976. The acceptance of the system seems to be good and it is being used in productive ways. Moreover it is being used personally by senior medical staff, that is, the research staff for which it was designed.

Our experience to date indicates that modern data base techniques (e.g. the SCHEMA), when expressed in the user's own terminology, are readily understood, accepted, and used by those personnel. The use of terms such as "PATIENT", "PANEL", and "WORKSHEET" and the time orientation of the STUDY DATA file are not accidental. They play an important role in making the computer scientist's techniques understandable by and useful to the clinical researcher.

Another point of interest is the utility of two distinct data structures, the STUDY DATA file (and its associated files) and worksheets. The STUDY DATA file acts as a pool of data whose contents can be abstracted and arranged into a form more suitable for viewing and analyzing (i.e., a worksheet). In addition, worksheets serve to store data and to communicate them between steps in the analysis process.

ACKNOWLEDGMENTS

Although the authors accept the responsibility for the words in this paper, a much wider group is responsible for the content those words express. We would like to express our appreciation to our co-contractors Howard K. Thompson, Jr., M.D., at the Baylor College of Medicine, T. Graham Christopher, M.D., at the University of Washington, and Arthur W. Nunnery, M.D., at the University of Oklahoma, our project officer, William R. Baker, Jr., Ph.D., of the NIH, and to our colleague Thomas L. Lincoln, M.D., of the Rand Corporation. They have all contributed significantly to the design and continuing development of the CLINFO prototype. This research was supported by Contract No. NO1-RR-5-2111 from the Division of Research Resources, National Institutes of Health.

REFERENCES

1. Palley, N. A., and G. F. Groner, "Information Processing Needs and Practices of Clinical Investigators—Survey Results," *AFIPS Conference Proceedings* (1975), Vol. 44, AFIPS Press, Montvale, New Jersey, 1975, pp. 717-723.
2. Groner, G. F., N. A. Palley and N. Z. Shapiro, "A Structural Characterization of Clinical Research Participants and Their Activities," R-1540-NIH, The Rand Corporation, January 1975.
3. Groner, G. F., M. D. Hopwood, N. A. Palley, N. Z. Shapiro, and W. L. Sibley, "A Plan for the Development and Evaluation of a Data Management and Analysis System for Clinical Investigators," R-1541-NIH, The Rand Corporation, August 1974.
4. Sibley, W. L., M. D. Hopwood, G. F. Groner, and N. A. Palley, "A Prototype Data Management and Analysis System for Clinical Investigators: An Initial Functional Description," R-1621-NIH, The Rand Corporation, August 1974.
5. Data General Corporation, *Extended BASIC User's Manual,* publication 093-000065-4, September 1973.
6. Fries, James F., "Time-Oriented Patient Records and A Computer Databank," *The Journal of the American Medical Association,* Vol. 222, December 18, 1972.
7. CODASYL Data Base Task Group (DBTG) Report, April 1971. Available from the Association for Computing Machinery.

Why restrict the modelling capability of codasyl data structure sets?

by CHARLES W. BACHMAN

Honeywell Information Systems
Billerica, Massachusetts

ABSTRACT

Several issues have been raised concerning changes to the capabilities of the CODASYL Data Description Language specifications for data structure sets. The paper argues against new restrictions suggested and for removal of existing restrictions. The issues are:

allow recursive set declaration
keep multiple member declaration
allow alternate owner declarations

The concept of "record-roles" is introduced to justify the need for these capabilities. The expanded capabilities described offer an alternate means of achieving the same end result without the need to introduce the "record-role" into the CODASYL Data Description Language.

INTRODUCTION

The concept of data structure sets has been well established through the publicity and use of I-D-S, the Honeywell Integrated Data Store system.[1-3] In recent years this concept has been adopted by the various CODASYL committees and imbedded in the CODASYL Data Description Language[4] and the COBOL Data Manipulation Language.[5] A number of hardware and software suppliers have implemented the data structure sets of these languages (DDL/DML) as part of their systems. They include:

IDMS	(Cullinane for IBM 360/370)
I-D-S	(Honeywell GE200, GE400, GE600, H6000)
I-D-S II	(Honeywell for H66, H64)
EDMS	(Xero Sigma 6/7/8)
DMS 1100	(Univac for Univac 1100 Series)
PHOLAS	(Philips for Unidata 7000, P1000)
PHOLAS	(Siemens for S4004)

Other implementations have been reported for CDC and DEC.

The capabilities of the data structure set, as developed in I-D-S and now defined in both the CODASYL DDL and the COBOL DML, provide for set-type declarations which:

(1) restrict the record type declared as owner to be different from any of the record types declared as member.
(2) permit declaration of one or more record types to serve as member records of an occurrence of a set type, and
(3) restrict to one the number of record types which can be declared to serve as owner records of occurrences of a set type.

THE PROPOSALS

This paper is a refinement of a working paper written in response to an assignment accepted at the IFIP-TC2 meeting on Data Description Languages held in Namur, Belgium in January 1975. There were three closely related proposals for changes to the CODASYL DDL discussed at that meeting. These proposals relate directly to three points enumerated in the prior paragraph. Assignments were given to defend a number of such proposals. The proposals interrelated and my working paper treats them as a single concept.

The first proposal, which was unanimously supported at the meeting, was to remove the restriction that the record-type declared as owner could not also be one of those declared as member. I strongly concurred with this proposal as a removal of an unnecessary restriction.

The second proposal was to add a restriction that only one record-type could be declared as member of a particular set-type. This proposal received mixed support. I strongly disagreed with this proposal, for essentially the same reasons that I support the first and third proposals. It adds an unnecessary restriction.

The third proposal was to remove the restriction that only one record-type can serve as owner of a particular set-type. This proposal received scant attention. The meeting did not express an opinion on the subject. I strongly recommended it, as I have done to the DBTG at least seven years ago. It is the removal of a restriction.

ARGUMENTS

There are several arguments for permitting a set-type to permit the record-type declared as member to be the same as the record-type declared as owner. There is a specific argument which will be treated first and then a general argument which relates to all three proposals mentioned earlier.

The specific argument treats the need for tree-like data structures, catalogues, organization structures and parsing trees. For these structures, it is necessary to support recursive sets, which provides the capability to build trees, with branches which have branches, which have branches, etc. An example of this is illustrated in Figure 1.

If all the straight lines in Figure 1 are considered to be "branches," then this structure can be built with a single record type and a single set-type. However, the membership of the "branch" record-type in the "fork" set-type must not be mandatory. A record declared to be as the lowest level branch is never a member of a "fork" set-occurrence. This special branch is characterized as the trunk. Figure 2 is a data structure diagram[6] which illustrates the branch/fork structure.

In this data structure diagram, the "fork" set-type is illustrated with a broken line, meaning that the "branch" record-occurrences are sometimes members of it. That is, they are members if they are not the "trunk" branch of the tree. Note that all branch record-occurrences, whether or not they are the trunk, own "fork" set-occurrence with zero, one, or more subordinate "branches" records.

The third Namur proposal, which supports alternative owners in data structure sets, would permit the trunk branch to be treated as a distinctly different type of entity. Figure 3 is a redrawing of Figure 1.

It illustrates a tree with one trunk and many forks and branches. When this is drawn as a data structure diagram, the illustration of Figure 4 is developed. This structure has some advantages.

The "branch" record-occurrences can now be treated as mandatory members of the "fork" set-type, i.e., no branches are floating in the air. This can be very important from a naming point of view, as each branch needs a field for its "branch name" while the trunk does not need such a name. Branches are frequently named with an articulated grammar. All the branch names at a single fork of the tree must have unique local names. The higher level branches (i.e., the branches which are farther from the trunk), are uniquely named by concatenating their unique local names to the tree unique name of the branch immediately below. This is expressed below in BNF (Backus/Nauer/Form)

⟨branch-name⟩::=⟨character-string⟩ |
  ⟨branch-name⟩⟨articulation-character⟩
  ⟨character-string⟩
⟨character-string⟩::=⟨character⟩ | ⟨character-string⟩
⟨character⟩
⟨character⟩::=a | b | c | . . . . | z | 0 | 1 | . . . . | 9
⟨articulation-character⟩::=any symbol which is not a blank
  or a character

In our information systems today, there are many examples of tree structures. In catalog (file) systems, we find

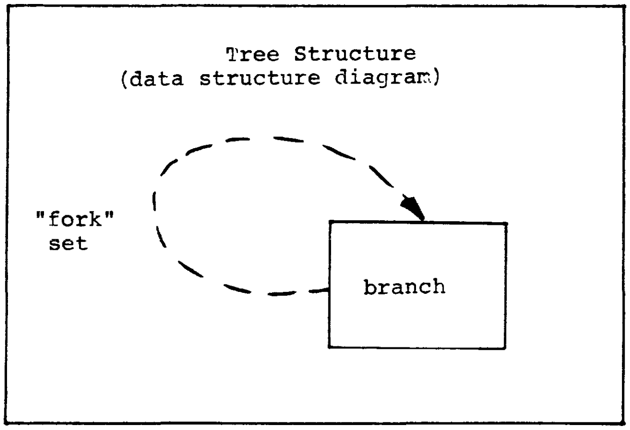


Figure 2

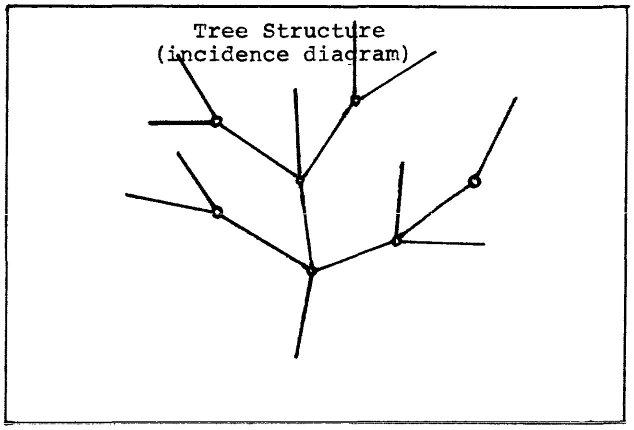


Figure 1

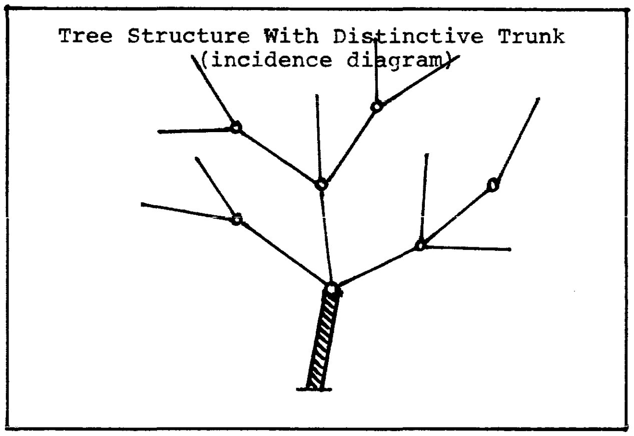


Figure 3

directories of directories of directories of. . . of named files. In corporate organizations we find companies which have departments, which have departments, etc. Figure 5 is a data structure diagram which illustrates this.

This example illustrates a specific need for this type of structure and supports the proposal. Some will argue that this is an incorrect approach to the fundamental corporate organization structure and that they are really hierarchies of different types of organizational-units. At one time, the General Electric Company had a well defined hierarchical organization structure which is illustrated in Figure 6.

If you were a "section" manager, you knew exactly where you were in the management hierarchy. This is an easier structure to handle manually than by computer, as people did not get quite as upset when someone thought it appropriate for a particular "unit" manager to report directly to a "section" or "department" manager. If this organization structure were declared to an I-D-S database system with the structure illustrated in Figure 6, then no "unit" could directly report to a "section" or "department", it would have to be assigned to a "subsection."

The proposal to support alternative owner record-types for a single set-type should be accepted because it is useful. It does not require that the database administrator use either the structure of Figure 5 (alternative owners) or Figure 6 (unique owners). It should be supported because it allows each administrator the choice.

The more general argument, for the support of alternative owner record-types for a set-type, also supports the need to retain the capability for multiple member record-types for a set-type. In an information system, real world entities are represented by records. These entities are classified by entity-type in order to facilitate the processing of data concerning them. Further, each entity-type may portray several concurrent roles or behavior patterns and sometimes these roles are shared by other distinctively different entity-types. For example, a person, or a company, or a governmental unit may serve in the role of "employer" of people, and as an "owner" of property. Within such a designated role, the record-types representing the entity-types should be capable of being the owner or member of a set-type which is role related, and be the holder of a field which is role related. Figure 7 is a data structure diagram which illustrates the employer role played by persons, companies and government units.

To model this structure, it is necessary to declare the "person" record-type, "company" record-type and "gov-

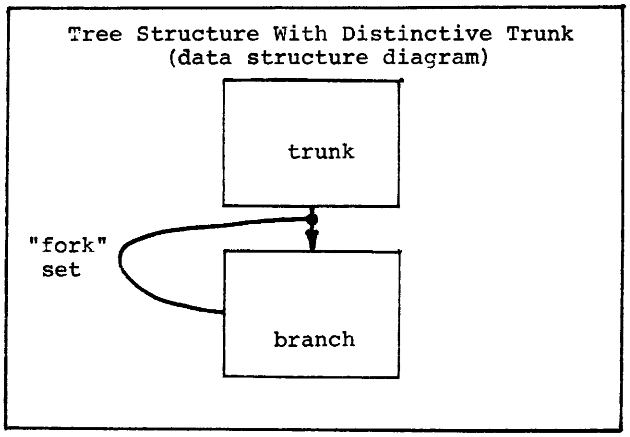


Figure 4

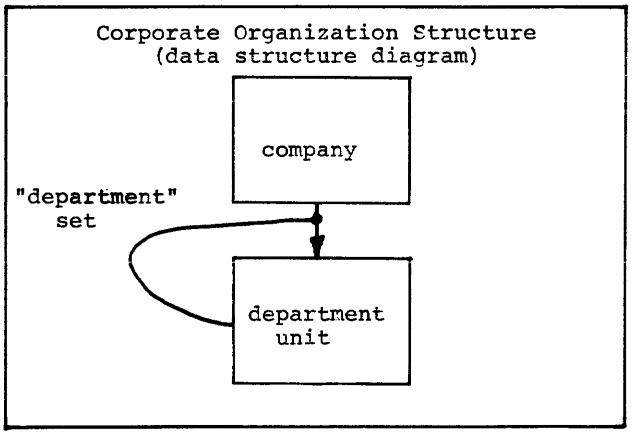


Figure 5

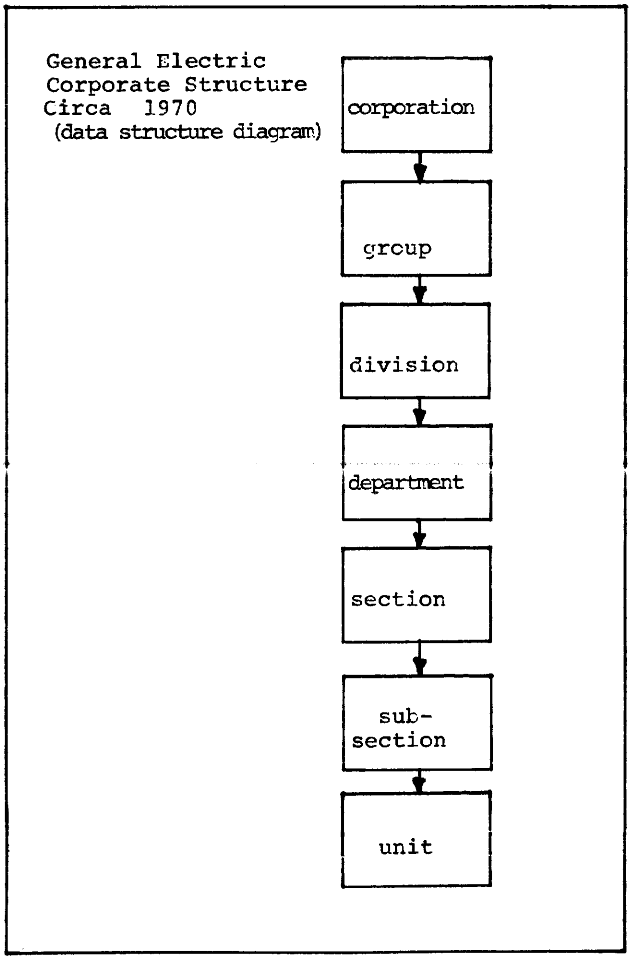


Figure 6

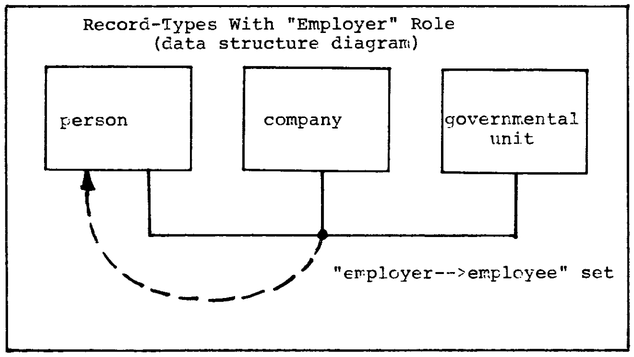


Figure 7

ernmental unit" record-type such that all are able to assume the rule of owner of the "employer——›employee" set. It is necessary to declare the "person" record-type as a sometime set member, "sometime," since all persons are not necessarily employers.

In the case of a person who is self-employed, the same "person" record would be the owner and member of the same occurrence of the "employer——›employer" set. Both the alternative owner (prop. 3) and recursive set (prep. 1) proposals would need to be accepted to support this structure.

The reader should glance back to Figure 6, one of the possible means of implementing the organizational unit aspects of a corporate structure. Given this structure, now imagine how the organization-to-employee relationship would have to be handled. Each organizational record, from the "corporation" record through to the "unit" record, must be able to handle the role of employer. All seven of the organizational records need to be declared as alternative owner types to the "organization——›employee" set. Figure 8 illustrates this extension to Figure 6.

If one assumes that each of the units needs to have a manager, who is a person, then each of these seven organizational unit role record-types must also be declared as a member in the "managed" set. The support of the "managed" set gives an example of the usefulness of the ability to declared multiple record-types as members of the same set-type, (Proposal 2). Figure 9 illustrates the further extension of Figure 6 to include the "manager——›organization" set.

RECORD-ROLE CONCEPT

For the theoreticians (and it is they who have largely argued for reducing the number of member record-types declarable for a set-type to one, and keeping the owner types declarable to one) the introduction of the "record-role" concept may be of great importance. This is because there will be no argument from the practitioners over having only one role declared as the owner of a set-type and only one role declared as the member of set-type if roles become declarable entities. Furthermore, the owner and the member declarations could be restricted to be different "roles."

In the work at Honeywell Information Systems on this subject, the word "record-role" has been used to characterize the role concept introduced above. The following definitions apply:

- A "record-occurrence" is the database representation of a real world entity.
- A "record-role" is a declaration of the a collection of the properties (fields and sets) which a record-occurrence may represent on behalf of one role of a real world entity.
- A "record-type" is a declaration of a collection of one or more record-roles which a record occurrence may represent, while roles are all played concurrently by a real world entity.
- A "record-class" is the collection of all record-occurrence of a particular record-type. A record-occurrence is always in only one record-class, defined by a record-type.
- A "role-class" is the collection of all record-occurrences of a particular record-role. A record-occurrence

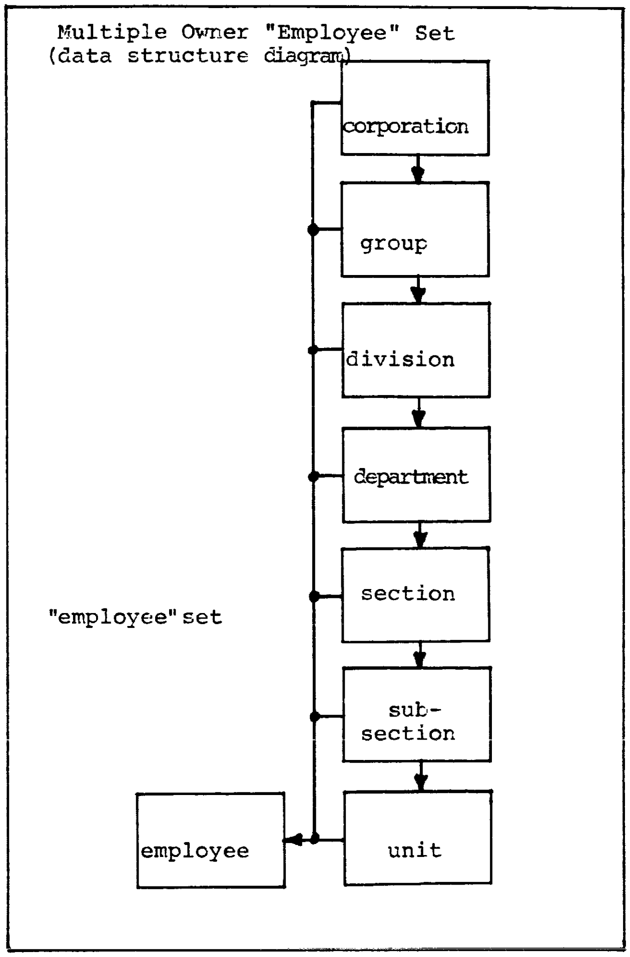


Figure 8

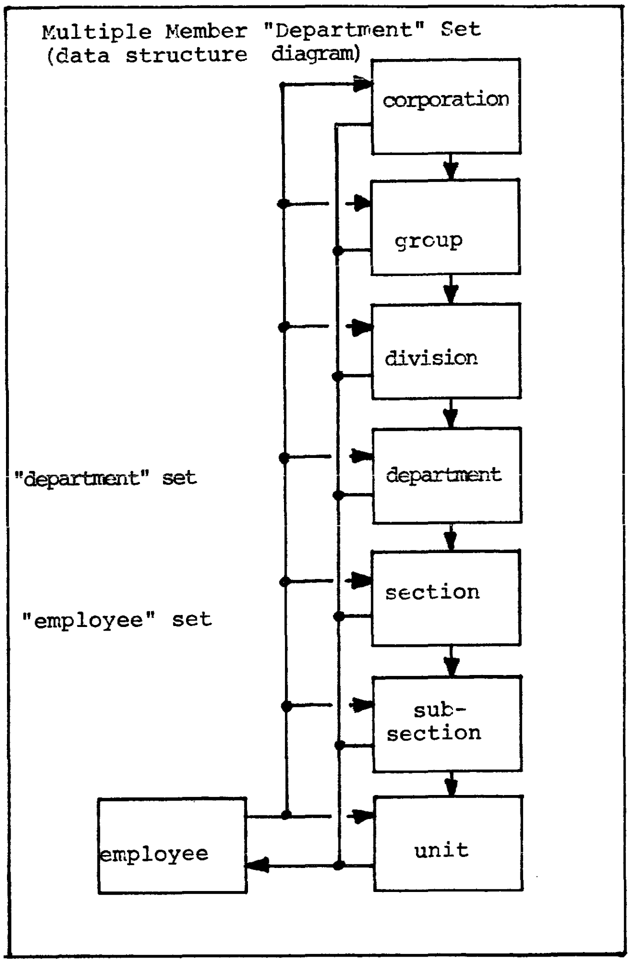


Figure 9

is in one or more many role-classes depending upon its record-type declaration.

- A "role-occurrence" is a subdivision of a record-occurrence which is the representation of that record-occurrence playing one role.

This distinction between record-role and record-type has not been made in existing database systems. While a record-type may have represented several record-roles, there was no mechanism of sharing the record-role declarations between two or more record-types. Thus the same fields and sets, relative to the role, had to be multiply declared, once for each record-type which played the role. This led to the requirement for multiple member declarations and alternate owner declarations for sets.

With the record-role concept, the declaration of fields, groups and sets are all associated with the record-role declaration. Field may be accessed using field-names which are qualified by record-role-name rather than record-type-names. Sets are ordered by role declared fields. Set owner selection is based upon role declared fields. Record occurrences of different record-types coexist within the same set type as owners or members when the record-types share the same record-role with the declared function.

In the paper "The Evolution of Data Structures,"[7] there was a sequence of data structure diagrams which were used to illustrate the progressive introduction of new meta objects and new inter-object relationships into data structuring capabilities. The first two of the following three data structure diagrams are reprinted from that article. Figure 10 illustrates the meta entities: record, field, group, owner, member and data-structure-set, where there may be an unlimited number of member record-types and owner record-types declarable for a set-type. The diagram of Figure 10 is the meta object structure which I recommend to DBTS and have recommended and used for a number of years.

Figure 11 is a simplification of Figure 10 where the restrictions of a one owner entity-role and a one member entity-role have been placed on the set-type. The set-owner and set-member meta entities have been merged with the set-type meta entities, as they exist on a 1 : 1 : 1 basis. This yields the restricted structural capability which was recommended by some of the attendees at the Namur Conference.

The data structure diagram of Figure 12 introduces the meta entity "record-role". In this structure, the "record-role" meta entity has displaced the "record-type" meta entity in its direct relationship to the set, group and field. The record-type concept is now at the side, associated by a declared relationship with one or more "record-roles".

"Record-role" declaration may be associated with one or more record-types. From the viewpoint of the set-type, there is only one "owner" record-role and only one "member" record-role. This fits more easily into the viewpoint of both the relational model and the Data Independent Access

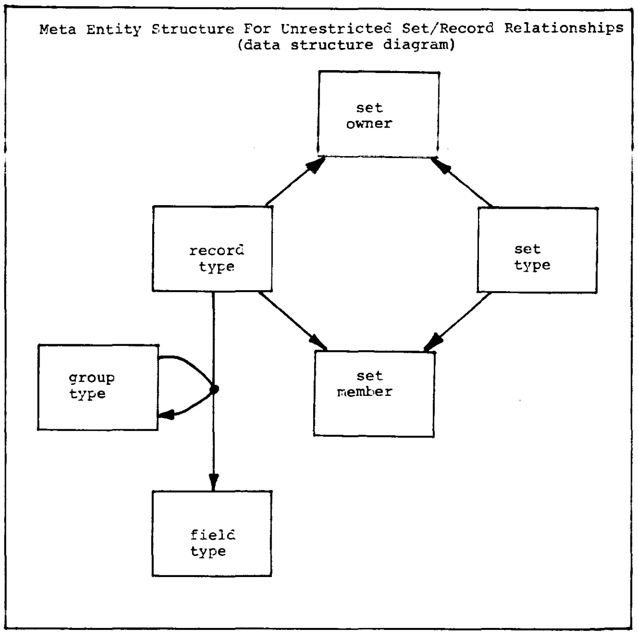


Figure 10

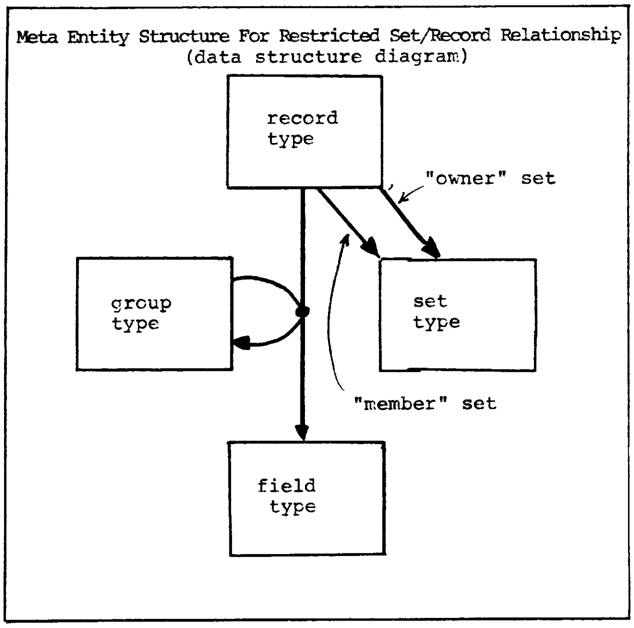


Figure 11

Model. However, the requirements of the real world which we wish to model can be satisfied as each real world entity can be recognized acting in one or more roles and its record-occurrences are combined with the declared role occurrences.

At this time I have no interest in trying to introduce the "record-role" concept into existing data description languages and data manipulation languages. Rather, I wish to provide the rationale, within these languages, for:

(1) recursive set-types,
(2) multiple member set-types, and
(3) multiple owner set-types

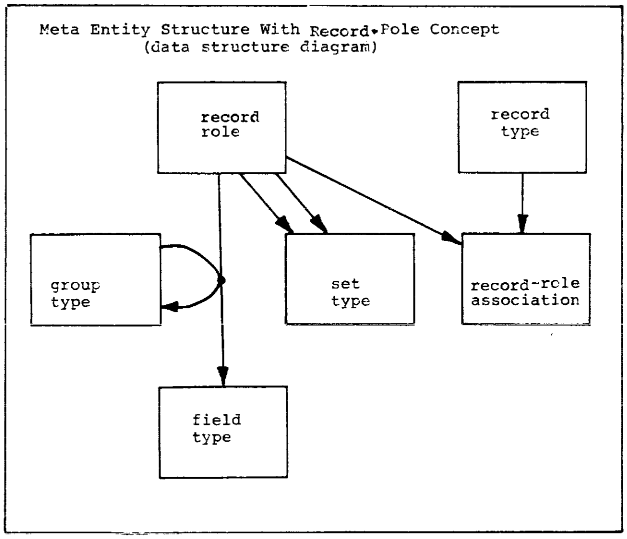


Figure 12

which provide an alternate means for achieving the objectives achieved by the record-role concept while remaining within the limitations of the presently available meta entities. However, the eventual introduction of the "record-role" may be the unifying factor that we seek.

The data structure diagramming technique has been extended to support the concept of record-role. Figure 13 is a redrawing of Figure 7 with focus on the record-roles of employer and employee.

They are illustrated by the two hexagons so designated. The record-types with which the roles are associated are designated by the background boxes. The employer role is played by the company, person and governmental-unit record-types. If classical data structure diagrams were thought to represent record-types and the relationships between them, then the new diagrams illustrate record-roles, their associations with record-types and their relationships with other record-roles. Record-roles are illustrated by hexagons and the background boxes name the record-types which play the role. Each record-type is considered to have one or more roles. In this example "person," "company" and "government-unit" are the record-types. "Employer" and "employee" are record-roles. A complete data structure diagram would show each record-type once as a box on top of a stack of hexagons. Each hexagon representing a record-role played by the record-type. It would also show each record-role, once as a hexagon at the top of a stack of boxes. Each record-type would appear once more as a background box behind each record-role it plays. Figure 13 is thus an incomplete data structure diagram as it shows the record-types playing each record role but does not graphically illustrate each record-

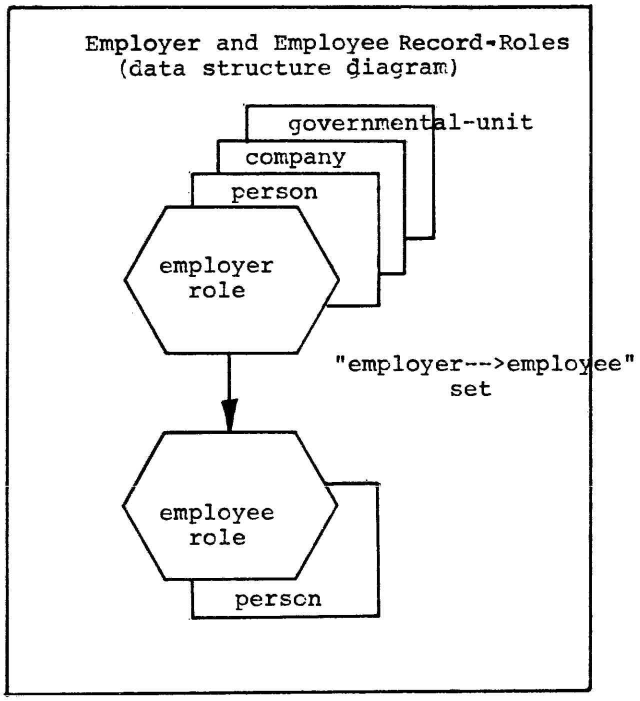


Figure 13

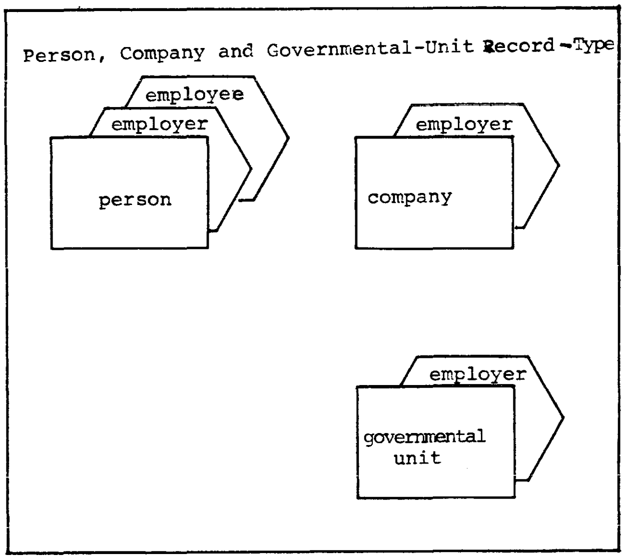


Figure 14

type with its record-roles. Figure 14 illustrates each record-type with the record-role that it plays. Thus most old data structure diagrams can be considered as being examples where the record had only one record-role. Thus no role factoring is necessary. If alternative owners or multiple members exist in those diagrams, then the record-roles for them has not yet been factored. The importance of the record-role concept to data structure diagrams may not be immediately obvious at the first comparison of Figures 7 and 13. However, consider the following analogy. If identical programming code appears in several parts of a computer program, it is common to factor this code out as subroutines or at least as macro procedures so that the documentation is more easily understood. The record-role concept is the data structure diagram equivalent of a subroutine call. The diagram illustrates the shared aspects of the record-role and also all the places it has been invoked. For data structure diagrams, representing complex organizations with many record types, record roles and their relationships, the record-role has proven to be extremely useful in simplifying the diagrams. They are much more readable. Of necessity, these diagrams are only effective after the new concept has been understood, used awhile and accepted.

SUMMARY

The data structure set is almost the only structural tool currently available to the database administrator to represent the relationships between entities in his enterprise. At this time when all of its usages are unknown, it seems desirable not to place any restrictions upon its application. Proposal 1, to permit recursive sets (where owner and member are of the same record class), is a proposal to remove a restriction. Proposal 2, to prohibit multiple member record-types, is a proposal to add a restriction. Proposal 3, to permit alternative owner declarations, is a proposal to remove a restriction. These facilities, within today's record-network model would provide a workable implementation of the role concept, an expression of the evident and important multiple behavior patterns which are characteristic of real world entities.

REFERENCES

1. Bachman, C. W. and S. B. Williams, "A General Purpose Programming System for Random Access Memories," *Proceeding AFIPS Conference Proceeding*, FJCC Volume 26 AFIPS Press Montvale N. J., 1964 pages 411–422.
2. "Integrated Data Store," *DPMA Quarterly*, January 1965.
3. "Software for Random Access Processing," *Datamation* April 1965, pages 36–41.
4. *CODASYL Data Description Language Journal of Development*, June 1973, (c13.6/22:113) Superintendent of Documents, U. S. Government Printing Office, Washington D. C. 20402.
5. *CODASYL COBOL Journal of Development*, January 1976, (110-GP-1D) Material Data Management Branch, Dept. of Supply and Service, 5th Floor, 88 Metcalfe Street, Ottawa, Ontario, Canada, K1A OS5.
6. "Data Structure Diagrams, Data Base 1, 2," 1969, *Quarterly Newsletter of ACM SIGBD*, pages 4–10.
7. "The Evolution of Data Structures." *Proc NordDATA Conference*, August 1973, Copenhagen, Denmark, pages 1075–1093.

The entity-relationship model—A basis for the enterprise view of data

by PETER PIN-SHAN CHEN
Massachusetts Institute of Technology
Cambridge, Massachusetts

ABSTRACT

The concept of the enterprise view of data is very useful in the database design process and in the construction of conceptual schema. This paper discusses the use of the entity-relationship approach in describing and maintaining the enterprise view of data. Fundamental operations for changing the enterprise schema are presented. Finally, an example is given to show the differences between the entity-relationship approach and the data-structure approach in modeling the enterprise view of data.

INTRODUCTION

The subject of the logical view of data has attracted considerable attention in the past ten years. However, most researchers have focused on the user view of data. The need for studying the enterprise view of data was not recognized until recently. Different users of a database may have different views of the database, but the enterprise should have a unique and consistent view of the database. This is particularly important in designing a logically meaningful and consistent database. The concept of the enterprise view of data is very useful in the database design process and in the design of conceptual schema.

Enterprise view and database design

Database design is a process to organize data into a form which matches the underlying data model of the database management system. There are three major types of database management systems: network, hierarchical, and relational. In the network database management systems, which include Honeywell's IDS and UNIVAC' DMS-1100, data will be organized into different types of records and can be represented by a data-structure diagram[1] (see Figure 1). In the hierarchical database management systems, which include IBM's IMS, data will be organized into a form similar to but more restricted than the data-structure diagram. In the relational database management systems,[2] data will be organized into a set of tables (or "relations"). In general, to design a database is to decide how to organize data into specific forms (record types, tables) and how to access them. Up to now, there are very few tools available to aid the database design process. Usually, the database designer relies on his own intuition and experience. Thus, the resulting database may not satisfy company's objectives and may cause problems in company's operations.

Another related problem in database design is that the output of the database design process—the user schema (a description of the user view of data)—is not a "pure" representation of the real world. One of the reasons is that the database designer is restricted by the limited capabilities of the database management system. For example, the many-to-many relationships between entities are difficult to represent directly in some database systems. Another reason is that the user schema may contain some features related to the storage representation of the database. For instance, it may describe which record types can be directly accessed and how to access other record types. In addition, the user schema is usually designed to be efficient for a certain type of data processing operations. For example, the data about employees may be grouped into two record types, employee-master and employee-detail, to improve the retrieval performance. Therefore, the user schema is usually not a direct representation of the real world. This makes the user schema difficult to understand and difficult to change.

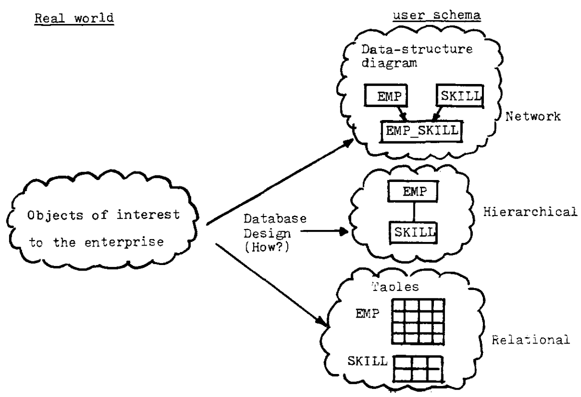


Figure 1—Conventional database design process

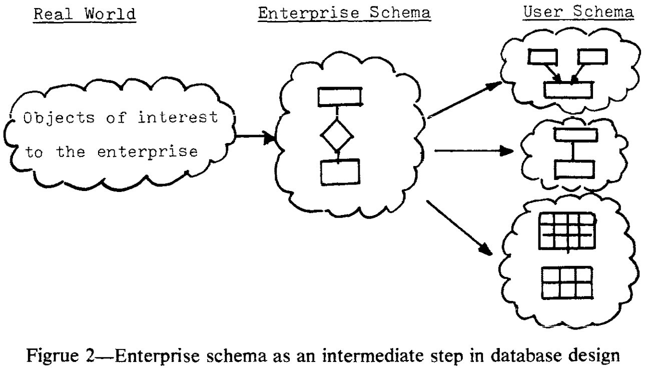
A possible solution to the above problems is to introduce an intermediate stage in the database design process: defining the enterprise schema, which is a "pure" representation of the real world and is independent of storage and efficiency considerations. The enterprise schema will then be translated into different types of schemata for different database management systems (see Figure 2). It can also be translated into several schemata for the same database management system to optimize different types of data processing operations. There are several advantages of this approach:

(1) The enterprise schema is easier to understand than a user schema since the former does not have the restrictions of the underlying database management system;

(2) The enterprise schema is more stable than the user schema, since some types of changes in the user schema may not require any change in the enterprise schema. If the enterprise schema needs to be changed to reflect the changes in the enterprise environment, the changes can be performed easily since efficiency and storage issues are not considered.

Enterprise view and conceptual schema

What is the difference between the enterprise schema and the conceptual schema proposed by the ANSI/X3/SPARC group?[3] Basically, they are very similar since both are descriptions of the enterprise view of data. In the SPARC's approach, the conceptual schema serves as the interface between the external schema (user view of data) and the storage schema (physical view of data) (see Figure 3). The requirement of serving as an interface between two other schemata may introduce some undesirable features into the conceptual schema. If this restriction on the conceptual schema is ignored, there is almost no difference between the conceptual schema and the enterprise schema. Therefore, the techniques discussed in this paper are also suitable for describing and maintaining the conceptual schema in the SPARC's architecture.



Figrue 2—Enterprise schema as an intermediate step in database design

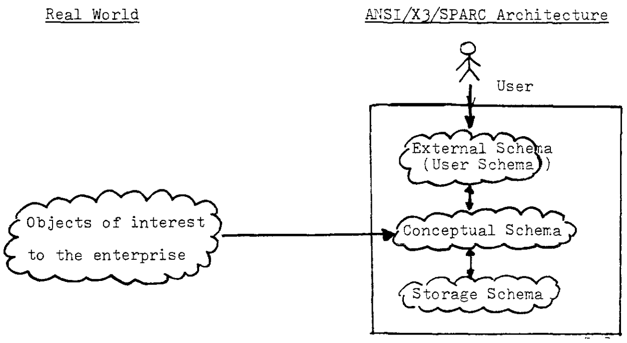


Figure 3—Enterprise view and conceptual schema

Approach used in the paper

In order to describe the enterprise view of data, a mental framework to model the real world is needed. Different people may be used to different mental frameworks. The mental framework used in this paper is the Entity-Relationship (E-R) model.[4,5] The E-R model and similar approaches[6–9] have been found useful in modeling the real world. A diagrammatic technique called the Entity-Relationship (E-R) diagram will be used in this paper to represent the enterprise view of data.

This paper is divided into three parts. The first part discusses how to use the E-R model and diagrammatic technique to describe the enterprise view of data. This is an extension of the work reported in Reference 5. The second part describes fundamental operations for changing the enterprise view of data. This is an area where very little work has been done. The operations proposed in this paper will be useful in maintaining the enterprise schema. The third part uses the E-R approach to analyze an example given by Bachman[10] concerning changes in the conceptual schema.

MODELING THE REAL WORLD USING THE ENTITY-RELATIONSHIP MODEL AND DIAGRAMMATIC TECHNIQUE

In this section, we shall use examples to show how to use the Entity-Relationship (E-R) model and diagrammatic technique to describe the enterprise view of data. A more formal definition of the model can be found in Reference 5.

It is assumed that the responsibility of defining and maintaining the enterprise schema belongs to a person called the *enterprise administrator*. The following is the suggested procedure for the enterprise administrator to define the enterprise schema:

(1) *identify entity sets of interest to the enterprise*

An *entity* is a "thing" which can be distinctly identified. According to the needs of the enterprise, entities can be classified into different *entity types* such as EMPLOYEE, STOCK_HOLDER. An *entity set* is a group of entities of the same type. In the E-R

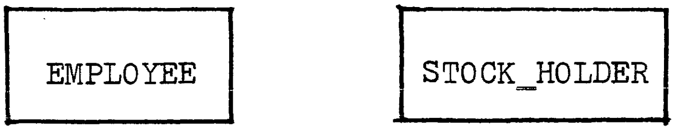


Figure 4—Entity sets

diagram, an entity set is represented by a rectangular-shaped box (see Figure 4). The terms, "set" and "type," can be interchanged in the E-R diagram. The reader may use either one to interpret the E-R diagram.

There are many "things" in the real world. In addition, different enterprises may view the same thing differently. It is the responsibility of the enterprise administrator to select the entity types which are most suitable for his company.

(2) *identify the relationship sets of interest to the enterprise*

Entities are related to each other. Different *types* of relationships may exist between different types of entities. A *relationship set* is a set of relationships of the same type. For example, PROJ_EMP, which describes the assignment of employees to projects, is a relationship set defined on two entity sets, EMP and PROJ. A relationship set can also be defined on more than two entity sets. For example, PROJ_SUPP_PART is a relationship set defined on three entity sets PROJ, SUPP, and PART. In the entity-relationship diagram, a relationship set is represented by a diamond-shaped box with lines connecting to the related entity sets (see Figure 5). The "m" and "n" associated with the PROJ_EMP relationship in the E-R diagram indicate that the relationship is an m:n mapping. That is, each employee may be associated with several projects, and each project may have several employees. In certain companies, each employee belongs to at most one project, and the PROJ_EMP relationship is a 1:n mapping.

There are many types of relationships between entities. The responsibility of the enterprise administrator is to select the relationship sets (or types) which are of interest to the enterprise. He also has to specify the type of mappings (1:1, 1:n, m:1, or m:n) of the relationships.

(3) *identify relevant properties of entities and relationships (i.e., define value sets and attributes)*

Entities and relationships have properties, which can be expressed in terms of *Attribute-value* pairs. "Blue," and "4" are examples of *values*. Values can be classified into different *types* such as COLOR or QUANTITY. A *value set* is a group of values of the same type. An *attribute* is a mapping from an entity set (or a relationship set) to a value set (or a group of value sets). For example, "address" is an attribute which maps entities in the entity set EMP to values in the value set NAMES_OF_LOC. Note that we relax the constraint imposed in Reference 5 that the mapping from the entity set to the value set has to be a function (i.e., m:1 mapping). In other words, we now allow that an attribute (such as address) can have several values (such as locations) for the same entity (employee). This relaxation in the definition of attribute will make the changes in the enterprise view simpler. This point will become clear in the next section.

In the E-R diagram, a value set is represented by a circle, and an attribute is represented by an arrow directed from the entity set (or the relationship set) to the desired value set(s) (see Figure 6). After selecting entity sets and relationship sets, the enterprise administrator identifies the attributes and value sets which are relevant to the company's operations.

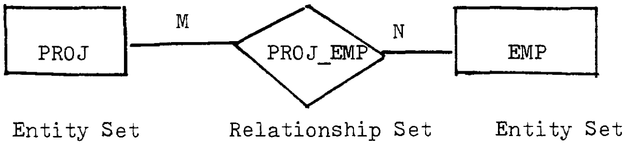


Figure 5—Relationship set

The three steps stated above cover a major part of the enterprise schema. For simplicity, we shall not discuss in this paper other issues related to the enterprise schema such as integrity constraints.

To design a database, the enterprise administrator first draws an E-R diagram such as the one shown in Figure 7. He then drew the attributes and value sets for each entity set and relationship set. The E-R diagram is then translated into a data-structure diagram or a set of tables ("relations") (see Figure 2). The rules and procedures used in the translation process were discussed in Reference 5. Here, we shall investigate how to change the enterprise schema (the E-R diagram) itself.

MODIFICATION OF THE ENTERPRISE VIEW

Although the enterprise schema is more stable than a user schema, it still needs to be changed from time to time

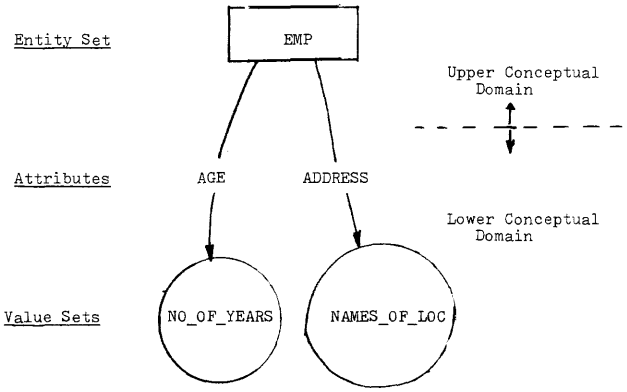


Figure 6—Attributes and value sets

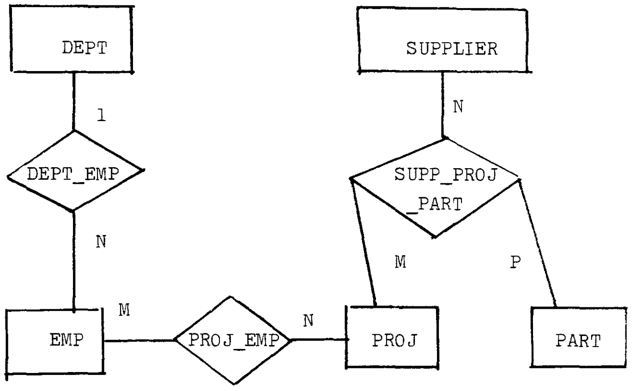


Figure 7—An entity-relationship diagram (with entity sets and relationship sets only)

to reflect the changes in the enterprise environment. Excepting a paper by Bachman,[10] very little work has been done in this area. In this paper, we use the E-R model as a basis for analyzing different types of changes in the enterprise view of data. We not only propose a set of operations but also analyze the consequences of these operations.

There are five basic types of operations: *add, delete, split, merge,* and *shift*. The first four operations are applicable to entity sets, relationship sets, attributes, and value sets. The *shift* operation is used when the enterprise administrator would like to view a value set in the old enterprise schema as an entity set in the new schema or vice versa. It is useful to think that the E-R diagram consists of two conceptual domains: (1) the upper conceptual domain which consists of entity sets and relationship sets; (2) the lower conceptual domain which consists of attributes and value sets. We shall discuss the first four operations in both the upper and lower conceptual domains. Finally, we shall discuss the shifting an entity set from the upper conceptual domain to the lower conceptual domain and the shifting a value set in the opposite direction.

Operations in the upper conceptual domain

The following are the basic operations applicable to entity sets and relationship sets:

(1) *Split an entity into several subsets*
For instance, the entity set EMP in Figure 8a can be split into two entity sets: MALE_EMP AND FEMALE_EMP in Figure 8b. The consequence of this operation is that the relationship sets associated with the entity set may also have to be split. For example, PROJ_EMP is split into PROJ_M_EMP and PROJ_F_EMP (see Figure 8b).

(2) *Merge several entity sets into one entity set*
This is the opposite operation of (1). The consequence is that the related relationship sets may have to be merged.

(3) *Split a relationship set into several subsets*
An example of this operation is: the relationship set PROJ_EMP in Figure 8a can be split into two relationship sets, PROJ_MANAGER and PROJ_WORKER, in Figure 8c. Note that the type of mapping in the new relationships may be different from that in the original relationship. For instance, the mapping in PROJ_MANAGER is 1:n while the mapping in PROJ_EMP is m:n.

(4) *Merge several relationship sets into one set*
This is the opposite operation of (3). Note that these relationship sets have to be defined on the same group of entity sets.

(5) *Add a new entity set*
For example, a new entity set called SUPPLIER may be added to the E-R diagram in Figure 8a. The result is shown in Figure 9a. Note that it is possible to have stand-alone entity sets in the enterprise schema, although in many cases relationships between the new entity set and the existing entity sets are established immediately (see the next operation).

(6) *Add a new relationship set*
We may add a new relationship set for the new entity set such as the relationship set PROJ_SUPP in Figure 9b. We may also add a new relationship set for existing entity sets such as the relationship set PROJ_MANAGER in Figure 9b.

(7) *Delete an entity set*
For instance, after deleting the entity set EMP in Figure 9b, we have Figure 9c. The consequences are: (i) the relationship sets related to the entity set are

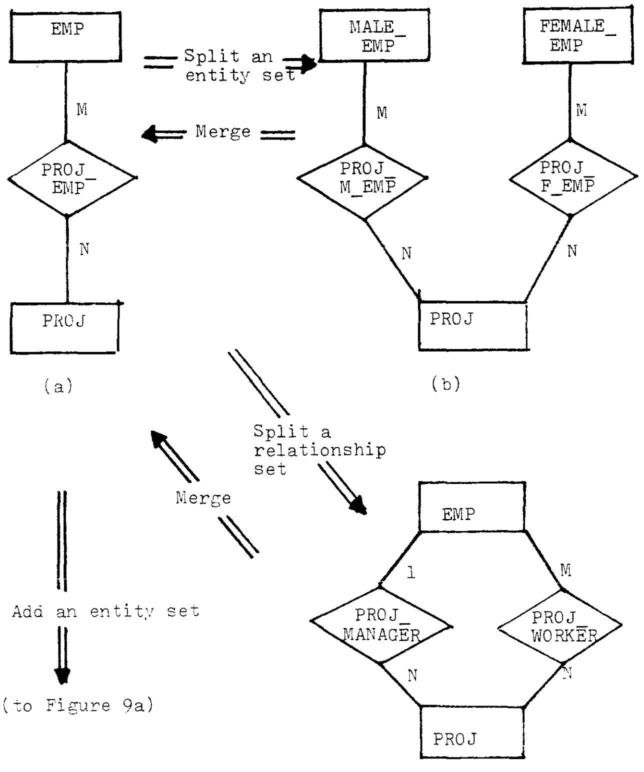


Figure 8—$\left\{\begin{matrix}\text{Split}\\ \text{Merge}\end{matrix}\right\}$ $\left\{\begin{matrix}\text{Entity}\\ \text{Relationships}\end{matrix}\right\}$ Sets

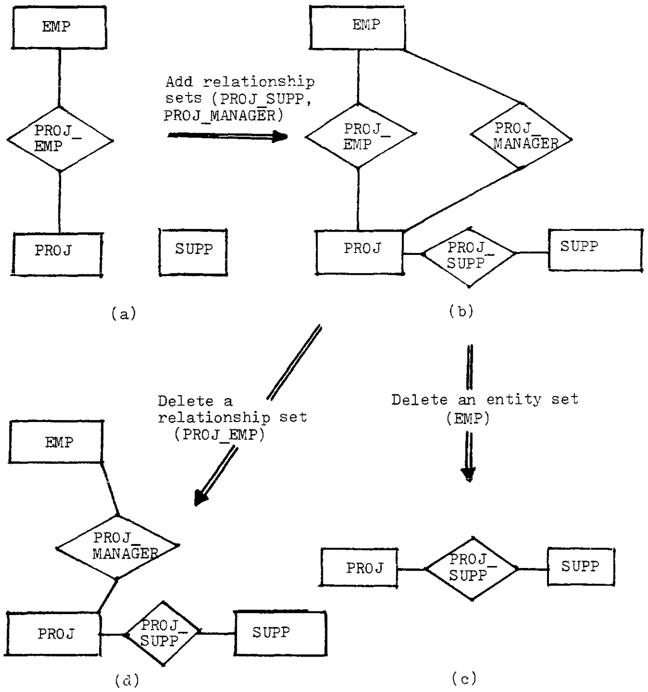


Figure 9—$\begin{Bmatrix}\text{Add}\\ \text{Delete}\end{Bmatrix} \begin{Bmatrix}\text{Entity}\\ \text{Relationship}\end{Bmatrix}$ Sets

also deleted; (ii) attributes related to the deleted entity set and related relationship sets are also deleted.

(8) *Delete a relationship set*
An example is: delete the relationship set PROJ_EMP in Figure 9b, and the result is shown in Figure 9d. The consequence of this operation is that the attributes of the relationships are deleted (not shown in Figure 9d).

Operations in the lower conceptual domain

Assume that the entities in the entity set EMP have two attributes, LEGAL_NAME and PHONE, which map the entities to the value sets NAME and PHONE_# (see Figure 10a). We shall use these attributes and value sets as the basis for the discussion of the following operations:

(1) *Add a value set*
For example, a new value set called DOLLARS may be added to Figure 10a. The result is shown in Figure 10b. Usually, this operation is followed by an "add attribute" operation.

(2) *Delete a value set*
After deleting the value set PHONE_# in Figure 10a, we get Figure 10c. The consequence is that all attributes associated with this value set will be deleted.

(3) *Split a value set into several subsets*
The value set NAMES in Figure 10a may be split into two value sets FIRST_NAMES and LAST_NAMES in Figure 10d. The consequence is that attributes related to the value set may have to be adjusted. Although the attribute LEGAL_NAME is not split in Figure 10d, it is possible to split it into two attributes: LEGAL_FIRST_NAME and LEGAL_LAST_NAME. It is the responsibility of the enterprise administrator to make this decision.

(4) *Merge several value sets into a value set*
This is the opposite operation of (3).

(5) *Add an attribute*
For instance, Figure 11b is obtained by adding the attribute OTHER_NAME to Figure 11a.

(6) *Delete an attribute*
Deleting the attribute LEGAL_NAME from Figure 11a, we have Figure 11c. The value set associated with the attribute will be deleted by another operation ("delete value set") if desired. In some cases, the value set may be still associated with other attributes (see Figure 11c).

(7) *Split an attribute into several attributes*
For example, Figure 11d is obtained by splitting the attribute PHONE in Figure 11a into two attributes, OFFICE_PHONE and HOME_PHONE.

(8) *Merge several attributes into one attribute*
This is the opposite operation of (7). The attributes have to be defined on the same entity set (or relationship set).

Operations between two conceptual domains

Assume that there are two entity sets (EMP and PROJ), one relationship set (PROJ_EMP), four value sets (NAMES_OF_PLACES, SOC_SEC_#, PHONE_#, and

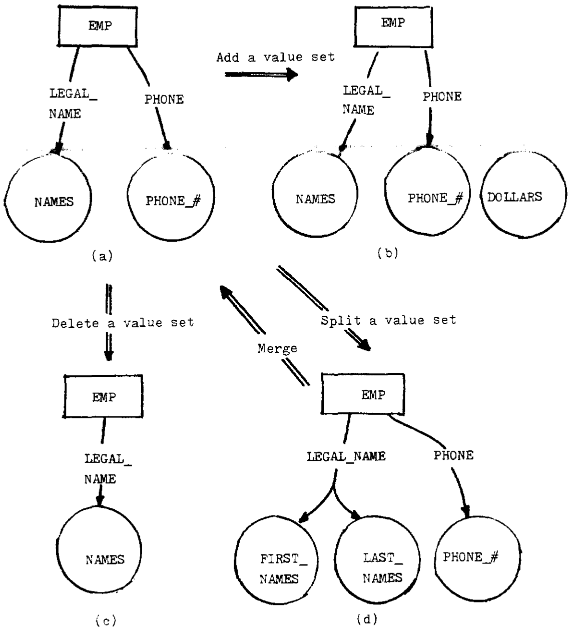


Figure 10—Operations on value sets

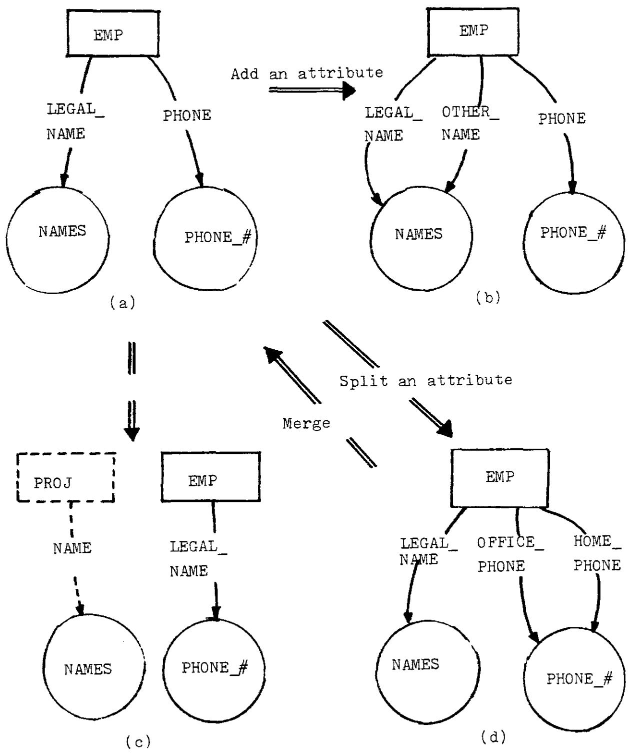


Figure 11—Operations on attributes

PROJ_NAMES), and four attributes (ADDRESS, SOC_SEC_NO, PHONE, and NAME) as shown in Figure 12a. We shall use them as the basis for the discussion on the following operations:

(1) *Shift a value set from the lower conceptual domain to the upper conceptual domain*
When the enterprise environment changes, it may become natural to view PLACE as an entity set instead of a value set. Thus, in Figure 12b "ADDRESS" becomes a relationship set, and "PLACE" has an attribute "NAME" which points to the value set NAMES_OF_PLACES. Since PLACE is an entity set, we may establish new relationships of it with other entity sets such as PROJ or add more attributes and value sets to describe properties of "places."

(2) *Shift an entity set from the upper conceptual domain to the lower conceptual domain*
When the enterprise environment changes again, it may become natural to view PROJ as a value set instead of an entity set. In Figure 12c, PROJ is deleted from the upper conceptual domain, and the relationship set PROJ_EMP becomes the attribute INVOLVED_PROJ. The entity set PROJ in Figure 12b may have been associated with several value sets, but only the value set PROJ_NAMES which is used to identify the entities PROJ remains in the lower conceptual domain.

ANALYSIS OF AN EXAMPLE

In a recent paper, Bachman[10] uses data-structure diagrams to illustrate the changes in a conceptual schema. In this section, we shall first state his example and then use E-R diagrams to interpret his example.

Description of the example using data-structure diagrams

The following is a simplified version of Bachman's example:

(a) In the beginning, the enterprise administrator declared a conceptual schema as shown in Figure 13a. The reader is assumed to have some knowledge of the data-structure diagram.[1] Simply speaking, a rectangular-shaped box represents a record type, and an arrow represents a data-structure-set (i.e., 1:n relationship between record types). In Figure 13a, there are two types of conceptual records, COMPANY and PERSON, and a data-structure-set "a" representing the fact that each person is associated with exactly one company and that each company has a set of personnel.

(b) Later, the enterprise administrator recognized that the personnel of the company were persons in their own right. This fact may be discovered at the merger

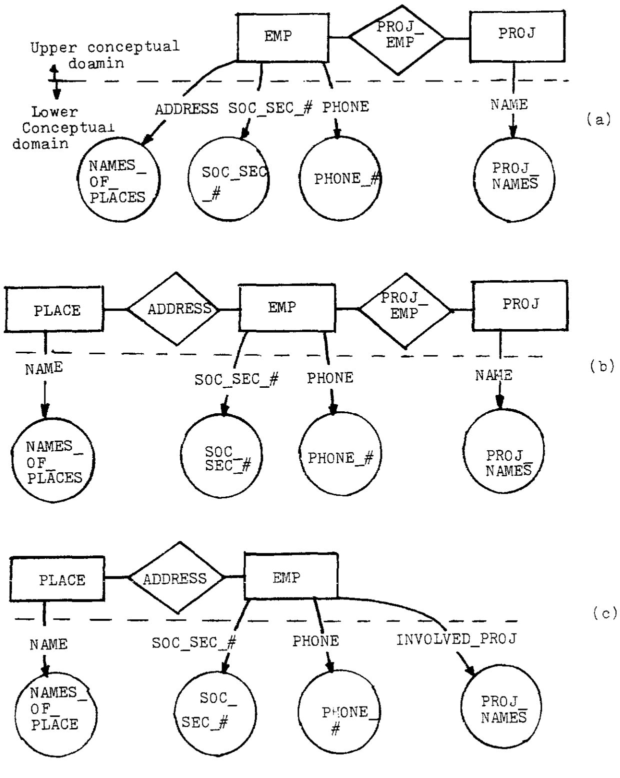


Figure 12—Shifting a set from the upper conceptual domain to the lower conceptual domain and vice versa

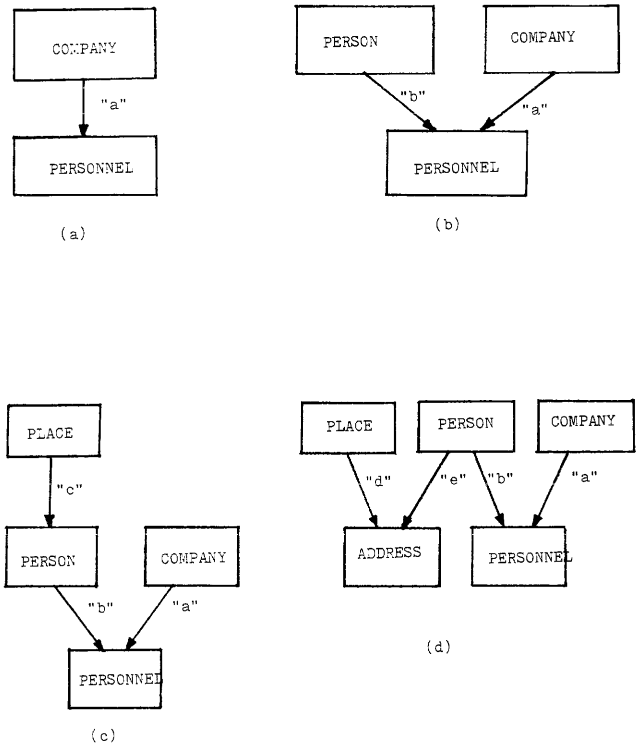


Figure 13—Expressing changes in the enterprise view using data-structure diagrams

of several companies that some of the persons held two jobs and were personnel to two of the merged companies. Figure 13b illustrates the data-structure diagram for the new conceptual schema. Basically, the old personnel type record has been split into two record types, PERSONNEL and PERSON. The "PERSON" has attributes NAME and ADDRESS (not shown in the figure).

(c) After a while, the enterprise administrator decided to factor the address of residence out of the person record. Figure 13c illustrates the addition of the "PLACE" conceptual record type and the data-structure-set type "c." It was also assumed that each person has a unique address (place).

(d) It is now recognized that people move from place to place and that it is desirable to know current address as well as past addresses. Another reason may be: it is discovered that a person may have more than one address. In either case, a new conceptual record type ADDRESS is added to the conceptual schema (see Figure 13d).

Analysis using entity-relationship diagrams

In the following, we shall use E-R diagrams to explain the above example:

(a) The E-R diagram in Figure 14a is corresponding to the data-structure diagram in Figure 13a. There are two types of entities, PERSON and COMPANY, in the enterprise view. Since the mapping between COMPANY and PERSON is 1:n, the relationship set PERSONNEL is represented by a data-structure-set "a" in Figure 13a.

(b) Figure 14b is the corresponding E-R diagram for Figure 13b. Since the relationship set PERSONNEL is an m:n mapping, it is represented by a relationship record type PERSONNEL and two data-structure-sets "a" and "b" in Figure 13b. Note that Figures 14a and 14b have the same entity sets and relationship set in the upper conceptual domain, and the difference is the type of mapping between the entity sets.

(c) Now the enterprise administrator prefers to view "PLACE" as an entity set rather than a value set. Thus, we have Figure 14c. The attribute ADDRESS in Figure 14b becomes a relationship set in Figure 14c. Since the mapping between PLACE and PERSON is 1:n, the relationship set ADDRESS is represented by the data-structure-set "c" in Figure 13c.

(d) The enterprise administrator discovers that the mapping between PLACE and PERSON is an m:n mapping instead of a 1:n mapping. The new enterprise view is represented by Figure 14d. Since the mapping is m:n, the relationship set ADDRESS is represented by the record type ADDRESS and two data-structure-sets "d" and "e." Note that Figures 14c and 14d

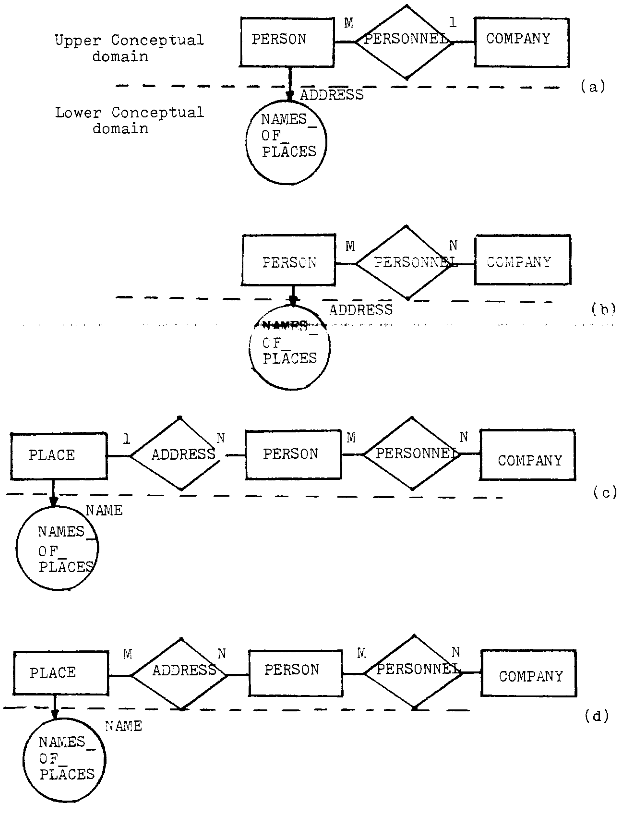


Figure 14—Analysis of Figure 13 using E-R diagrams

are almost the same except that the type of mapping between PLACE and PERSON is different.

In general, the E-R diagram is easier to use to analyze the changes in the enterprise view than the data-structure diagram. Bachman also raised the issue of the ambiguity in Figure 13d: If one wants to modify a person's address, does he have to create a new "address" record or to change the name of the place where the person is living? This question can be easily answered using the E-R approach. Consider Figure 14d. Since the PLACE is an entity set, to change a person's address is to change the relationship between the person and "his place." We should not change the name of the place where the person is living since "NAME" and "NAMES_OF_PLACES" are used to describe a property of the PLACE entities (see Figure 14d).

SUMMARY

The enterprise schema is useful as an intermediate step in database design. In this paper, we have shown how to use the entity-relationship model and diagrammatic technique to describe the enterprise schema. Since the enterprise environment changes from time to time, the enterprise schema will have to change to reflect these changes. Five basic types of operations (add, delete, split, merge, and shift) which are useful in modifying the enterprise schema have been presented, and the consequences of these operations have been discussed. Finally, we have used an example to analyze the differences between the entity-relationship approach and the network approach in modeling the enterprise view of data.

REFERENCES

1. Bachman, C. W., "Data Structure Diagrams," *Data Base 1*, 2, Summer 1969, pp. 4–10.
2. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," *Comm. ACM 13*, 6, June 1970, pp. 377–387.
3. ANSI, *Interim Report of ANSI/X3/SPARC Group on Database Management Systems*, ANSI, February 1975.
4. Chen, P. P., "The Entity-Relationship Model," (abstract), *Proc. 1st Very Large Database Conf.*, Framingham, Mass., Sept. 1975, ACM.
5. Chen, P. P., "The Entity-Relationship Model: Toward a Unified View of Data," *ACM Tran. on Database Systems 1*, 1, March 1976, pp. 9-36.
6. Moulin, P., J. Randon, M. Teboul, et al., "Conceptual Model as a Database Design Tool," *Proc. IFIP TC-2 Working Conf.*, Jan. 1976, Black Forest, Germany, pp. 459–479.
7. Hall, P., Todd S. Owlett, "Relations and Entities," *Proc. IFIP TC-2 Working Conf.*, Jan. 1976, Black Forest, Germany, pp. 430–458.
8. Deheneffe C. and H. Hennebert, "NUL: a Navigational User's Language for a Network Structured Data Base," *Proc. ACM 1976 SIGMOD Conf.*, Washington, D.C., June 1976, pp. 135–142.
9. Tozer, E. E., "Database Systems Analysis and Design," Technical report, Software Sciences Limited, England, April 1976.
10. Bachman, C. W., "Trends in Database Management—1975," *Proc. AFIPS 1975 NCC*, Vol. 44, AFIPS Press, Montvale, N. J., pp. 569–576.

Data architecture and data model considerations*

by EDGAR H. SIBLEY and LARRY KERSCHBERG
University of Maryland
College Park, Maryland

ABSTRACT

The Data Base Management System is now a well established part of information systems technology, but the many architectures and their plethora of data models are confusing to both the practitioner and researcher. In the past, attempts have been made to compare and contrast some of these systems, but the greatest difficulty arises in seeking a common basis. This paper attempts to show how a generalized data system (GDS), represented by two different models, could form such a basis; it then proposes that data policy definitions can restrict the GDS to a specialized model, such as a relational or DBTG-like model. Finally, it proposes that this concept forms a better basis for data structure design of specific system applications.

INTRODUCTION

The seventies has seen the acceptance of the database management system (DBMS). Commercial systems and research efforts have proliferated, and the subject has become a major conference topic. However, the potential user is still left with most of the questions that first appeared: Which is the best system? Am I locking myself into one technique or implementation method?

There have been attempts at explaining similarities and differences in the basic classes of systems,[1,2] debate on the effectiveness of different data models,[3] and description of the selection and acquisition process,[4] but confusion remains.

Possibly the reason for difficulty is:

1. The topic is complex. DBMS exist, but they are so different that they defy simple comparison. They also run the gamut of size and sophistication.
2. They differ in methodology of data modelling, retaining, and querying, as well as their internal storage. Testing is expensive: some representative system must be implemented on several DBMS for comparison, or difficult simulations[5] are needed.

Further problems arise in large scale database research and there is need for a common basis[6] as a formalism for describing such systems. Methods of defining the functionality of DBMS include set theory and graph theory constructs.[7] This paper attempts to define such a common basis, and show how it can be used to compare models.

DEVELOPMENT OF A FRAMEWORK

There are at least three distinct levels in an information system: the information and its structure, the data model, and the storage structuring. Obviously, no short paper can cover all three, and we will concentrate on the data model. However, consideration must be given to the information system/data model interface to set the stage for ways to define a good data model with its need to reflect the way data are interrelated, manipulated, and protected.

A *data model* is a system in which a schema may be defined; the DDLC's definition language[8] is principally a mechanism for defining the names and attributes of data elements, groupings, and relationships, while the definition of policy (integrity, security, efficiency, etc.) is almost an afterthought.

The information system—Data model interface

There is an important interface between the organization view of information and the data model constructed to represent it. This interface is being investigated by researchers who are attempting to define a process for producing a good data base design given a set of user needs or aspirations. Reference 9 is a survey of current techniques.

Complete knowledge of the information system and its data usage characterizes the company. Operating policy, however, summarizes the internal constraints of the organization, and the way that the functional subsystems interact; one author[10] refers to these as operative and directive

* The authors wish to express their gratitude to the U.S. Army Computer Systems Command who, through grant number DAAG29-76-G-0300 (Title: Problems in the Translation and Standardization of Relational and Network Type Data Base), provided partial support and technical advice.

information structures. Several researchers[11,12] have recently advocated the collection of transactions as a basis for designing logical data structures.

Our goal, however, is to develop a framework in which to study data models, and to incorporate important parameters of the interface into the data model: Data Utilization and Operating Policy. By incorporating these parameters into the data model framework we expect to examine some classes and:

- Explain subtle differences,
- Explore declarative versus procedural aspects, and
- Characterize their "semantics."

Data architecture—A level concept

A recent paper[13] proposes that there are four abstraction levels for data machines and models. These, in increasing abstraction, are:

1. *The Defined and Populated Database:* a fully operational data system, with database defined via some definition language.
2. *The Database System:* it involves no data, but represents a specific system, with its description.
3. *The Data Model:* the data system prior to its application. The class(es) of data structures that may be supported by the system have been fixed, but not used.
4. *Data Model Theory:* a conceptual or generalized data base management system generator, assumed to be able to support all classes of data models.

These levels form a progression: From one to four is abstraction—from four to one is utilization; each level naturally subsumes or subsets the previous one.

As an example, Level 3 may be a Relational Model;[14] i.e., it can support a relational data system, but no other. Then Level 2 might be the implementation of a payroll data definition; i.e., a definition of those items (and their attributes) with procedures making up a relational payroll system. At Level 1, we see sets of payroll tuples; i.e., entries for specific people.

We shall use this concept as a basis for the paper. However, there are special operations performed in going from one level to another, defined as follows:

- Level 4 to 3: Data Policy Definition.
  In this step, the management and information system designers state the major constraints on the operational system. The resulting data model (or database machine) at Level 3 is restricted: only some classes of data structures are now allowed; some types of operation are restricted, allowing privacy or security (policy) decisions to be stated; some actions are performed automatically, allowing validation and integrity (policy) to be stated.
- Level 3 to 2: Data Operation Definition.
  Here, the administration is working with a restricted system in which data structure, some efficiency, and specific policy considerations may be stated: e.g., Data Policy Definition specified a relational system with validation at input, now Data Operation Definition defines a database of 2-tuples involving social security number and name, where the former is a nine digit element. This also involves definition of the procedures for building, maintaining, manipulating and retrieving data.
- Level 2 to 1: Data Population and Utilization.
  Finally the data must be loaded and used.

The data model generator—generalized data system

Level 4 data architecture or system can be viewed as either a theory or a machine: i.e., as either an abstract description of a method, or as a machine implementation of that method. If the concepts of Level 4 can be expressed in set theory, then the "machine" could be either a theoretical or working set processor. In this paper, we discuss two candidates for Level 4 machines, and then show how each may be restricted to Level 3 machines. It is, however, of tantamount importance that these machines be truly general, and this exercise is an attempt to show the need and generality, as well as to illustrate the parts and use of such machines. Obviously, if the Level 4 machine is sufficiently general, it will *cover* all possible data machines at Level 3 (at least relational, hierarchic, and network models). It may be considered a meta-data model or generalized data system (GDS). Furthermore, the use of such a system provides a framework for data model comparison.

The two models discussed here are:

i. The Functional Model[7] and
ii. The Set Theoretic and Extended Set Processor[15–19] modified to allow data policy and data operation definition.

THE FUNCTIONAL MODEL OF DATA

Here we consider the Functional Model of Data as a GDS. First, Level 4 structure is presented, then data policy constraints are shown to add form and structure to the model. The constraints are semantic, and allow the Functional Model to be viewed in restricted cases as either relational or DBTG (network) data models.

Level 4 structure

Level 4 is a meta-data level where the Functional Model of data is viewed as a directed graph; its nodes represent sets and its arcs represent total functions. Nodes are either entity sets or value sets. *Entity Sets* may have any number of incoming or outgoing arcs; *Value Sets* may have only incoming arcs, because "values" are the ultimate logical representation of information, so no arcs leave value set nodes. A typical Level 4 Functional Model graph is shown in Figure 1.

The definitional facilities for the Level 4 Functional Model consists of three creation and naming operations for value sets, entity sets, and functional specification of an entity set (i.e., specification of functions whose domain is the entity set).

There are also operations for deleting value sets, entity sets, and functions. The deletion operations have the following side-effects:

- Deletion of an entity set also implies deletion of those functions incident on it (both incoming and outgoing);
- Deletion of a value set also implies deletion of those functions incident on it;
- Deletion of a function does not affect its domain and range sets, but some may become isolated nodes and may no longer be relevant.

Data policy definition

Data policy decisions are of the following types:

- The methodology to be used to obtain the data structures.
- The representation of elements in the nodes (i.e., sets) of the Functional Model graph.
- The logical access mechanisms to be supported at Level 3.

While both information and management policy ramifications may be stated in a declarative fashion (CODASYL DDL,[8]), management policies are often enforced by transaction-driven "triggers."[20]

To refine these ideas we first address the ramifications of data policy. The most important decision is the choice of data model methodology; this will determine the richness and complexity of the allowable data structures. The methodology adopted in the Functional Model is semantic predication analysis,[21] a process developed to analyze the semantic structure of sentences.

A predication represents a whole sentence: e.g., an assertion, a command, or a question; it may be decomposed into zero, one, or two arguments and a predicate. Arguments may themselves be predications. "Downgraded predications" may qualify arguments (the semantic equivalent of adjectival clauses) or may modify predicates (the semantic equivalent of adverbial clauses). The lowest semantic level consists of semantic features which serve as atomic semantic description units. Downgraded predications play the role of semantic features of the arguments or predicates that they qualify or modify.

Here we assume that a specification of data interrelationships is available. The information analyst's role is to obtain the corresponding predication structures and map these to the Functional Model (an abstraction process). Consider the statement:

"Companies supply parts to departments in some volume"

The predication structure is shown in Figure 2, where the main predication structure "companies supply parts" is represented by the predication PN$_1$ with arguments A$_1$ (COMPANIES), A$_2$ (PARTS) and predicate P$_1$ (SUPPLY). The arrow under SUPPLY denotes the direction of the relationship represented by PN$_1$; it corresponds to the active voice of P$_1$. PN$_2$ and PN$_3$ are downgraded modifying predications representing the indirect object and adverbial

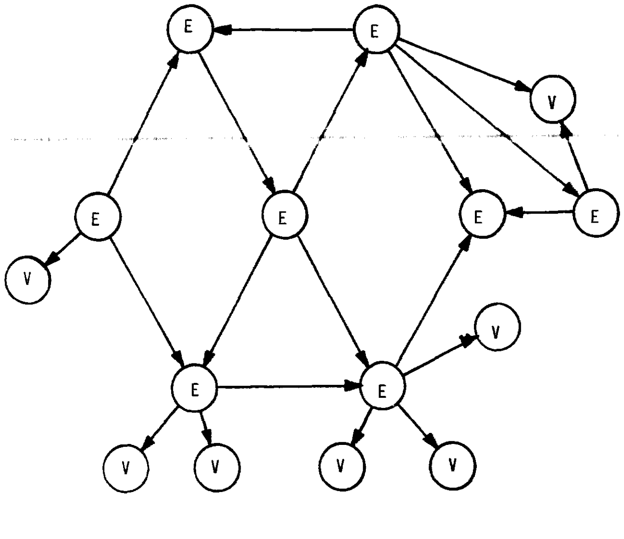


Figure 1—Functional model graph at level 4

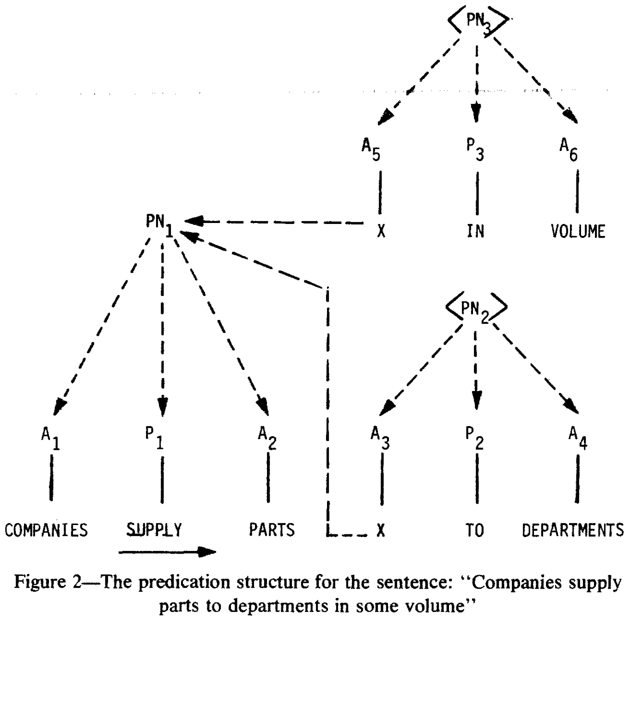


Figure 2—The predication structure for the sentence: "Companies supply parts to departments in some volume"

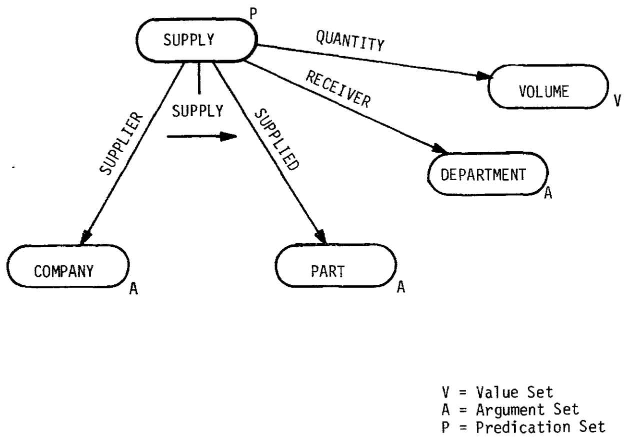


Figure 3—The functional model data structure

information, respectively. The X's refer to P_1, which their corresponding predications modify. Semantic features are not present in this example, but correspond to descriptions of the arguments (COMPANIES, PARTS, DEPARTMENTS and VOLUME). For example, DEPARTMENTS might be characterized by NAME, ADDRESS, and NUMBER.

The choice of the abstraction used to map predication structures to Functional Model data structures is part of data policy. As an example, the model might be restricted as follows:

- Semantic features map to functions whose range sets are value sets.
- Arguments corresponding to "real-world" entities map to named argument sets.
- Predications map to named predication sets, and the arcs pointing to arguments become named functions. Also the predicate and its arrow are attached to the predication set.
- Downgraded predications are represented by functions whose domain is the main predication set and range is either an argument set or a value set.

Figure 3 depicts the Functional Model data structure based on the predication structure of Figure 2 and the above abstraction rules.

Thus the choice of the methodology used to model the organization is one aspect of data policy which induces structure in passing from Level 4 to Level 3. At Level 3 the Functional Model has entity sets classified as either predication sets or argument sets (denoted P and A, respectively) and value sets remain unchanged. Functions perform two roles: a function may provide information about a predication structure or it may represent a semantic feature. The data structures supported by the predication analysis and abstraction process are depicted in Figure 4. All these data structures are possible for the Functional Model, but most applications use cases b, c, and d. If we restrict further:

- Every predication structure must have at least two argument sets, and
- Functions emanating from a predication set cannot have predication sets as ranges,

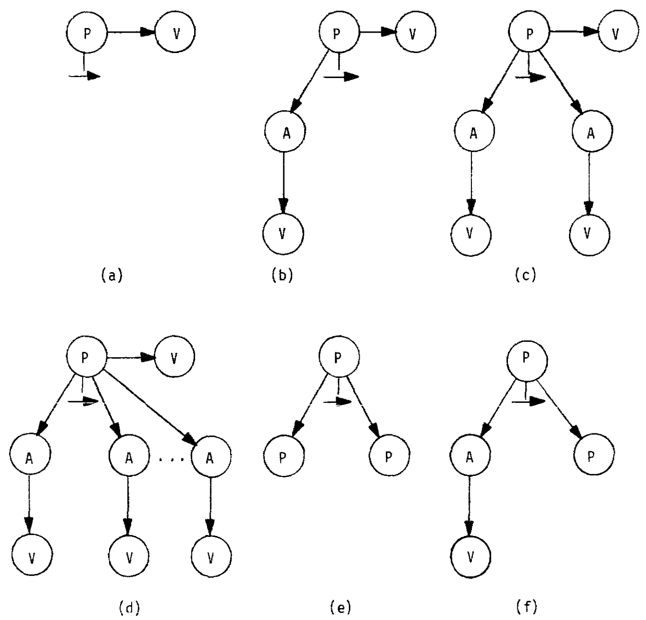
then only cases c and d are admissible. This is precisely the case in the Entity-Relationship model,[22] where predication and argument sets are called relationship sets and entity sets, respectively. In the Functional Model arcs all represent total functions, whereas in the Entity-Relationship model arcs are either 1:N mappings (for entity sets to relationship sets) or functions (for entity sets to value sets).

So far, we have only considered the methodology for obtaining data structures. Although the set types play an important role, the function types are important in expressing logical access mechanisms. Consider a binary relation α from set A to set B, and its representation, R_α, as the set of ordered pairs

$$R_\alpha=\{(a, b)\,|\,a\epsilon A \ \&\ b\epsilon B \ \&\ a\alpha b\}$$

Obviously, there exists an "inclusion" function, i, which assigns an element (a, b) of R_α to its corresponding element (a, b) of AxB, the Cartesian product of A and B. In addition, there exist functions $f:R_\alpha\rightarrow A$ and $g:R_\alpha\rightarrow B$ such that the diagram of Figure 5 "commutes."[23] In terms of the Functional Model predication structures, the binary relation α corresponds to a predication structure consisting of a predication node (labelled "α"), argument nodes (A and B) and arcs (f and g) (see Figure 5). The actual representation of the elements in the predication set is by means of the set R_α.

The functionality types of f and g are important in modelling the semantics of α: i.e., whether α is a relation, a partial function, or a total function. There are sixteen possible configurations for the predication structure, because f and g may be one-to-one, onto, both of these, or none of these. The most important configurations are



(a) (b) (c)

(d) (e) (f)

Figure 4—Level 3 functional model data structures

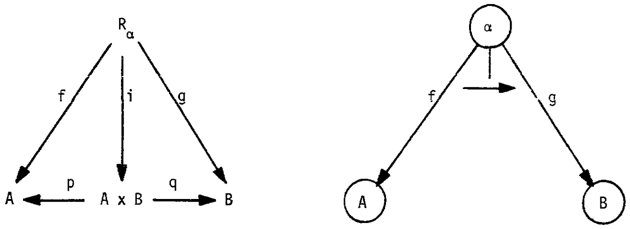


Figure 5—Representation of a binary relation and predication structure

summarized in the following:
Fact:
Let α be a binary relation from A to B with functional specification f and g.
Then:

1. If neither f nor g are one-to-one (1:1), then α is a relation;
2. If f is 1:1, then α is a partial function;
3. If f is 1:1 and onto, then α is a total function;
4. If f is 1:1 and onto, while g is 1:1, then α is a 1:1 function;
5. If f is 1:1 and onto, while g is onto, then α is an onto function;
6. If f and g are both 1:1 and onto, then α is also 1:1 and onto.

The functionality type of f determines whether α is a relation, a partial function, or a total function. If α is a function, then the functionality of g determines whether α is one-to-one, onto, a one-to-one correspondence, or none of these.

In terms of predication structures, a predication set may thus represent a relation or a function, which implies that, in the latter case, α may be represented by an arc. This is indeed true, but it must be considered a Data Policy decision.

One-to-one functions play an important role in accessing a particular element of a set. If the function f in Figure 5 is one-to-one, a particular $a \epsilon A$ will participate in (at most) one ordered pair $(a,b) \epsilon R_\alpha$, so that the second element of the ordered pair may be obtained by evaluating the function g. Thus, any one-to-one function outgoing from a set is a candidate (key) for accessing the elements of the set.

The notion of logical access can be extended to predication sets involving more than two argument sets and various value sets (as in Figures 3 and 4). In this case, the predication node represents an n-ary relationship, where each element may be viewed as an n-tuple of entities and values. For n-ary predication sets, a composite key is precisely the concept in Reference 22.

Finally we illustrate a management policy constraint whose predication structure has another predication node as an argument (e.g., Figure 4(f)). The operational constraint "A company must supply at least three parts to some department during a quarter to remain a valid supplier." could be modelled by a predication structure consisting of a "must" predication set with argument sets COMPANY and SUPPLY (a Figure 3 predication set), and semantic features (functions to value sets) TOTAL-PARTS and QUARTER. Elements of the "must" set might be updated by program on the occurrence of a SUPPLY transaction, but the validation would probably take place at the end of each operating quarter.

The relational model of data

The data policy constraints on the Functional Model predication structures which transform it into the Relational Model are:

- Value sets are *domains*;
- Argument sets and predication sets are *relations* whose elements are represented by *tuples* of values from domains;
- Functions whose range sets are value sets become the *attribute names* of their respective relations;
- Functions from predication sets to argument sets are replaced by the attribute name corresponding to the key (perhaps the composite key) of the argument set.

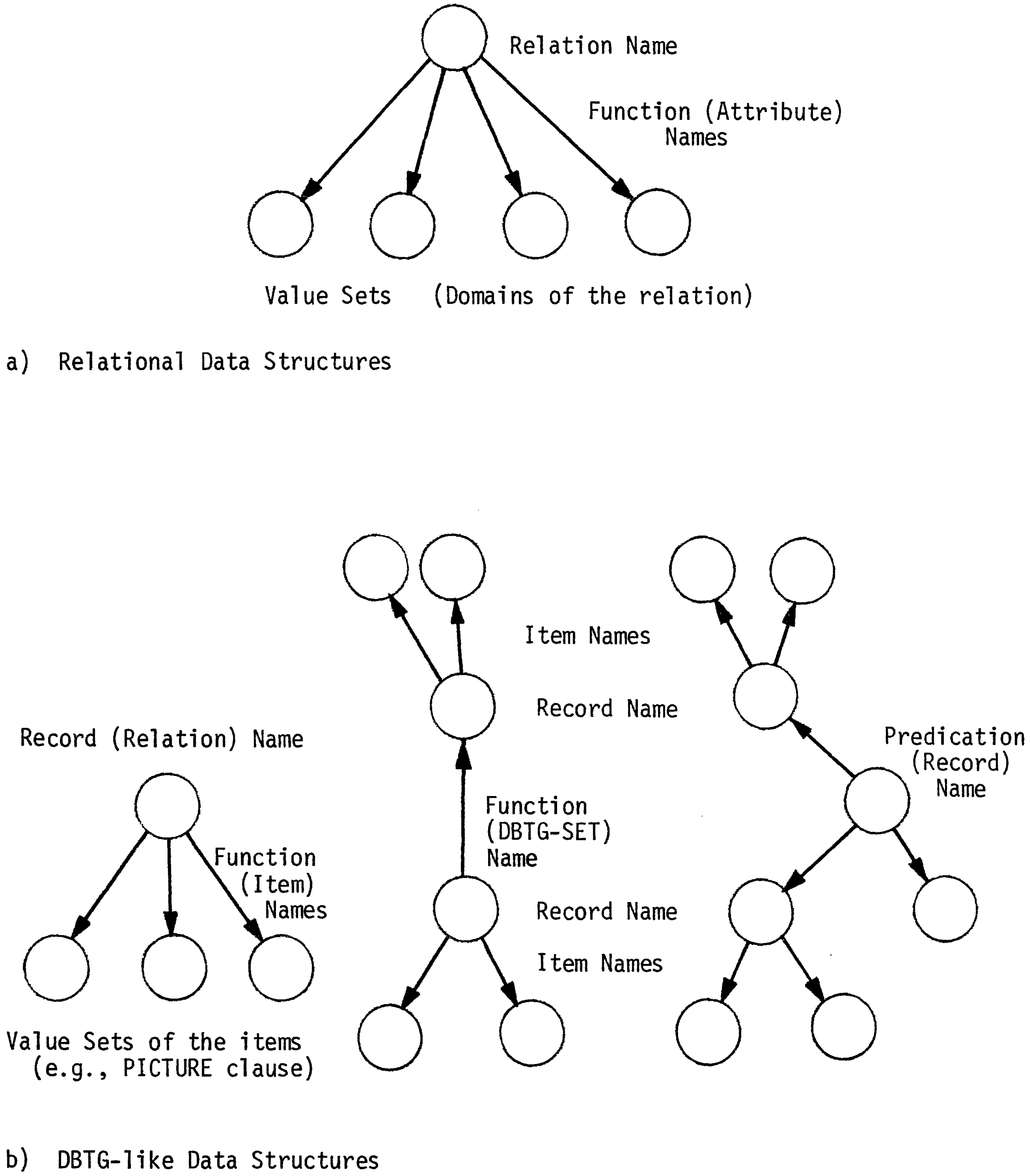
The consequence of these restrictions is to allow only nodes which represent Level 4 entity sets as relations: represented by tuples. The arcs can now only point away from nodes, and their functions are the names of the value sets of each element of the tuple: see Figure 6.

The DBTG data model

The Functional Model can also be transformed into DBTG type Data Structures by data policy definitions. First, the record will be simplified by ignoring repeating groups—in this case a record is a tuple, and may be represented in the same way as the relational data structure. The DBTG-set, in its simplest form, is a representation of a functional predication (i.e., a predication which is a function). The arrow of the function name is in the opposite direction to the arrow in the equivalent "Bachman" Diagram.[24] For a more complicated DBTG-set structure, there is a predication (a record) between two other relations (also records). This represents an intermediate or link record between two others; these structures have been discussed as linking records between two relations in Reference 25 and termed "associations" in Reference 26.

Whether the model transforms a function name into a DBTG-set or follows the Predication (Record) name model is a data policy decision: the same results may be obtained provided that one record is functionally dependent on the other—but not if the relation is N to M.

The insertion property of a DBTG-set is either AUTOMATIC or MANUAL. Which set has which property is defined at Level 3, but the ability to define these properties of a function is a Data Policy decision, which delimits the Level 4 to 3 structure and defines the DBTG Model. Similarly, the deletion properties (MANDATORY and OP-


Relation Name
Function (Attribute) Names
Value Sets (Domains of the relation)
Record (Relation) Name
Function (Item) Names
Value Sets of the items (e.g., PICTURE clause)
Item Names
Record Name
Function (DBTG-SET) Name
Record Name
Item Names
Predication (Record) Name

a) Relational Data Structures

b) DBTG-like Data Structures

Figure 6—Level 3 data structures in the functional model

TIONAL) are Level 4 to 3 data policy decisions which define the operation of a DBTG model. These two properties are therefore *declaratives* associated with the arcs, similar to the functionality types in a functional model, but obviously having stronger effect. The enforcement of this policy is, of course, procedural.

Comparisons

It is useful to compare and contrast relational and DBTG data policy operations. Relational systems like System R[27] and INGRES[28] use the theory of normal forms to provide good data structures and then utilize these simple structures through data-name linkages and data operations like JOIN and PROJECT; they apply other data policy such as consistency, integrity, and validation through system modification of the user statement (INGRES), which is immediately initiated, or through system triggers (SYSTEM R) which are transaction driven, but may be delayed. These two implementations differ in the fact that INGRES specifies policy in declaratives with procedural enforcement, while SYSTEM R is procedural both in declaration and enforcement.

The DBTG data policies are both declarative and procedural: some, such as AUTOMATIC, are declarative and apply a primitive function on operation implying a "semantics" at storage of a record; others, the Data Base Procedures, are procedural and are invoked by some trigger: e.g., on update (validation), privacy and security checking.

THE SET THEORETIC DATA MODEL

The set theoretic processor owes its existence to the concept of an extended set. The *extended set* consists of elements which themselves may be conventional sets, sequences, ordered sets, atoms, or even extended sets, but each element is identified by a position-identifier (numeric or mnemonic). The extended set is enclosed in square brackets [ ] while the ordered sets or sequences are in angle brackets ⟨ ⟩. Thus if X is an extended set consisting of elements Y and Z in positions 1 and NEXT, we have:

$$X=[\langle 1,Y\rangle, \langle \mathrm{NEXT},Z\rangle]$$

If we use braces { } to enclose sets, then if Y is the set of the first three integers and Z is the four-tuple consisting of "0,1,0,2" (in order), then:

$$Y=\{1,2,3\}=[\langle \#,1\rangle,\langle \#,2\rangle,\langle \#,3\rangle]$$

and

$$Z=\langle 0,1,0,2\rangle=[\langle 1,0\rangle,\langle 2,1\rangle,\langle 3,0\rangle,\langle 4,2\rangle].$$

Reference 16 shows that the extended set is a normal extension of set theory and that predicate calculus operations can be defined on the extended set. All normal set operations (union, intersection, difference, etc.) may also be defined, except that operations are position dependent.

Level 4 structure

The model consists of the following objects or elements: Atoms, which represent a number or character string; Sets, composed of any object; Ordered sets or sequences, composed of any object; Extended sets, composed of any object. Of course, no object may contain itself. The other model objects may all be represented as extended sets, position identifiers (which are atoms), and atoms. Commas are used to delimit objects in a sequence or set.

The basic language of the Level 4 data model consists of: predicate calculus expressions; algebraic expressions; assignment (storage) statements; retrieval expressions, with predicates to limit the response; and macros which simplify operations (e.g., Average, Sum).

In order to illustrate the operations of the extended set processor (XSP) at Level 4, a series of operations is given in Figure 7. The first ten operations are all of the "storage by assignment" type: they store ten extended sets, with synonyms (names) ME, YOU, etc. The VALUE function invokes a "copy" operation, which does not necessarily duplicate the string: the result may be represented internally by pointers. The set AUTHORS is therefore made up of the two extended sets ME and YOU.

The assignment operation for X defines a new extended set: it is a simple set containing the names of the two authors of this paper. This particular operation extracts (from the set AUTHORS) the values which have position indicator NAME. Furthermore, the summation operation (SUM) counts the elements (two) of this newly defined set X. The keyword LIST in the find instruction of Figure 7 is used to denote a set of 0 to "N" replications. The names and symbols which have not yet been described, such as PHONE, NAME, AUTHOR-TYPE and PICTURE, have no semantic meaning at Level 4. Later, some will have semantics in defining policy, but *here they are merely symbols*. At Level 4 the XSP is unrestricted. It will store, maintain, retrieve, or manipulate the whole or any part of any extended set. However, the XSP must have operations which can be triggered by events or conditions; these operations, applied in going from Level 4 to Level 3, are the Data Policy definitions.

Data policy definition

There are two types of XSP system applied constraints: static and dynamic. The *static constraint* is one that must always be satisfied: e.g., the data structure class must be hierarchic, or the access to some parts of the database are password protected. Such constraints imply a system action whenever violation occurs, and these may be implemented as special system actions ("compiled into the system") or as interpreted actions, with well defined error response. Most current systems "compile" these constraints (i.e., they "do not support other models of data" or "allow the following types of data protection . . .").

The *dynamic constraint* is one that is satisfied at specific times or for special operations; e.g., the validation of an element is only to be performed at input, or security is to be checked against type of user and type of operation for every access. These constraints are essentially interpreted.

The relational model of data

An example of the definition of allowable data structure classes for a relational model is given in the restrictions of Figure 8. This definition states that all sets are made up of ordered sets which contain "attribute-value" pairs (e.g., the elements ME and YOU in Figure 7 are ordered sets of three pairs). Moreover, the position identifiers shall be

```
ME = [<NAME,SIBLEY>,<SS#,017|32|7992>,<PHONE,(301)262-7138>]

YOU = [<PHONE,(301)937-7726>,<NAME,KERSCHBERG>,<SS#,294|36|4321>]

IT = [<TITLE,DATA ARCHITECTURE ETC.>,<AUTHOR,{VALUE(ME),VALUE(YOU)}>]

AUTHOR-TYPE = [<NAME,NAME-TYPE>,<PHONE,PHONE-TYPE>,<SS#,SS#-TYPE>]

NAME-TYPE = {PICTURE X(25 MAX)}

PHONE-TYPE = {PICTURE '('999')'999'-'9999}

SS#-TYPE = {PICTURE 999'/'99'/'9999}

ARTICLE-TYPE = [<1,TITLE-TYPE>]

TITLE-TYPE = {PICTURE X (UNRESTRICTED)}

AUTHORS = {VALUE(ME),VALUE(YOU)}

PRINT (NAME-TYPE)

X = {NAME | SET = AUTHORS}

Y = SUM(X)

Z = SUM ({AUTHOR|SET=IT})

X1 = {NAME|SET=AUTHORS AND SS# NOT='017|32|7992'}

AUTHORSHIP-TYPE = [<AUTHOR,LIST (AUTHOR-TYPE)>,<ARTICLE,ARTICLE-TYPE>]

AUTHORSHIP = {VALUE(IT)}
```

Figure 7—Some operations of an XSP at Level 4

```
POLICY RESTRICTIONS OF XSP.

     OBJECTS:  ATOM, ORDERED-SET, SET.

     MEMBERSHIP:  MEMBER (ORDERED-SET) IS <POSITION-ID, ATOM>,

                  MEMBER (SET) IS <#, ORDERED-SET>.

     CONSTRAINT:  ALL POSITION-ID (ORDERED-SET) IS MEMBER (POSITION-ID

                  (ORDERED-SET-DEFINITION)).
```

```
LEVEL 3:  DEFINITION WITHIN RESTRICTION.

     SET OBJECT.            AUTHORS.

     ORDERED-SET OBJECT.    AUTHOR-TYPE.

     ATOM OBJECT.           NAME-TYPE, PHONE-TYPE, SS#-TYPE.


TIMING.

     APPLY PHONE-TYPE, SS#-TYPE ON INPUT.

     APPLY NAME-TYPE ON OUTPUT, INPUT.

     APPLY AUTHOR-TYPE ALWAYS.
```

```
An Invalid DEFINITION would be:

     SET OBJECT.  AUTHORSHIP.

     ORDERED-SET OBJECT.  AUTHORSHIP-TYPE.

            ...  (See Figure 4.1:  this is not a first
                                   Normal Form definition)
```

Figure 8— Policy statements and level 3 definition for an XSP: Example 1: Relational model

found within a definition. Thus, if we consider the Level 3 definitions of Figure 8: AUTHOR-TYPE defines the position identifiers (NAME, PHONE, SS#), then ME and YOU conform to this type of ordered set. Moreover, the set AUTHORS in Figure 7 now conforms to the definition AUTHORS in Figure 8.

It is now obvious that Figure 7 contains definitions of extended sets which can *apply* (or be applied) with semantics at Level 3. We class the elements by statements in Figure 8 which show the definition that AUTHORS contains elements (tuples) which comply with the (Figure 7) AUTHOR-TYPE definition, while the (validation) criteria of the atoms SS#-TYPE, etc., are applied on input, with NAME-TYPE checking also on output. Moreover, because some extended set operations might allow generation of invalid sets during valid operations, the AUTHOR-TYPE criterion is applied on all operations (this may be a duplicative statement, depending on the overall constraint . . . ALL POSITION-ID . . . etc., being universally applied, or "compiled into the system").

If we gave XSP, the definition for AUTHORSHIP-TYPE, an error must occur, because the definition in Figure 7 allows non-atomic elements in the ordered set (due to LIST).

The DBTG data model

In dealing with the definition of a DBTG-like structure, one is faced with some prior decisions. Using the definition of Reference 25, a DBTG-set consisting of an owner record (A_i) and three member records (B_{ij}, B_{ij}', B_{ij}''), in that order, will be represented as

$$\langle A_i, \langle B_{ij}, B_{ij}', B_{ij}'' \rangle\rangle$$

The restrictions imposed in an earlier section on DBTG-records is applied here also: no repeating groups are allowed. Thus the definition of a record in Figure 9 follows that of an ordered set in Figure 8. The definitions therefore are special only in their inclusion of membership in DBTG-sets.

Data Policy is represented by the ability to have AUTOMATIC operation of DBTG-set inclusion on storing a predefined record.

Comparisons

It has been suggested that the implementation of a relational structure in a DBTG system is a matter of:

1. Only allowing system owned sets (DBTG termed this a "SINGULAR SET");
2. Removing concepts of database procedures, inclusion and deletion properties (AUTOMATIC, MANDATORY, etc);
3. Making keys unique (using CALC with a DUPLICATES NOT ALLOWED clause);
4. Allowing new macros like JOIN and PROJECTION on (mathematical) sets of records.

It will be seen that the definitions do not truly reflect the first requirement, and that the second can be considered a triggered or system applied procedure (the result of an operation depending on the functional properties of the DBTG-set, etc.). Thus the restrictions are not correct. By further refining the models, it should be possible to determine true similarities and differences. However, it is first necessary to add the operations and their mappings—a non-trivial task.

```
POLICY RESTRICTIONS OF XSP.

    OBJECTS:     ATOM, RECORD, DBTG-SET, SYSTEM-SET.
    MEMBERSHIP:  MEMBER (RECORD) IS <POSITION-ID, ATOM>,
                 MEMBER (DBTG-SET) IS <1, RECORD> OR
                 <2, LIST {RECORD}> ,
                 MEMBER (SYSTEM-SET) IS DBTG-SET.
    CONSTRAINT:  ALL POSITION-ID (RECORD) IS MEMBER
                 (POSITION-ID (RECORD-DEFINITION)),
                 ...
```

```
LEVEL 3:  DEFINITION WITHIN RESTRICTION.
    SYSTEM-SET OBJECT.  AUTHORSHIP
    DBTG-SET OBJECT.  AUTHORSHIP-TYPE.
    RECORD OBJECT.  ARTICLE-TYPE, AUTHOR-TYPE.
    ATOM OBJECT.  TITLE-TYPE, NAME-TYPE, PHONE-TYPE, SS#-TYPE.

TIMING.
    APPLY AUTOMATIC TO ARTICLE-TYPE ON STORE.
```

Figure 9—Policy statements and Level 3 definition for an XSP: Example 2: DBTG model

CONCLUSIONS

Work with two GDS (Functional and Extended Set Models) leads us to the following conclusions:

1. The model of an enterprise information structure may be defined independently of the GDS used for its implementation.
2. If the GDS is "universal" it may store any information structure (both data syntax and semantics) in terms of its primitives.
3. The specialization of the GDS to a data model like that supported by most current research and commercial DBMS involves Data Policy definition. Using this, a GDS may be restricted to perform as one or more specific data models.
4. The process of mapping from an information structure within a GDS to a data structure within a traditional

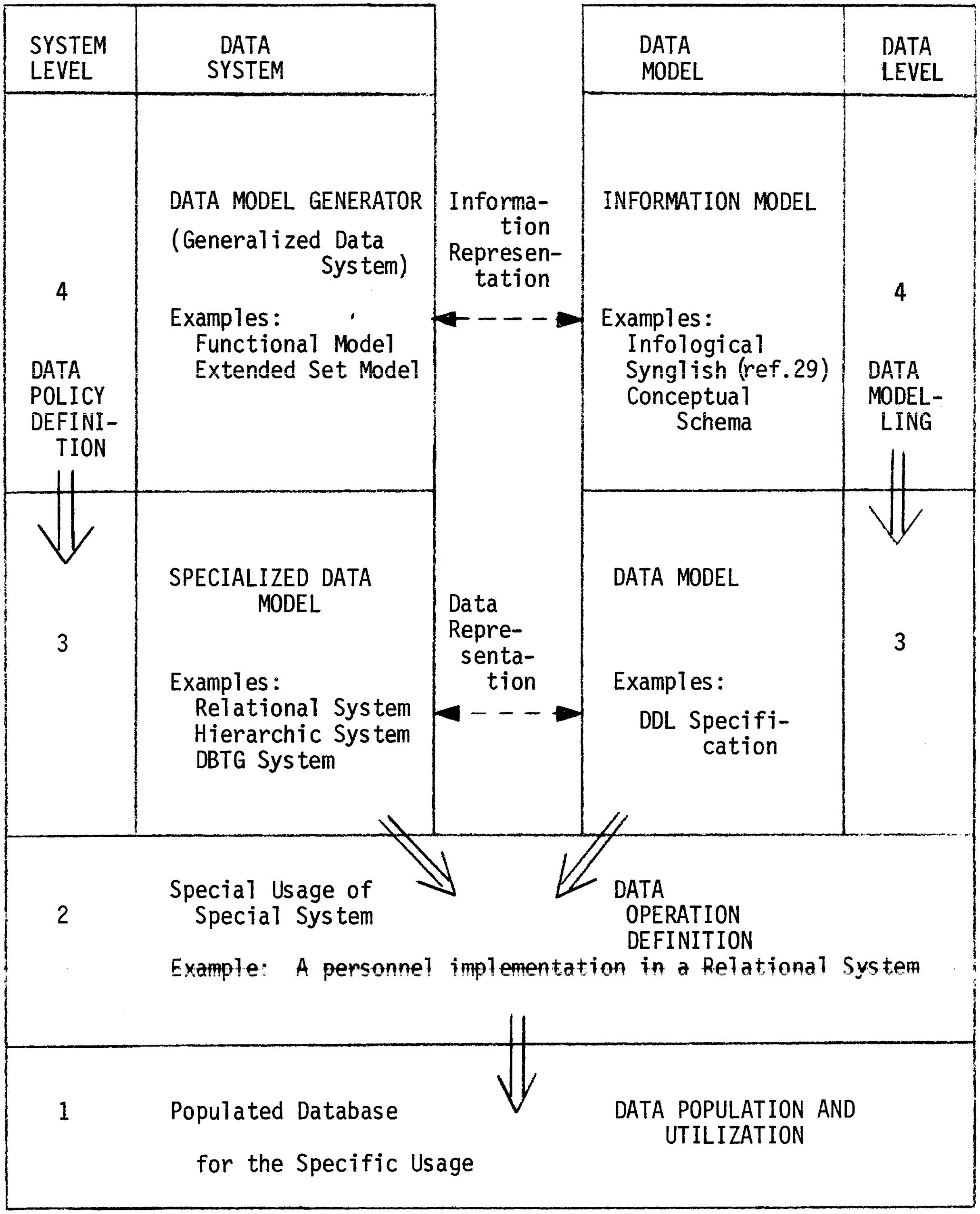


Figure 10—The parallelism of data policy and modelling

DBMS follows the restrictions of the Data Policy definition. Consequently there must be a correspondence between Data Policy restriction and *data modelling* (i.e., passing from information structure to data structure). This process is diagrammed in Figure 10.

Most information analysts already have a specialized data model (e.g., DBTG systems) in mind when constructing their information model. Thus, they take the Data Modelling route in Figure 10. We suggest that the correct (more general) process is to express conceptual information structures in the GDS and to "tune" the information structures to data structures by means of Data Policy decisions. In other words, it is advantageous to represent information at Level 4 so that all semantics of the information are retained.

Data Policy definitions impose added structure (with restrictions) on allowable data structures. Tradeoffs will then make it easier to consider the losses in representation of information structures in the supported data structures.

REFERENCES

1. Sibley, E. H., Guest Editor: "Special Issue on Data Base Management Systems," *ACM Computing Surveys,* Vol. 18, No. 1, March 1976, pp. 151
2. Kerschberg, L., A. Klug and D. Tsichritzis, "A Taxonomy of Data Models," *Proceedings Second International Conference on Very Large Data Bases,* Brussels, September 1976.
3. Rustin, R., Editor: *ACM SIGMOD 1974 Workshop on Data Description, Access, and Control,* "Data-Structure-Set versus Relational," May 1974, pp. 144.
4. CODASYL Systems Committee: "The Selection and Acquisition of Data Base Management Systems," Published by ACM, New York and IAG, Amsterdam, March 1976, pp. 252.
5. Reiter, A., "Data Models for Secondary Storage Representations," *Proceedings 1st International Conference on Very Large Data Bases,* ACM, Sept. 1975, pp. 87-119.
6. Hardgrave, W. T., and E. H. Sibley, "Database Research: Some Comments on Future Directions," *SIGMOD FDT 7,* pp. 3-4, 1975.
7. Kerschberg, L. and J. E. S. Pacheco, "A Functional Data Base Model," Computer Science Monograph, Pontificia Universidade Catolica do Rio de Janeiro, February, 1976, also available as Technical Report 13, Dept. of Information Systems Management, University of Maryland, 1976.
8. CODASYL Data Description Language Committee, "Data Description Language-Journal of Development," National Bureau of Standards Handbook 113, Washington, 1973, pp. 136.
9. Novak, D. O. and J. P. Fry, "The State of the Art of Database Design," *Proceedings Fifth Texas Conference on Computing Systems,* Austin, 1976, pp. 30-38.
10. Langfors, B., "Theoretical Aspects of Information Systems for Management," *Proceedings IFIP Congress 74,* pp. 937-945.
11. Rund, D. Sheppard, "Data Base Design Methodology Parts I and II," AUERBACH Publishers Inc., 1976.
12. Kahn, B. K., "A Method for Describing the Information Required by the Data Base Design Process," *Proceedings International ACM-SIGMOD Conference on Management of Data,* Washington, D.C., 1976, pp. 53-64.
13. Rothnie, J. B. and W. T. Hardgrave, "Data Model Theory: A Beginning" *Proceedings Fifth Texas Conference on Computing Systems,* Austin, 1976.
14. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," *Comm. ACM,* 13, June 1970, pp. 377-387.
15. Childs, D. L., "Description of a Set-theoretic Data Structure," *AFIPS Conf. Proc.,* Vol. 33, Part 1, AFIPS Press, Montvale, N.J., 1968, pp. 557-564.
16. Childs, D. L., "Feasibility of a Set-theoretic Data Structure: A General Structure Based on a Reconstructed Definition of Relation," *IFIP Congress 1968,* North Holland, Amsterdam, 1968, pp. 420-430.
17. Childs, D. L., "Extended Set Theory: A Formalism for the Design, Implementation, and Operation of Information Systems," Volume IV, *Current Trends on Programming Methodology,* edited by R. T. Yeh, Prentice-Hall,
18. Hardgrave, W. T., "A Technique For Implementing a Set Processor," *Proc. ACM Conference on Data: Abstraction, Definition, and Structure,* SIG-PLAN Notices, March 1976.
19. Hardgrave, W. T., "Set Processing: A Tool for Data Management," *Proc. ACM/NBS Fifteenth Annual Technical Symposium,* June 1976.
20. Eswaran, K., "Aspects of a Trigger Subsystem in an Integrated Database System," *Proceedings Second International Conference on Software Engineering,* San Francisco, 1976, pp. 243-250.
21. Leech, G., *Semantics,* Penguin Books Ltd., Middlesex, England, 1974.
22. Chen, P., "The Entity-Relationship Model—Toward a Unified View of Data," *ACM Transactions on Database Systems,* Vol. 1, No. 1, March 1976, pp. 9-36.
23. MacLane, S. and G. Birkhoff, *Algebra,* Macmillan Co., 1968.
24. Bachman, C. W., "Data Structure Diagrams," *Data Base,* ACM SIGBDP Newsletter No. 1, 2, Summer 1969.
25. Sibley, E. H., "On the Equivalences of Data Based Systems," *ACM SIGMOD 1974 Workshop on Data Description, Access, and Control,* May 1974, pp. 43-76.
26. Schmid, H. A. and J. R. Swenson, "On the Semantics of the Relational, Data Model," *Proceedings International ACM-SIGMOD Conference on Management of Data,* 1975.
27. Astrahan, M. M. et al., "System R: A Relational Approach to Data Base Management," *ACM Transactions on Database Systems,* Vol. 1, No. 2, 1976.
28. Stonebraker, M., "High Level Integrity Assurance in Relational Data Base Management Systems," *Proceedings of the 1975 ACM-SIGMOD Workshop.*
29. Kerschberg, L., E. A. Ozkarahan, and J. E. S. Pacheco, "A Synthetic English Query Language for a Relational Associative Processor," *Proceedings Second International Conference on Software Engineering.* San Francisco, 1976, pp. 505-519.

Security risk assessment in electronic data processing systems*

by ROBERT H. COURTNEY, JR.
IBM Corporation
New York, New York

ABSTRACT

Concern for the safety of a data processing facility and the data within it should result in the selection of such security measures, including insurance, as are appropriate to bringing the risk within tolerable limits at the lowest cost. These security measures should be selected on the basis of the benefit/cost relationships which they afford. This, in turn, requires a quantification of the potential benefits afforded by each security measure or group of measures for comparison with the cost. Because the benefit afforded by the security measure is lessening or elimination of security problems, which is risk reduction, we must be able to quantify the risk so as to measure the benefit afforded by its elimination or diminution. A workable procedure for doing this is described.

INTRODUCTION

Most management decisions involve the assumption of risk—the chance that things may not turn out the way we hope or want them to. Decisions made in spite of uncertainties and, indeed, in recognition of them are generally accepted as essential to dynamic, successful management. Most frequently, however, the key to success lies not in the willingness to accept uncertainty, or to assume risk, but in the ability to recognize and quantify the elements of that risk so as to deal with them in a fully objective way. Virtually every manager must come to grips with and manage risk in some form. For this reason, risk management has become recognized as a distinctly identifiable function of general organization or project management.

The steadily growing dependence of virtually every kind of organization on electronic data processing systems has introduced new concerns, which themselves have grown rapidly in the past few years. These concerns are attributable to a wide variety of factors, but there are two principal ones. First is the recognition of the recent rapid growth in the centralization of the data keeping and information extraction processes with the attendant potential for the loss of the entire facility or major portions of it. Such loss might result in a severe set-back for the organization. The second major factor is recognition of the increasing dependence of the enterprise on employees with skills, talents, and disciplines, and sometimes motivations, quite different from those with which management has been familiar in the past. There is a feeling that these people might present new, unfamiliar problems and unfamiliar problems generally yield more discomfort than do familiar ones.

This paper was prepared in response to a clear need for a rational, systematic approach to a quantitative analysis of the security risks associated with electronic data processing systems. There has been broad agreement for some time that a risk analysis technique is needed, but no readily workable, broadly applicable technique has been described in sufficient detail for it to be in general use. A methodology is offered here for assessing the risks specific to an organization, the particular system within that organization, and, to the extent necessary, the specific entities of which each system is comprised. More specifically, the purpose of this document is to describe one way of doing a risk analysis which has been found workable by several organizations.

Earlier versions of this paper provoked active discussion of the precise nature of those undesirable things which should be considered appropriate topics for risk assessment. Opinions ranged from a belief that consideration should be given only to catastrophic events, which would result in a complete loss of the data processing capability, to the opinion that only events ascribable to intentional misconduct need evaluation. At the opposite extreme, some held the position that consideration must be extended to all things which might result in undesired modification, destruction, or disclosure of data or suspension of data processing services. The latter is the correct approach. While this will increase the work required to complete the risk analysis, there is no basis for a′ priori exclusion from consideration of subsets of the total array of undesirable things which can happen to data and data processing capabilities. It is not until the impact of the event and its

* This paper is derived from a working document prepared by the author for study and use by the Federal Information Processing Standards Task Group 15, Computer Systems Security, of the United States Department of Commerce National Bureau of Standards.

frequency of occurrence have both been examined, which is in fact a risk assessment, that there can be a reasoned justification for the exclusion from further consideration of any potential source of damage.

It has been argued that the inclusion into the relatively formal risk analysis procedure of such things as data entry errors, mistakes in handling tapes, and operator errors is an invasion of the normal province of the data processing manager. It can also be viewed as a tool to assist him in the identification of the need for and selection of appropriate safeguards.

The purpose of performing a risk assessment is to obtain a quantitative statement of the potential problems to which the data processing facility is exposed, so that appropriate, cost-effective security safeguards can be selected. It is assumed that, once armed with such information, no security measure will be selected which costs more than tolerating the problem. The risk assessment should establish that threshold.

THE TWO PRINCIPAL FACTORS

Most people who have seriously considered or attempted to devise a risk analysis procedure readily agree that a useful technique must yield a quantitative statement of the impact of specific security problems. However, there is less than uniform agreement that the unit of measure can or should be monetary. In addition to a measure of impact, a statement of the probability of the occurrence of a particular event is essential to a useful risk assessment. We contend here that the two key elements in risk analysis are:

1. a statement of impact, that is, a quantitative statement of how badly a specific difficulty hurts, and
2. a statement of the probability of encountering that difficulty in a specified period of time, which is, of course, anticipated frequency of occurrence.

Both parameters are needed to describe risk in terms of cost per unit time, such as dollars per year.

The probability of an undesirable thing happening is usually more difficult to determine with confidence than is a measure of the impact of its happening. It has been suggested that we are so accustomed to making unconscious, gross, subjective judgments of probability in reaching decisions that it is difficult to accept a formalization of the process. Whatever the reasons for finding it difficult, statements of the potential economic impacts of events, without regard to their relative probability, cannot lead to the identification of those security exposures that are worthy of corrective action and those which are not. There are, of course, many events which could have catastrophic consequences but which appear to have such low probability of happening as to not justify the expenditure of significant resources to lessen the potential damage. For example, at another time and in another social climate, we judged the probability of nuclear attack to be sufficiently high that we were persuaded to provide and stock fallout shelters. We have now, for the most part, abandoned those shelters, not because the damage which would be caused by such an attack is believed lower, but because we judge the probability of attack to be too low to justify the cost, in dollars and inconvenience, of maintaining these facilities. Our decision to abandon this security measure was based on a reassessment of the probability of occurrence and not on its cost.

As another example, assume that a hypothetical major corporation has centralized most of its data processing facilities into a single location. Also assume that no plans have been made for data processing support elsewhere in the event of a catastrophic loss of that facility. Suppose that the sum of all costs to the corporation of such a loss is, at first estimate $150 million, including not just the replacement costs of hardware, but also lost business opportunities, customer ill-will, interruption of proper cash management, and other key activities. As such, the availability of the $150 million figure, while possibly interesting, does not lend itself to any real measure of the problem. It does not suggest how much might reasonably be spent in reducing the exposure. If we then determine, through further analysis, that we might reasonably expect such a loss with a frequency of .003/year, we do have a basis for a decision on corrective action. It is clear that we have an exposure in the order of $450,000/year.

Continuing our example, we have three options in addressing the exposure; we can:

1. Tolerate it.
2. Lower the dollar impact by implementing those measures which cost less than a total of $450K per year, if any.
3. Lower the probability of the loss occurring by implementing protective measures costing less than the exposure.

The point made here is that, unless we had quantified both the impact and the probability of occurrence, we would not be in a position to make an informed election of any of the three options.

Parenthetically, it should be noted that insurance is *not* a fourth option. Insurance provides a means of smoothing the impact of the loss when and if it happens. As such, it is a matter to be considered after the election of the other options. The availability of insurance does not necessarily lessen the desirability of minimizing risk by other measures. The downward adjustment of risk should lessen the *amount* of insurance required (in the case of reduced impact) or the *rate* (in the case of reduced probability or occurrence). In the unlikely event that such reductions do not change the cost of insurance, this should affect either the decision to insure or the decision to apply security measures.

It was stated earlier that there is something less than complete agreement that the impact of security problems is best measured in dollars. Those people who seem to have the most difficulty in assigning dollar values to security problems are, for the most part, either:

1. considering the safety of data collections which, if disclosed or otherwise harmed, would have some identifiable and undesirable political or social ramifications, and are possibly affected by privacy legislation, or
2. have an involvement with defense or intelligence activities. The risks associated with activities in these two categories are generally much more difficult to assess quantitatively than are many other exposures. However, this does not lessen the desirability of such assessment.

Reluctance to use dollars as a means of sizing the negative social impact of security problems is understandable. Many people will not look kindly on those who appear coldly to assess in dollars the hurt which might befall people as a consequence of some security problem which impacts those people personally. The appearance of an objective measurement in dollars, for example, of an individual's privacy concerns might be abhorrent to many people. This reluctance to use dollars as the measure has led to several parallel development efforts to define the severity of problems in these categories using relative sensitivities as, for example, on a scale of one to five. Such rating schemes are potentially valuable. Their development and use should be encouraged. They are means of communicating an assessment of the potential for harm to people of the loss of security to files of specific types. For example, a rating of "1," indicating great sensitivity, for psychiatric data and "2" for files containing personal data which is a little less sensitive, such as tax files. Such ratings do not, however, provide an adequately meaningful parameter for guidance in the selection of economically feasible security measures. Such rating schemes can and should coexist with risk analysis techniques which quantify the problem in dollars.

It is conceivable that a convention which uses relative sensitivities on a scale of one to five can be coupled with another which describes probability of occurrence to provide an expression which says that the probability of a 2-sized problem is 0.3/year. However, this does not provide much help in evaluating the need for, or relative effectiveness of, specific security measures.

A specific security measure is often difficult to justify in terms of its ability to contain only one problem, and the best security measures are usually those which contain or assist in containing more than one.

Any summation of risks contained by a specific measure or combination of measures requires that these risks be expressed in common units of measure. If some problems are describable only in economic terms and others in nondimensional sensitivity ratings, the ability of specific measures to contain a variety of problems will be awkward to assess and security measures difficult to cost-justify.

Much of the problem of quantifying subjective concerns often seems to go away if, in performing the risk analysis, we defer until last the decisions we do not know how to make. It then frequently becomes clear that other, more easily quantifiable concerns fully justify the required security enhancements. Under these fortunate circumstances, we are relieved of the need to make these decisions.

As an example, it may be difficult to arrive at a quantitative statement of the privacy impact of an exposure of the violations records associated with vehicle operators' license files. That such data are generally public record does not lessen belief that their consolidation in a machine-based system results in potential for their unfair exposure to too many people who should not or do not need to see them. The clear need to protect such data against unauthorized changes will justify a level of security sufficiently high to displace concerns for disclosure as the dominant factor in selecting security measures.

It is not argued that serendipity will always prevail in such matters, but there is no reason to ignore any assistance that it might provide. The present limited experience in applying this risk analysis methodology to data collections where social concerns are of major importance leads to the strong belief that data security considerations other than protection against disclosure will often dictate security measures adequate for protection against disclosure and thus relieve the need for quantification of the social impact, real or imagined.

While those considering the problem of the social impact of losses of data security have trouble expressing in dollars the damage which might be done to people, the defense establishments have a greater problem in addressing the other factor in the risk analysis expression, the probability of occurrence.

It is usually not easy to assess the dollar implication of losses of classified data or denial of processing capability. An even greater problem is encountered when trying to assess the probability of espionage and sabotage. Because the losses in such cases can be very great, it becomes difficult to accept as tolerable any probability of occurrence. This dilemma can lead, if we are not careful, to such logical dead ends as the statement that "if it can happen, we must assume it will happen with a probability of 1."

It is important to avoid the inclination to overemphasize the significance of technical problems simply because their solutions are intellectually challenging. Frequently, the more intellectually stimulating problems are also those with low probabilities of occurrence. The probability of occurrence of these more exotic problems is lowered by the limited number of people in a position to pose each specific problem. Thus, we are more inclined to concern for the potential for damage by a systems programmer, of whom there are relatively few, but whose capabilities provide quite challenging problems, than we are to a critical evaluation of the effectiveness of a guard force in keeping strangers, of whom there are many, out of a facility and in restraining persons from carrying off what they should not remove from the premises, including media for data storage.

In many electronic data processing environments, the number of people in a position to do damage tends to be inversely proportional to the amount of damage which they might do. This is desirable and should be a goal of those concerned with the security of these facilities. Unfortunately, this comment does not apply to many other security-sensitive environments not involving electronic data processing.

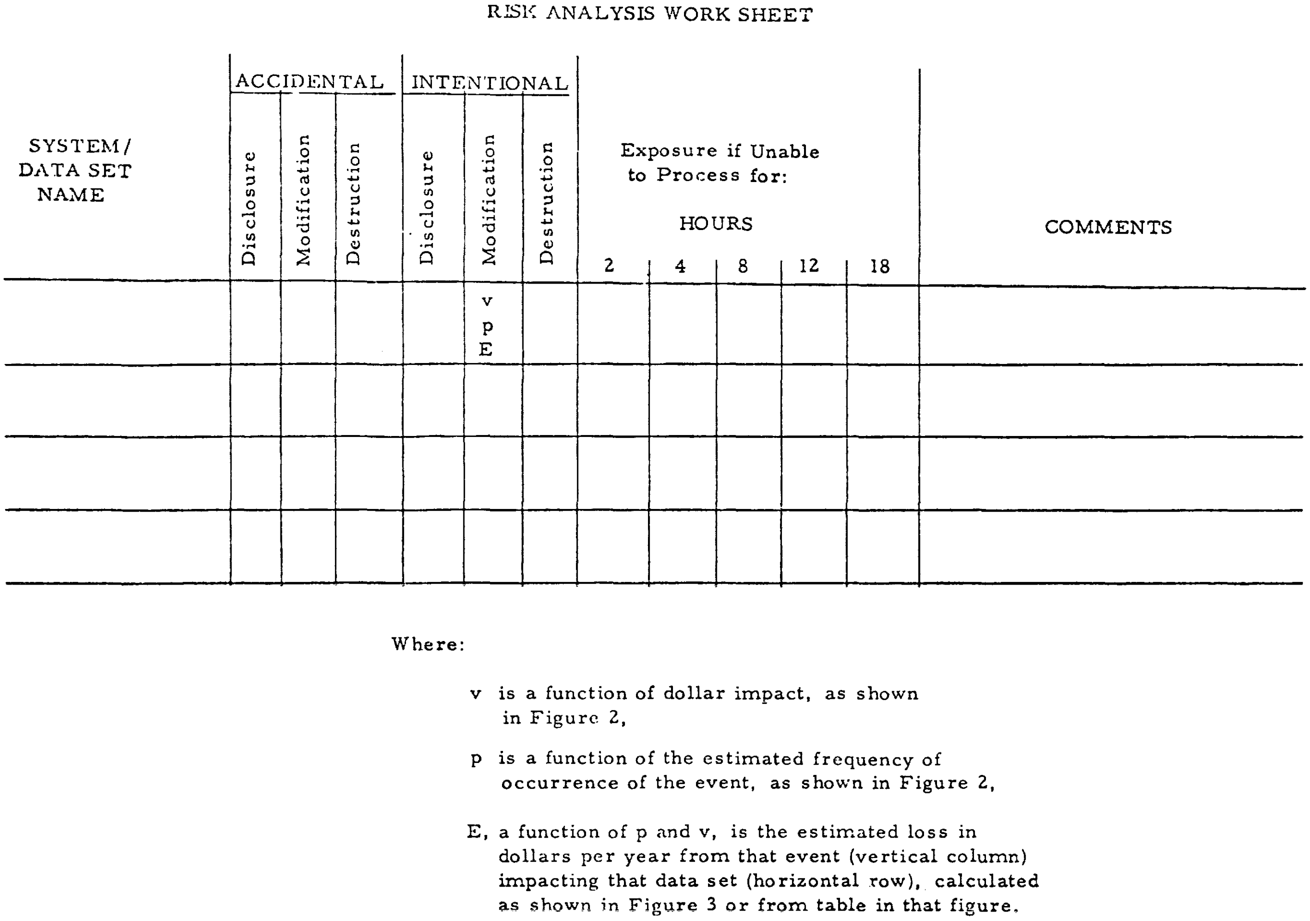
The great majority of systems in both government and the private sector fortunately do not have their security needs dominated by unquantifiable events or probabilities of occurrence due to undefined potential social impact or defense problems. Even in systems free of these problems, it may at times be difficult to arrive at precise assessments of event impact or probability. It is usually quite feasible to arrive at figures which, while inexact, are good enough for the purpose of evaluating exposures and for guidance in selecting appropriate security measures.

THE METHODOLOGY

The proposed risk analysis procedure proposed is fairly well described by the sample form shown as Figure 1. The form is for use in evaluating the risk of damage to data from all causes, including its loss to physical threats, such as fire.

Data security problems are those which result from the accidental or intentional, but unauthorized, disclosure, modification, or destruction of data or the loss of the ability to process it. There are, then, six bad things which can happen to data and, in addition to those six, there can be the denial of processing capability. It is important that all six be kept in mind because security measures to be fully cost effective need to address the broadest possible array of problems. If our attention converges on too narrow a definition of data security it is likely that a set of security measures could be selected which contains a smaller problem scope than would have other measures selected with the broader problem definition in mind.

Because there are six categories of undesirable things which can happen to data in addition to the loss of the ability to process it, and because the negative dollar impact of these things or their probability of occurrence, or both,



RISK ANALYSIS WORK SHEET

	ACCIDENTAL			INTENTIONAL			Exposure if Unable to Process for:					
							HOURS					
SYSTEM/ DATA SET NAME	Disclosure	Modification	Destruction	Disclosure	Modification	Destruction	2	4	8	12	18	COMMENTS
					v p E							

Where:

v is a function of dollar impact, as shown in Figure 2,

p is a function of the estimated frequency of occurrence of the event, as shown in Figure 2,

E, a function of p and v, is the estimated loss in dollars per year from that event (vertical column) impacting that data set (horizontal row), calculated as shown in Figure 3 or from table in that figure.

Figure 1—Risk analysis worksheet

may vary widely as a function of which data is being considered, experience has shown it to be desirable to look at the dollar impact of an event and its probability of occurrence in a rather fine-grained structure; that is, looking at the results of each bad thing happening to every file or dataset.

The selection of appropriate security measures is highly dependent upon the specific problem to be contained. If our problem structure is too coarse, combining, for example, the consequences of both accidental and intentional things, the results will not usually provide the desired guidance to select a set of cost-effective protective measures.

Refer now to the format of the sample work sheet. The far left column is for listing an inventory of the data. It is suggested that these files be aggregated by system, such as "Life Beneficiary Pay," "Inventory Control, Sub Assy," or "Personnel Data, Non-Exempt Emp.," because they are most conveniently inventoried this way and are always most conveniently considered for risk assessment in the context in which they are used. "System" is used here to describe a major application area, as implied by the examples, and does not imply a physical facility.

Some files are used to support more than one system. In such cases, it may be more convenient to list them with each system in which they are used, and note in the Comments column that they have been so listed. It is not safe to list only once those files which are used to support several systems because the system with which they are included may be the one less dependent on that file than other systems might be.

The first objective is to assign values for impact and probability for each intersection in the matrix. Many intersections describe problems which are sufficiently small that they may be neglected. Ordinarily, if the sum of v and p, as described in Figure 2, is less than 6, then it can be neglected. In some cases it is acceptable to set the threshold higher, but caution must be used. There may be security measures which will contain a large number of low cost problems but which cannot be cost-justified unless several of these small problems are identified. Care must be taken to avoid disregarding an intersection because the per-instance dollar impact is low; it may well be that the probability of occurrence is sufficiently high to yield a high annual cost for this problem. If the cost of a particular data entry error is only $10, this should not be ignored as too small to be important until it is also known that it does not happen many times per day.

If the dollar impact of the event (vertical column) on the data set (horizontal row) is:

$ 10, let v=1
100, let v=2
1,000, let v=3
10,000, let v=4
100,000, let v=5
1,000,000, let v=6
$10,000,000, let v=7

If the estimated frequency of occurrence is:

Once in 300 years, let p=1
Once in 30 years, let p=2
Once in 3 years, let p=3
Once in 100 days, let p=4
Once in 10 days, let p=5
1 per day, let p=6
10 times per day, let p=7
100 times per day, let p=8

Figure 2—Selection of values of v and p

There is a strong tendency to attempt impact and probability assessments far more exact than are actually required. This contributes materially to the time required to complete a risk assessment, without a corresponding increase in the value of the product. It is not uncommon when working on a risk analysis to find the discussion bogged down on the question of whether there is, in a particular instance, a $115,000 or a $132,000 problem when, in fact, it makes no difference which is chosen.

It is better to do the risk assessment making every gross estimate of both impact and probabilities, and then refine specific items later only if it is found that a decision to pursue a particular course requires greater precision. For this reason, an artifice is proposed to induce the risk analysis team to be sufficiently inexact, at least on the initial pass, to complete the job in a reasonable time. The use of factors of 10 (orders of magnitude) for both dollar impact and probability is recommended.

It must be understood that the assignment of probabilities to specific human behavior problems in this area cannot be done from a sound statistical base. This should not seriously inhibit the risk analysis. With on-going systems with which there is a body of experience, particularly as it applies to high-probability errors and omissions problems, the task is relatively easy. There is usually an experience base from which the team can work. It is usually more difficult to assign probabilities to dishonest behavior problems. Nevertheless, with the proposed gross quantification intervals, it has not been found too difficult.

Even if all white collar crime were reported to law enforcement agencies, (as opposed to the estimated 10-15 percent of the detected instances) there would still not be a statistical base which would provide data better than informed judgment based on a thorough knowledge of the environment under consideration. People are far too complex to permit a statistically-based behavior analysis as it relates to the probability of members of groups committing specific crimes. In fact, if all the people in the United States who are users of data processing systems were simple gas molecules in a teacup, there would not be enough of them to obey the basic pressure/volume/temperature relationships of the gas laws. It is clear that the behavior of people, with their near-infinite complexity, will not yield to a rigorous, statistically-based, behavior prediction.

Common sense is a very powerful weapon in attacking a probability analysis. In a life insurance beneficiary payment system where several hundred to a thousand or more people know that it is relatively easy to change beneficiary address without risk of anyone verifying that new address, there is an exposure to at least one dishonest person successfully diverting checks to an address or mailbox from which he can get and then cash the checks. Such a situation should yield a probability much higher than once in 30 years, probability much lower than every ten days and so, using our exponential scale, we are discussing either once every 1000 days or 100 days. The selection of the more appropriate one of these two depends on several factors, including the general climate in which the system functions. If the number of people who know of the potential exposure is in the order of one to two hundred, it is perhaps reasonable to work with the three year, or 1000 days, probability. If the number of such people is in the thousands or if employee dishonesty is a sustained problem, then the 100 day approximation is probably better. This selection is left to the risk analysis team. However, the team must consider the general environment. If employee dishonesty is relatively rampant and accepted by management so long as it does not exceed established bounds, then a much higher probability of loss must be anticipated than would otherwise be the case.

It is quite commonly stated that there is no justification for an attempt to identify a back-up facility because no one else can possibly spare the machine time necessary to replace the whole capability of the system which is down or which was lost in some fire or other catastrophe. The flaw in this rationale is the assumption that it is necessary to find a facility which can replace *all* of the capability that was lost. It is generally true that only 10 to 15 percent of the work is so critical to the organization that it cannot be deferred for four to six days without catastrophic impact. It is important that this 10 to 15 percent be identified, and that contingency plans be laid which include the availability of all of the things necessary to process elsewhere, including forms, programs, communications, data and people.

The time intervals which should be on the form vary with the nature of the organization under consideration. The intervals shown are more appropriate to large scale commercial banking, for example, than they are to fire and casualty insurance companies. The risk analysis team must determine the time intervals useful to their particular industry area. Once these time dependencies are well understood, the data processing manager will usually find that the amount of processing required and the cost of an inability to process after loss of his facility makes an arrangement with other facilities for back-up not only feasible, but highly desirable.

HELPFUL HINTS

It is important that proper weight be given to the impact of errors and omissions. Data is more often destroyed or otherwise rendered useless, or even harmful, by people making mistakes than through dishonesty or malice. People whose loyalty and honesty are unquestioned, but whose judgment and competence leave much to be desired, are our greatest enemies. Data security considerations must not be limited to concern for the acts of dishonest people. Otherwise, it is very difficult to achieve proper cost justification of appropriate security measures. Weigh *all* potential security problems.

It is of utmost importance when considering the potential for damage by dishonest or malicious people to keep in mind that the vast majority of all white collar crime is committed by employees, not outsiders.

Most of the losses to dishonest employees occur when employees misuse system resources to which they are authorized access to get their jobs done. The people who steal from Accounts Payable usually work there or are authorized to enter or modify these data. The people who steal from inventory through manipulation of the data processing facility most commonly work in Inventory Control. The people who work in Accounts Payable usually do not steal from inventory or payroll. This situation must be kept in mind when considering the exposure to data security problems. Most improprieties against property directly involving the data processing system are conducted by people who work in that portion of the business from which the theft occurred, and by people who are very familiar with that particular functional area.

It is usually best to eliminate perceived individual personal integrity when performing a risk analysis. While the probability than an individual will engage in dishonest conduct clearly varies widely from person to person and clearly influences the exposure to problems originating in this manner, the factors which influence this individual integrity are not easily perceptible. Further, individual personal integrity is not a constant. It varies dramatically with time and with personal situations of which a risk evaluation team may be totally unaware. Satisfaction of personal pride, frequently reflected in care and precision in the conduct of a job, an admirable trait, can also lead to other endeavors to satisfy this pride. Conflicts between two ethics, for example, the need to pay for an urgently needed operation on a child and a desire to be honest, can be resolved in a manner not favorable to the employer. The highly motivated employee who feels passed-by on promotions may decide to get his increased income in a manner of his own choosing. For these reasons, it seems most satisfactory to eliminate individual personal integrity as a factor in the risk assessment.

People in a position to engage in white collar crime are sometimes deterred when the potential for reward from dishonest conduct is limited to the absolute minimum necessary to the conduct of the job to which he is assigned. They frequently are deterred by reasonable assurance of detection if they do something wrong. This is to say that people are primarily deterred by limits on the value to them of their dishonest activities, by the fear of being caught and, to a lesser extent, by fear of punishment.

The loss of the physical facility itself should be treated independently of the matter of loss of processing capability. It is misleading to consider the loss of processing capability as part of the cost of a loss of the physical facility. The loss of the physical facility in a properly planned operation may not result in a loss of all processing ability, and the loss of all processing ability need not involve the loss of the physical facility. There may well be other facilities on which the more critical data processing functions can be continued until the prime system is replaced with the result that the cost of the loss of the facility is only modestly greater than the replacement cost of all that destroyed. For this reason, it is recommended that loss of the ability to process be treated as a data security problem and taken into account when considering the impact of other problems on specific files.

When considering the problem of fire, bear in mind that fire can deprive the facility owner of services without destroying or in any way damaging the data processing complex itself. In high rise buildings, for example, severe fires on any floor below the facility, and, frequently, on any floor above, may disable that facility by depriving it of power, air conditioning, communications or elevators. Fire which destroys the supply of pre-printed paper forms can seriously inconvenience the operations and effectively cripple any function totally dependent upon those forms. It may well take longer to replace a destroyed supply of customized forms than it does to replace the hardware facility. It is necessary, to consider all aspects of each possible loss to fire.

It is important that data security and physical security definitions do not become confused. Data security problems are those which are contained by security measures in data processing hardware and software, and physical security problems are not just those which are contained by physical measures such as fire detection and quenching facilities. These interpretations do not yield a usable distinction and should be avoided.

THE RISK ANALYSIS TEAM

The composition of the team to perform the risk assessment is particularly critical to its success. The task cannot be done both quickly and well. It takes time. With even the best teams and a near optimum situation, experience has shown that the time required is about one month for each 2000 data sets or files considered. That figure is predicated on the assumption that the team is configured as suggested here and devotes the recommended amount of time to the task. Some experienced teams have reported an average rate as low as a file every 15 minutes of actual evaluation time. The proper consideration of the impact and probabilities required to complete the recommended procedure requires the assignment of well-informed, properly motivated people. This job *cannot be delegated to clerks as a routine task.* Because it takes good people, and in a large shop, quite a while, it is suggested that the best way to convene a risk analysis team is to agree that the people working on it will be required for only a half day per day with the other half spent at their normal duties. The alternative to this mode of operation, the full-time task force approach, seems to provide only a fast wind-up with a quick fade before a significant amount of work is completed.

The participants on the risk assessment team must include competent, senior representation from each of the following:

1. EPD operations management.
2. The department supported by or owning the data under consideration at the time.
3. The programmers responsible for support of that department, operation or function currently under consideration.
4. System programming, if the installation is large enough to have this as a separate function.
5. The internal audit function.
6. The department responsible for physical security. While these people may not be able to contribute a significant amount initially, their involvement usually results in their education in a way that may not be achievable by any other means.

In addition to the above, a *strong senior management commitment* to risk assessment is essential to its success. No amount of lower level concern will be truly effective unless everyone who has a role in achieving security believes that more senior management has sufficient commitment to this area. It is often difficult to convince senior management that they should be concerned without a quantitative expression of the problems as might be derived from the risk assessment. This situation leads to a chicken and egg problem. There is a need of senior management support to get a properly manned risk analysis team organized, but management may not be sufficiently concerned about data security until it sees the product of their labor.

Values of p

	1	2	3	4	5	6	7	8
1					$300	$ 3K	$ 30K	$300K
2				$300	3K	30K	300K	3M
3			$300	3K	30K	300K	3M	30M
4		$300	3K	30K	300K	3M	30M	300M
5	$300	3K	30K	300K	3M	30M	300M	
6	3K	30K	300K	3M	30M	300M		
7	30K	300K	3M	30M	300M			

Values of E $/year

Figure 3—Determination of Annualized Risk, E.

Thus, more than a modest amount of selling may be required to get a risk analysis program under way. Once it is functioning smoothly, there is usually continuing team dedication to the job as the team members sense the importance of the product—often as much through the surprises which they uncover as anything else.

BIBLIOGRAPHY

1. *Privacy Act of 1974,* Pub. L. No. 93-579, 88 Stat. 1896, December 31, 1974.
2. Privacy Act Implementation Guidelines and Responsibilities, Office of Management and Budget, Circular No. A-108, *Federal Register,* Vol. 40, No. 132, p. 28947, July 9, 1975.
3. Supplementary Guidance on Implementing the Privacy Act, Office of Management and Budget, *Federal Register,* Vol. 40, No. 234, p. 56741, December 4, 1975.
4. *British Computer Society Code of Good Practice,* National Computing Centre Ltd., London, England, April 1973.
5. Feistel, H., "Cryptography and Computer Privacy," *Scientific American,* Vol. 228, No. 5, May 1973.
6. Martin, J., *Security, Accuracy, and Privacy in Computer Systems,* Prentice-Hall, Englewood Cliffs, New Jersey, 1973.
7. Orceyre, M. J., "Data Security," *Journal of Chemical Information and Computer Sciences,* Vol. 15, No. 1, February 1975.
8. Parker, D. B., *Computer Abuse,* Stamford Research Institute, Menlo Park, California, Nov. 1973.
9. *Privacy in a Free Society, Roscue Pound American Trial Lawyers Foundation,* Cambridge, Massachusetts, June 1974.

Publications of the U.S. Department of Commerce, National Bureau of Standards:

1. Executive Guide to Computer Security.
2. NBS Special Publication 404, *Approaches to Privacy and Security in Computer Systems,* September, 1974.
3. NBS Technical Note 780, *Controlled Accessibility,* Bibliography, June, 1973.
4. NBS Technical Note 809, *Government Looks at Privacy and Security in Computer Systems,* February, 1974.
5. NBS Technical Note 827, *Controlled Accessibility Workshop Report,* May, 1974.
6. NBS Technical Note 876, *Exploring Privacy and Data Security Costs—A Summary of a Workshop,* August, 1975.
7. FIPS PUB 31, *Guidelines of Automatic Data Processing Physical Security and Risk Management,* June, 1974.
8. FIPS PUB 41, *Computer Security Guidelines for Implementing The Privacy Act of 1974,* May, 1975.
9. FIPS PUB 39, *Glossary of Terminology for Computer Systems Security,* to be published January 1976. Available as TG-15 Working Papers of 9/75.
10. Encryption Algorithm for Computer Data Protection: Federal Information Processing Standard, proposed, *Federal Register,* Vol. 40, No. 149, p. 32830, August 1, 1975.

Working Papers of Federal Information Processing Standards Task Group-15:

1. TG-15/24.1, *Index of Automated System Design Requirements as Derived from the OMB Privacy Act Implementation Guidelines,* August 12, 1975 (to be published as NBSTR).
2. TG-15/30, *Toward a Taxonomy of Computer Security Requirements for Federal Agencies,* by Alfred M. Pfaff.

Publications available from International Business Machines Corp., White Plains New York:

1. *Data Security and Data Processing,* Volumes 1-7, Joint Study by IBM Corp., Massachusetts Institute of Technology, TRW Systems, Inc., and the Management Information Division of the State of Illinois (G320-1370 through G320-1376).
2. *Considerations of Data Security in a Computer Environment* (G520-2169).
3. *Considerations of Physical Security in a Computer Environment* (G520-2700).
4. *42 Suggestions for Improving Security in Data Processing Operations* (G520-2797).
5. *The Fire and After the Fire* (G520-2741).
6. *Proceedings of the IBM Data Security Symposium,* April 1973 (G520-2838).
7. *Proceedings of the IBM Data Security Forum,* Sept. 1974 (G520-2965).
8. "OS/VS2 System Integrity," W. S. McPhee, *IBM Systems Journal,* Vol. 14, No. 3, 1975 (G321-0042).

Problem areas in computer security assessment*

by S. GLASEMAN, R. TURN and R. S. GAINES
The Rand Corporation
Santa Monica, California

ABSTRACT

An important problem area in providing security in computer systems is to avoid excessively costly and constraining security practices while providing an adequate level of security. In addition, there are problems in determining an appropriate level of investment in techniques and practices which enhance security and in the measurement of returns on those investments, i.e., to what degree is security improved by any given technique?

The resolution of these problems depends on the development of a capability for identifying and evaluating the risks of storing and processing sensitive data in imperfectly secure computing environments. This paper provides background information on security assessment, surveys recent work and the present status of computer security assessment, and identifies the research needed to move this field forward.

INTRODUCTION

One of the goals of contemporary computer science is the design and production of secure computer systems. Toward this end, current research is focused on designing secure ~~operating systems[1] and secure data base systems.[2] Consid~~erable progress is being made, even though certain underlying philosophical questions, such as the following, have not yet been satisfactorily answered: What is secureness in computer systems? How is it evident that a computer system is secure? Are there degrees of secureness and if so, how are they measured?

In the interim, however, it is necessary for the managers of operational computer systems to understand and assess the security situation in their systems. Are the security techniques employed adequate? Are the associated costs and operational constraints justifiable? What losses are likely should the security system fail? For example, one problem is the inability of current operating systems to provide multilevel security in resource-sharing computer facilities where different categories of sensitive national defense information are processed. In such cases it is a common practice to use "periods processing"—the computer is dedicated to processing information in only a single sensitivity category at a time. This results in decreased efficiency and throughput, and requires time-consuming procedures for changing between processing periods.

In order to provide some answers to the above day-to-day questions dealing with security, the use of the concept of risk, and the associated techniques of risk assessment and risk management, have been suggested and are currently being explored. In this paper we will discuss various approaches to what we will call security assessment in computer systems, examine the difficulties that arise, and use a conceptual model to suggest promising directions in the development of a security assessment methodology.

RELEVANT CONCEPTS

Since the concept of risk admits of a significant element of subjectivity, it is perhaps not useful to attempt a rigorous definition of it or of the terms associated with it. However, since a set of shared perceptions is necessary if communication is to occur, we will briefly discuss several contexts within which some of the terms apply, then describe our own views.

Management Science texts often deal with the concept of an "insurable" risk.[3] That is, business plans involve both certain knowledge and elements of risk. The ways in which these elements may be combined to more or less insure a favorable outcome depend on the relative contribution each makes to knowledge. A related view of risk is held in the insurance industry. Here, risk has principally to do with the uncertainty of loss, and the degree of risk is measured by the probable variation of actual experience from expected experience.[4] In each of these cases, empirical data is available against which to apply statistical tools in evaluating the risks involved. One of the important constraints on our ability to assess security is the lack of such data.

Another example comes from Systems Engineering, which considers risk as associated with situations in which the set

* The preparation of this paper was supported, in part, by the National Science Foundation Grant No. MCS 76-00720. However, any views or conclusions contained in this paper should not be interpreted as representing the official position or policy of the sponsor or The Rand Corporation.

of possible outcomes is completely known, and the probability of each outcome is calculable.[5] In terms of computer security risk, we will rarely be able to claim "complete" knowledge of potential losses (outcomes), or to calculate all probabilities. However, we argue here that we can do much better than we now do in identifying the potentials for certain types of losses by the use of informed judgment based on close examination of all the elements which contribute to the potential for loss.

Two types of risks are recognized in economic theory: (1) speculative risks where there is an uncertainty whether an outcome may entail a profit, or a loss, or neither; and (2) pure risks where only losses are possible (but it is uncertain whether or not they will actually occur). In the context of computer security, we are dealing with pure risks. Gahin[6] has formulated an elaborate theory of pure risk management in the business firm. We feel that Gahin's ideas merit further study to determine their potential application in assessing and managing computer security risks. To motivate such study, we present a very brief outline of his work.

Gahin defines risk, R, as the maximum potential loss (to the subject of risk) due to a specific event (a specific cause of loss). That is, R is a loss that could be exceeded with a probability ξ, and as ξ approaches 0, the value at risk approaches the maximum possible loss. As defined, risk would be measured in cost units, such as dollars, but could also be expressed qualitatively. Associated with risk in Gahin's theory are a risk function, a security function, and a methodology for optimal selection of risk control devices.

The risk function is expressed as $R=f(V,X,N)$ and involves three variables: (1) the total value exposure, V, is the total economic value that could possibly be lost or destroyed by the same event; (2) the loss ratio, X, gives the statistics from previous experience with this particular risk event, and shows how much of the value exposed was lost in similar events in the past; and (3) the number of independent units of exposure, N, subject to the same type of risk (e.g., the number of warehouses that collectively contain the total inventory of a business firm) which contain parts of the total value. In general, R is directly proportional to exposure and to loss ratio, and inversely proportional to amount exposed.

Gahin refers to security as the outcome of protecting the firm against pure risks in terms of the reduction or elimination of the risk. He defines a security function, $S=f(A,L,D,I,F)$, also measured in units of cost (dollars), which is used to select among several risk control techniques. After an application of S, the new value of risk, R', is expressed as $R'=R-S$.

The categories of risk control techniques (the parameters in the security function) are: (1) risk avoidance, A, which is aimed at reducing the total value exposure, V, by avoiding situations or activities that involve the risk; (2) loss prevention, L, involving techniques which can reduce the frequency and/or severity of loss occurrences (components of the loss ratio, X); (3) distribution of the value at risk, V, over more independent exposure units by increasing N; (4) insurance, I, which is the transfer of the burden of loss partly or totally to another party by paying a premium to that party; and (5) self-insurance in the form of establishing a reserve fund, F, for offsetting possible losses. Gahin suggests several strategies (too lengthly to summarize here) for using the security function to select an optimal mix of risk control techniques. Underlying these techniques, however, is the assumption that for each risk control device its cost and its risk-reducing capability are known. As we will discuss below, determination of this information is a major problem.

For the purposes of the present discussion we will consider a computer system to consist of a physical installation, hardware, software, procedures, and communication links with remotely located satellite computers or terminals. Any of these components, or any combination of them, may be considered as subject to risk. That is, each represents a certain value which is subject to loss. Any weakness in the design, implementation or operation of a computer system must be regarded as a vulnerability which, if exploited (singly or in combination with other weaknesses), would represent a threat to the information entrusted to the system. Whether or not such threats will actually materialize, that is, whether vulnerabilities will be exploited, either accidentally or intentionally, and whether or not they will succeed in causing losses are the basic uncertainties in providing security in computer systems.

One objective of security assessment is to estimate the risk, or likelihood, that a loss will in fact occur as a direct result of a given threat. Improved procedures for assessing risk should also result in better means than are now available for estimating the magnitudes of the various losses that are possible.

These two major targets of security assessment—likelihood of loss, and magnitude of loss—generate requirements for a broad range of information. So, we can view security assessment as an information gathering and analysis procedure which will furnish the basis for more informed and more effective decision-making with respect to computer security alternatives. Suggesting the kind of information that is required to make such decisions is the main thrust of this paper.

RECENT RESEARCH

The need to satisfy operational requirements in an environment providing less than complete security implies the acceptance of a certain amount of risk, and we must have a sound understanding of the elements of such risk before we can make value judgments as to (1) appropriate levels of investment in techniques for risk reduction, and (2) evaluation of the increment of risk reduction associated with such techniques.

The greatest progress in using risk assessment for evaluating computer security appears to have been made in physical protection of computer facilities and equipment against natural disasters such as fire, flood, and the like, as well as against theft and sabotage.[7,8] This is not surprising since (1) experience from the insurance industry is readily applicable, (2) values of exposed equipment are relatively simple to determine, and (3) ample empirical data is available for determining probabilities of loss due to various threats. The principal document in this area, FIPS PUB 31[7] equates risk with expected loss measured in dollars (expected loss = value of the exposed asset probability of occurrence of the threat), and gives guidelines for determining the required values and probabilities. This appraoch appears quite adequate and, hence, we will refrain in this paper from further discussion of risks due to natural disasters or physical theft.

Risks to data and programs in a computer system are much more difficult to determine. There is very little experience in determining the value of exposed data files or programs, not all threats can be identified, and threat occurrences tend to be highly uncertain. Most importantly, empirical data about losses incurred by existing computer systems is virtually nonexistent. Without data, attempts to evaluate system security must remain completely subjective. This point will be elaborated in the next section.

Thus far, two approaches to security assessment have been proposed: the Illinois treatment of the economics of security,[9] and a risk assessment approach proposed by Courtney.[10] Courtney's approach, or similar approaches developed independently, have been included in several recent books on computer security.[11–14]

Using the Illinois approach, the total expected cost (in dollars/year) in using the security system k, TEC(k), is defined as: TEC(k)=X(k)+Y(k), where X(k) is the cost (dollars/year) to install and operate the safeguards comprising the security system k, and Y(k) is the expected loss (also in dollars/year) due to exposure if security system k is used. The loss, Y(k), is found by summing, over all possible threats (defined as access routes to the protected assets), the products of exposed value and the probability of safeguard failure. If several safeguards must fail simultaneously in order to cause a loss, a product of the individual failure probabilities will be used. However, major assumptions are required involving knowledge of all the possible access paths and the probabilities of failure of the safeguards that block these paths. The perception and demonstration that concepts from economics can be applied to assessment of risks in computer systems is an interesting one. However, the assumptions required to enable application of this approach in any but the crudest exercise beg the major issues involved in security assessment.

The thesis in Courtney's method is that risk managers seek to obtain more precise results than can be justified. He suggests assigning order of magnitude estimates of value to the assets at risk, and makes correspondingly rough guesses at the probabilities that any of the threats will actually result in loss. The emphasis is on relative magnitudes of risk, and its reduction by additional safeguards, rather than on actual estimates of future losses and costs. Furthermore, Courtney limits threats to six possible classes—destruction, modification, or disclosure of data in the system, either accidentally or intentionally. Thus, there would be no need to subject each possible threat to detailed analyses of the associated vulnerabilities and their exploitation.

In line with this reasoning, Courtney expresses risk as estimated loss in dollars per year, E, and establishes scales for estimated exposed value and postulated frequency of occurrence, v and p, respectively, that represent powers of ten: $E=10^{v}\ 10^{p-D}/D$, where D is chosen to allow expression of E on a yearly basis (as expressed in number of working days). For example, if three years represents 1000 working days, D=3. The scale for p is so chosen that p=3 represents the postulated frequency of one event per year. To illustrate, if the exposed value of a data file is estimated to be three thousand dollars (v=3), and if the expected frequency of events is once every 30 years (p=2), then the exposed value per year, E=$33. However, if the expected frequency of loss is once every 100 days (p=4), then E=$3333.

Given the estimated yearly exposure, E, the next step (as in other, similar, approaches) is to decide on a course of action. According to Courtney there are three options: (1) tolerate the risk, (2) lower the estimated loss by implementing measures that result in less exposure, and (3) lower the probability of loss occurring by implementing protective measures that cost less than E. A fourth option might be to cover the residual exposure by insurance. However, Courtney maintains that this is not a real option—while it soothes the loss after it has occurred, it does not necessarily lessen the desirability of minimizing the risk by other measures.

Here again, much is based on rough estimates of resource value, threats, and loss probabilities. The whole utility of this order-of-magnitude approach depends on these estimates. If they reflect no incremental improvement to information accuracy, this technique can claim to provide no increment to our ability to assess risks. Also lacking in Courtney's approach, at least in published versions, is a methodology for choosing risk-reducing measures and estimating their impacts on the estimated exposure, E. This is, as we will discuss below, one of the major difficulties in existing risk assessment approaches.

In addition to the above, the issue of security assessment has appeared in several other recent studies.[15–21] They have attempted to establish broad principles for computer security, tested and evaluated the security provided by existing operating systems, and developed techniques for identification of security-related design flaws in operating systems. However, for the most part, these efforts have not involved basic research aimed at understanding the components of risk, and they have not recognized that progress in security assessment is a prerequisite for attainment of increased security in currently operational systems.

A CRITIQUE

Each of the two major studies discussed earlier has, in effect, described a model of a computer system from a security point of view. These models relate to various elements that appear to contribute to the security of a system in such a way as to allow one to make quantitative judgments about the security offered by a system, or the amount of money that should be invested in enhancing the security of a system. There are shortcomings in both models which make it doubtful that they can be used to make serious quantitative judgments concerning how much security one can achieve or what level of investment should be made. There are several reasons for this which we will now review.

The first is that we lack a sufficient understanding and a sufficient set of facts on which to base the formulation of reasonable models of the security of computer systems. As a consequence, if we develop models which purport to describe computer security there are no means for validating any one particular model or choosing between several different models. Suppose that, nevertheless, we adopt a particular model. An additional difficulty then arises: We cannot apply the model in practice. The reason is that the models assume that we are able to supply values for the parameters of the model which, in fact, we are not able to supply. For example, in Courtney's model one is expected to estimate, for each system or data set that is to be protected, the probability of intentional disclosure or modification. In this simple phrase is included almost the whole of the computer security problem. Courtney's approach suggests that we understand the kinds of attacks that could be made on a system, the probability that someone will try, the probability that they will succeed if they do try, and that we know how to combine all of this information into a meaningful judgment of the probabilities associated with incurring losses (themselves related to resource evaluation) through various means of attack. Courtney points out that a high degree of precision in such probability estimates is not now possible, and, further, that it is not necessary. However, at this time security assessment is in its infancy, and the techniques currently proposed are, at best, formalizations of the crude, intuitive methods that have been in use informally for several years. Progress in security assessment depends primarily on the use of more precise information. Failure to pursue the goal of greater precision amounts to an agreement that today's assessment techniques are sufficient for our needs. Few people knowledgeable in computer security would feel comfortable with such a view.

It appears that what both the Illinois and Courtney approaches have done is to try and describe what we would do if we really had all the knowledge and information about system vulnerabilities, people's intentions regarding those vulnerabilities, and exact dollar values concerning the losses that we could expect if particular attacks should occur. It is our view that this is the simplest part of the problem. The hard part of security assessment is in the identification and analysis of the full range of vulnerabilities and threats that might occur in systems. This is a very complex subject and only a subset of the vulnerabilities in systems has, as yet, received very careful attention.

The issues which have received attention fall into two categories. The first is operating system vulnerabilities. The second area is the physical security of computer centers. The range of topics yet to be considered is large and includes some very serious problems to which little attention has yet been paid. These topics include: (1) the vulnerability of the hardware itself, (2) the possible ways which correctly functioning operating systems may nevertheless be exploited, and (3) the vulnerabilities in procedures for various operations in a computer center, including recovery from crash, backup of files, initiation of the system, authorization of new users for the system, and granting of new access rights for existing users for the system.

As we analyze in depth the particular threats that can be discovered in the context of particular existing systems, we may also begin to consider some of the measures one might take to block those threats. We do not wish to suggest that by this route one may discover all of the measures which should be taken to improve the security of a system. However, at present our list of security enhancements for systems is exceedingly small and any additions to that list would be a welcome fringe benefit from such a study.

IMPORTANT RESEARCH ISSUES

If we do not believe that the models currently being offered for security assessment are satisfactory, how may we proceed? We feel that it is appropriate to examine the assessment question a good deal more carefully in the context of one or more existing systems so as to acquire sufficient data on which to base the construction of models of computer systems from a security point of view. If we hope ever to develop valid quantitative methods for assessing the security of a system, the risk that is being exposed, and the probability of loss on which to base estimates of the dollar amounts that ought to be spent in enhancing security, then we first need a much more detailed analysis of the elements that contribute to security or to the lack thereof in computer systems. In fact, we prefer to leave it as an open question whether or not a quantitative assessment methodology can ever be developed.

Thus far, the development of requirements for technically enhancing the security of existing systems has proceeded in the absence of a clear understanding of what is being protected from whom, and at what cost. One important inference that may be drawn from the work discussed earlier is that significant progress in security assessment is constrained primarily by a lack of specific information in most of the areas which are associated with the concept of risk. One view of the interrelationships among these areas is shown in Figure 1. The figure is not intended as a complete specification; it is provided to indicate the complexity of the field. Those knowledgeable about the current state of computer security will recognize that many of the relationships shown in this figure are, at best, poorly understood. Nevertheless, even such preliminary and incomplete attempts to structure the security assessment field makes clear the need for a deeper examination of a number of important areas. They include:

a. the specification of system assets to be protected, and an estimation of their value.
b. the identification of system vulnerabilities, including, but not limited to, operating system vulnerabilities,

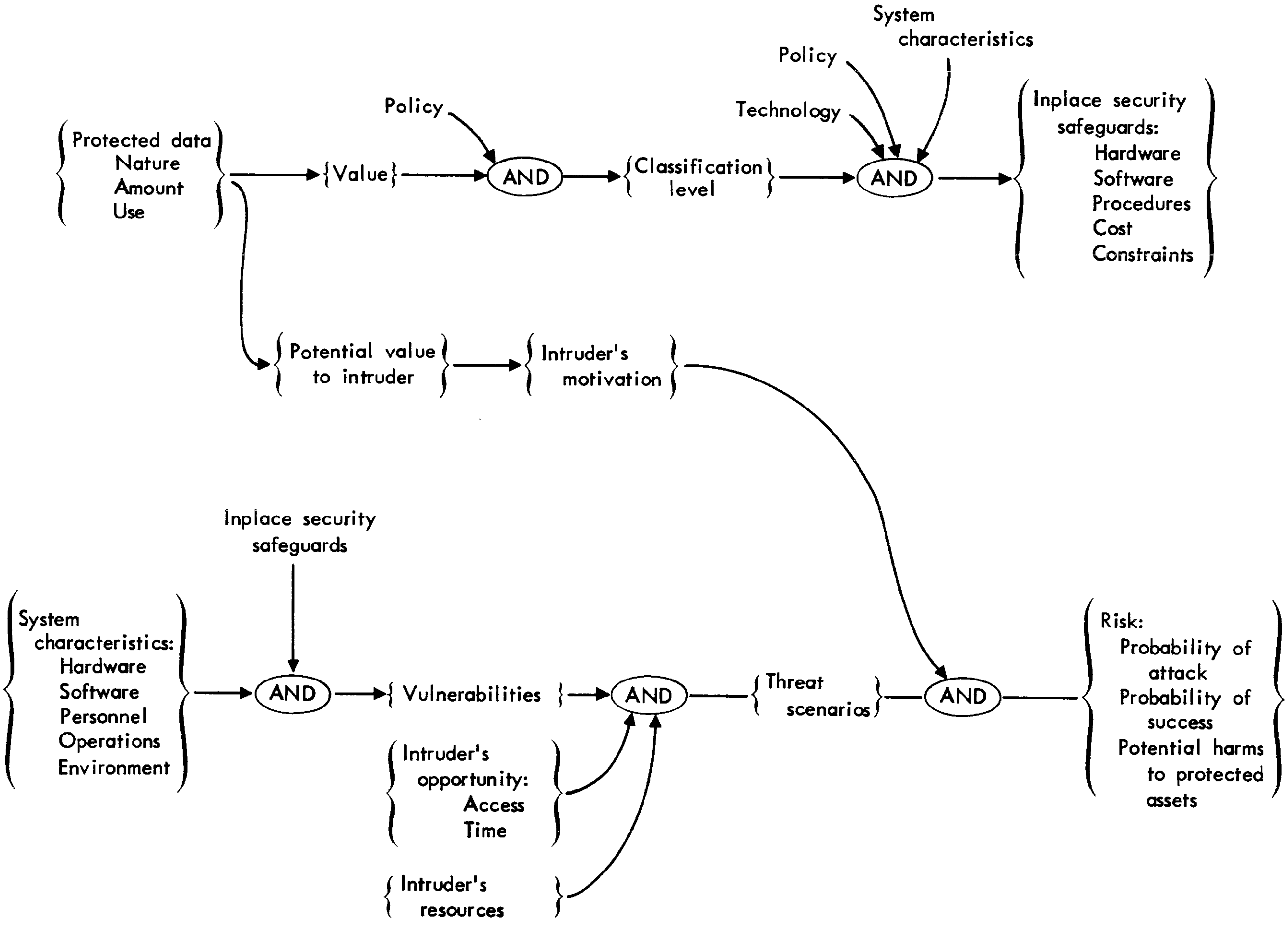


Figure 1—Some interrelationships among certain elements of risk

c. an estimation of the relative degree of exploitability associated with each vulnerability, or set of vulnerabilities,
d. identification of the resources required by an intruder for various attacks, and
e. the identification and specification of explicit threats to sensitive information.

Since these elements of risk and their interrelationships are imperfectly known, research is needed if we are to progress beyond present limited abilities to assess security. Each of these areas is briefly discussed below in order to call attention to some of the important and basic research questions yet to be attacked.

Protected assets

Any approach to security assessment requires estimation of the value of the assets to be protected. Prior to this we must identify those characteristics of the assets which establish the need for protection and affect the value estimation. What are these characteristics and how do they influence protection decisions? This question requires further research, especially as it applies to data stored and manipulated by computers, and to applications and systems software. One such characteristic is clearly the likelihood that a particular asset would be singled out as a target. A tentative list of other characteristics that may be relevant includes the following:

a. form of the data (narrative text, numeric information, etc.),
b. data classification,
c. file size, and fraction of sensitive data in the unit of measure (e.g., record, file, etc.),
d. frequency and duration of use,
e. duration of sensitivity,
f. dependence on and linkages with other data bases, and
g. storage media used.

It is likely that some of this information is already employed in rudimentary assessment procedures. For instance (see e, above), information which is valuable for only a short time is more readily entrusted to a computer than is data of continuing importance. A detailed examination of such characteristics will permit their more rigorous application in assessment procedures, and should aid the development of standardized assessment techniques.

In what way does the concept of value apply to those assets subject to protection? One major problem in assigning value is determination of the unit of measurement. The tendency is to value in terms of dollars; for example, costs are associated with developing alternatives to a compromised plan, and with re-creating a contaminated data file. While it is true that economic costs may be associated with several aspects of protected resources, this is only a part of the problem—other costs are involved which go beyond technical considerations. Dollar value is hard to assign, for example, to national security information or to personal information—a variety of intangible costs are involved. Such intangibles are usually expressed in terms of the degree of harm or damage that might result from a security compromise. But this is a very fuzzy area; what are the various possible harms? How does one determine their relative importance? One demonstration of the ambiguity in this area is found in the official criteria for the major national security information classification levels:

- Top Secret ". . . unauthorized disclosure of which could reasonably be expected to cause exceptionally severe damage to national security."
- Secret ". . . unauthorized disclosure of which could reasonably be expected to cause serious damage to national security."
- Confidential ". . . unauthorized disclosure of which could reasonably be expected to cause damage to the national security."

Such expressions clearly require individual and subjective interpretation before classification can proceed. To assign value to sensitive assets, we must be able to better specify the losses which might result from their compromise. In addition to its utility for the assessment of security, more rigorous specification of values might provide a basis for reevaluating the current scheme for assigning classification categories to sensitive information.

Vulnerabilities

What are the vulnerabilities associated with the protected assets? Penetration studies have produced convincing demonstrations of the vulnerabilities of present day operating systems, and fairly complete identification and categorization of design flaws and implementation errors which seem to produce such vulnerability. This focus is traceable to the quest for a perfectly secure ADP facility, for which the prime requirement is a flawless operating system. But, other types of vulnerabilities exist, and must also be considered in any effort to estimate the degree of security provided by a particular ADP facility. For example, hardware failures can result in data misroutings and consequent inadvertent disclosures of sensitive information, and in contamination of files by inappropriate insertion or omission of data.

Another aspect of vulnerabilities is that of ease of exploitation; some vulnerabilities are easier to exploit than others. Scavenging for residual information in systems which do not clear memory blocks upon deallocation, and trial and error search for passwords are examples of easily exploited vulnerabilities. Readily accessible, unhardened, computer-to-terminal communication lines represent another vulnerability, easy to exploit through wiretapping, but harder to fix—a cryptographic system must be installed.

However, there are other vulnerabilities which require much more effort to exploit. Vulnerabilities based on the time-dependent characteristics of asynchronous processing are in this category. If it were possible to weight all vulnerabilities on the basis of exploitability, the protector would be in a much better position than he is now to determine risks and choose strategies to allocate the funds for implementing security systems. Clearly, such funds are limited and some vulnerabilities, e.g., those rooted in system design, cannot be eliminated for technical or financial reasons based on requirements for major changes in software or hardware.

It is important to distinguish between an intruder's money and time costs. In some cases money costs might be considered immaterial, while time costs might be quite important, and, in fact, limiting with respect to the available options. If we assume an intruder will attack via the most exploitable vulnerability which will satisfy his needs, then the majority of limited protector funds might be allocated, not to potentially fruitless attempts to eliminate such vulnerabilities, but to dramatically increasing the time required to exploit them; thereby increasing the intruder's time costs, and the real and perceived likelihood of detection.

Resources

What resources can an intruder bring to bear against a system? How should these be taken into account in evaluating the plausibility of threat scenarios, estimating risks, and designing more secure systems?

Depending upon an intruder's goals and motivation, and on system characteristics, resources are of varying importance. For example, the disgruntled employee may or may not have the technical expertise sufficient to carry out his aims, but may have ample time to acquire it. On the other hand, an agent of a hostile government might have sufficient expertise, but access to the target system might be time-limited.

Some of the basic resources needed by potential intruders are: funding, time, technology, expertise, and opportunity. These resources combine in various ways to contribute to the probabilities associated with overcoming existing safeguards. Furthermore, knowledge of the combinations of resources associated with different types of intruders and vulnerabilities helps the protector to identify and deal with the threats represented by those combinations.

For example, the time required to exploit a vulnerability is sometimes a more important consideration than the amount of other resources needed—the more time required to successfully complete an attack, the more likely it is that the intrusion will be detected and thwarted. This is a major principle in providing physical security in other environments. It may also be applied to computer systems, even though the time scale may be different, if the system can be adequately instrumented for detecting questionable activities. This is further evidence that increasing the time required to exploit vulnerabilities which are too costly to eliminate may be a plausible protection option.

Threats

What threats are associated with the protected resources? The several dimensions of this area are virtually unexplored, even in the commercial world. For any threat to exist, several conditions must be met; they include, but are not limited to, the following.

a. There must exist an exploitable vulnerability.
b. Opportunity (whether for an intrusion, or for an accident) must exist.
c. Other resources must be available for intentional exploitation (funding, technology, expertise, etc.), and some failure process must be active for accidental exploitation.

Moreover, threats to a system posed by a malicious intruder may involve the following additional characteristics which strongly influence whether or not a possible threat becomes probable:

d. Some information in the system will be highly valued by the intruder.
e. Other sources for that information will be less attractive.
f. The intruder's potential benefits outweigh the losses he would suffer if his attempts fail.
g. Plans for gaining access exist, or can be generated.

How do these characteristics combine to produce plausible threat scenarios? How can these be used to estimate the probability that threats will be carried out? Research is needed to develop answers to such questions.

Taken together, the components of threat exhibit a high degree of interdependence. Therefore, it is necessary in security assessment to deal with their interactions, in addition to their individual contributions. Threats also depend, to some degree, on the perceived value of protected assets, and on opportunity, available resources, etc. The relative value of resources is sensitive to who is doing the evaluation—which in turn depends upon someone's perception of vulnerabilities, opportunity, and so on. Finally, the risks faced by any system can be expressed in terms of resource evaluation, intruder motivation, opportunity, other resources, and existing vulnerabilities. It is clear that the understanding of each specific issue discussed above depends to some degree on understanding the others.

CONCLUSION

In the foregoing we have listed a number of topics which should be explored in greater depth. The work proposed is basic. It provides the necessary background for judging the utility and cost-effectiveness of all other security work. It may also permit new insights into the kinds of protection mechanisms and procedures which will be beneficial in the imperfect real world.

Such topics might be studied in vacuo, simply by careful reasoning about the issues involved. However, we suggest that it might be of substantial benefit to consider these questions in the context of one or more existing systems which would serve as a source of data about the actual problems which arise in conducting a risk assessment related to the security of a "live system." One of the important questions in risk assessment has to do with which problems arise in practice, in contrast to the problems one can imagine in theory. Moreover, it is premature to search now for a global security assessment methodology. The development of such a general capability is constrained by a lack of exactly the kind of data that will become available only as a result of focused, individual efforts at assessing the security characteristics associated with specific systems.

We suggest that there are two major requirements for progress in security assessment:

(1) increased research emphasis aimed at the development of a better understanding of the informational elements of security assessment;
(2) experience, at the level of individual computer installations, in the application of a broader and more accurate information base to the assessment of computer security.

Improved information, and experience in applying it in various ways should provide the data needed to refine the focus of efforts to develop more general security assessment techniques.

There are no shortcuts, no "quick-and-dirty" solutions that will significantly enhance either our ability to provide a degree of security to sensitive resources, or our ability to accurately measure the risks involved in operating in an imperfectly secure environment. Rather, the whole area must be viewed as it actually is—a complex set of issues that must be put into broad perspective.

REFERENCES

1. Saltzer, J. H. and M. D. Schroeder, "The Protection of Information in Computer Systems,"*Proceedings of the IEEE,* Vol. 63, No. 9, September 1975, pp. 1278–1308.

2. Hsiao, D. K. and R. I. Baum, "Information Secure Systems," *Advances in Computers*, Vol. 14, Academic Press, N. Y., 1976, pp. 231-272.
3. Koonz, H. and C. O'Donnell, *Management: A Systems and Contingency Analysis of Managerial Functions*, (Sixth Ed.), McGraw-Hill Book Company, New York, 1976.
4. Mehr, R. I. and E. Cammack, *Principles of Insurance*, Richard D. Irwin, Inc., Homewood, Ill., 1957.
5. Hal, A. D., *A Methodology for Systems Engineering*, D. Van Nostrand Company, Inc., Princeton, New Jersey, 1962.
6. Gahin, M., *A Theory of Pure Risk Management in the Business Firm*, Ph.D. Thesis, University of Wisconsin, University Microfilms, Ann Arbor, Michigan, 1966.
7. *Guidelines for Automatic Data Processing Physical Security and Risk Management*, FIPS PUB 31, National Bureau of Standards, Washington, D.C., June 1974.
8. *The Considerations of Physical Security in a Computer Environment*, Document G520-2700-0, IBM Corporation, White Plains, N.Y., October 1972.
9. *Data Security and Data Processing Volume 3 Part 1 State of Illinois: Executive Overview*, Document G320-1372-0, IBM Corporation, White Plains, New York, June 1974.
10. Courtney, R. H., Jr., *Security Risk Assessment in Electronic Data Processing Systems*, IBM Systems Research Center, New York, December 1975.
11. Martin, J., *Security, Accuracy, and Privacy in Computer Systems*, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1973.
12. Hoyt, D. B. (Ed), *Computer Security Handbook*, Macmillian Information, New York, 1973.
13. *Where Next for Computer Security?* The National Computer Centre, Ltd., Manchester, 1974.
14. *An Executive's Guide to Data Security—A Translation from an IBM Svenska AB Publication*, Document G320-5647-0, IBM Corporation, White Plains, N.Y., October 1975.
15. Bushkin, A. A., *A Framework for Computer Security*, TM-WD-5733/000/00, Systems Development Corporation, McLean, Va., March 1974.
16. Abbott, R. P., et al, *Security Analysis and Enhancement of Computer Operating Systems*, NBSIR 76-1041, National Bureau of Standards, Washington, D.C., April 1976.
17. Carlstedt, J., R. Bisbey, and G. Popek, *Pattern Directed Protection Evaluation*, ISI/RR-75-31, USC/Information Sciences Institute, Marina Del Rey, Ca. June 1975.
18. Bisbey, R., et al, *Data Dependency Analysis*, ISI/RR-76-45, USC/Information Sciences Institute, Marina Del Rey, Ca., February 1976.
19. Anderson, J. P., *Computer Technology Planning Study, Volume 1*, ESD-TR-73-51, J. P. Anderson & Co., Fort Washington, Pa., 1973.
20. Jacobson, R. V., "Risk Analysis in Planning for Physical Security," in *Approaches to Privacy and Security in Computer Systems*, NBS Special Publication 404, National Bureau of Standards, Washington, D.C., March 1974, pp. 54-55.
21. Peters, B., *Operational Requirements for WWMCCS ADP Security*, TM-WD-5733/001/01, System Development Corporation, McLean, Va., June 1974.

Computer based information systems for the small firm—Why? cost? caveats, functional needs, contracts

by FREDERICK FRANCIS NEWPECK
University of New Mexico
Albuquerque, New Mexico

ABSTRACT

The following article presents advice to the small business considering automation of information flows. Topics discussed are, why and when to automate, the costs of automation, how to avoid failure, how to specify functional needs, and what to expect from the vendor contract.

WHY COMPUTERIZE A MANUAL SYSTEM?

If a small business has annual sales of $250,000, it should consider the purchase of a small business computer to automate some of its information flows.

A computer system would allow a seasonal business requiring huge amounts of clerical effort in a short period of time to expand and contract while maintaining a small clerical staff. Volume of data input may vary, but input is only a small part of the total clerical effort required to process data in a manual system.

To reduce the complexity of manual accounting systems, a computer can integrate accounts payable, accounts receivable, general ledger, payroll, and job cost accounting in a system which provides one-time data entry. When data is entered, the system automatically updates the necessary files and ledgers. Accuracy and integrity of the firm's accounting system is enhanced by eliminating human processing of data.

To control day-to-day business, management may need more accurate and timely information. Once data is accurately entered into a computing system, it is processed with 100 percent accuracy at electronic speeds. Furthermore, the computer can be programmed to display the same data in a variety of ways to give the manager a better current perspective. For example, if a summary report indicates a problem area, a preprogrammed detail report can provide analysis of the problem area. Manual systems could provide the same information, but at the cost of high clerical salaries and precious time. Hence, small business managers tend to "fly by the seat of their pants," making decisions on "feel" and soft data. A computer can rationalize management.

A computer can provide small business management with the sophisticated management information available to executives of the large firm: cost and performance analyses by product activity, area or individual; controls over inventory that significantly reduce the number of dollars tied up in it; and analyses of sales, credit sales and purchases which define cash needs and cash availability and provide the basis for the wise use of credit. In short, a computer-based system can be a competitive advantage. Ongoing cost analysis aids pricing decisions and better control over inventory, and cash management can reduce overhead expenses.

THE COST OF COMPUTER TECHNOLOGY

A small business can now afford a computer because the cost of computers have been dropping at a 35 percent annual rate for the past 15 years. The main parts of a computer can be manufactured now for less than $100. (MITS, an Albuquerque, New Mexico computer manufacturer, sells computer kits to hobbyists for approximately $400.)[1] Experts predict that by 1980 a computer equivalent in power to the IBM 7090, which sold for approximately $2 million in 1965, will cost $1 to manufacture! The implications are startling. Computers are used in ranges, watches, home environmental control systems and automobile fuel injection systems.[2] The implications for factory automation are obvious.

But, if computers are so cheap, why do small business computers cost between $5,000 and $90,000? (A typical small business system costs from 20 to 40 thousand dollars.) The price of components of the system—data entry terminals, printers, and storage devices—has not been dropping as rapidly as the processor. Substantial price decreases occur only when manufacturing processes are totally automated. Although the computer can be manufactured on a single integrated circuit chip which eliminates hand wiring, manufacture of computer terminals still requires human handling at almost every step. The net result is that the cost of computing systems will continue to decline at a 10-20 percent compounded rate through 1980.

However, the average cost of a purchased small business system has not decreased. The personnel costs of writing programs to tell the computer what to do have been increasing. And now, customers are buying more sophisticated peripherals and computer storage so that the system does more things faster. In fact, integrated business systems allowing small business management to automate information flows in a functionally integrated, on-line environment have only emerged within the last year and have yet to be announced by some vendors. These systems allow a clerk to enter the data once, the data is then made available to all hierarchical and functional levels of the organization. One result is that the functions of marketing, finance and production make decisions based on the same data. Hence, the classic goal conflict between production and marketing is eliminated. Information provides a synthesis of the goal of producing to meet customer needs at a competitive price and the goal of producing efficiently.* Hence, merely providing information to disparate functions of the organization may eliminate the need for managerial control and provide more freedom for middle level managers. Information itself is control and the computer can provide a freer organizational climate in which participants direct their own behaviors to a greater extent. Hardware to provide this organizational climate is readily available, however, the integrated software systems are in their infancy of development and availability.

THE PURPOSE OF AN AUTOMATED INFORMATION SYSTEM

Let us now examine how the formal information system develops in a corporation. When created, the business probably had one or two employees, the president and a close friend. The president carried accounts receivable, accounts payable, and inventory data in his head. As the firm grew, more people were added to handle the workload. The resulting specialization and fragmentation of the firm's memory among individuals require formal communication flows and reports to create the situation that once existed when one person possessed all the information. As the business continues to grow, data storage becomes further fragmented into functional areas as the number of specialized functions increases and the number of people in each specialty increases also. Under these circumstances, the problem is how to enable the firm to react to its environment as a single entity.

The answer lies in a computer-based information system in which the data is sensed and collected once and entered into a single database which can be accessed by any business function to obtain information. Because all functions base decisions on the same data, disparate functions like marketing and production are coordinated. Hence, total system optimization replaces suboptimization.

In summary, the purpose of an information system is to provide information to the management system for decision making. The purpose of automation is to increase the efficiency and effectiveness of communication channels within the business so that once again the firm can react to its environment as a single organism.

WHY DO COMPUTER BASED INFORMATION SYSTEMS FAIL?

The objectives for automation are simple and management is aware of the potential for coordinated operational control. Why then do some computer based information systems fail miserably while others are successful? One reason is a management structure that cannot control its current environment; another is the lack of top management involvement with the computer based system; and a third reason is software. Before the computer based information system can be successful, the management system must be well structured. A small company looking at automation for the first time usually has been growing at a 100-300 percent annual rate. Growth results in confusion within the management system that can only be resolved by restructuring management responsibilities. If the computer is expected to solve management problems, the result will most certainly be failure. Second, the prerogative to manage the system containing the valuable, irreplaceable data resource of the firm must never be delegated. Without top management involvement at every stage of implementation, the computerized system will fail and possibly the business itself.

Management participation must be educated. Management may already be expert in the selection, acquisition and use of machinery for its manufacturing or service business, but managers typically do not possess the same degree of competency with computers. Small business managers are ignorant of computer technology because they were not trained in the universities and/or because they have been too busy making money to educate themselves in the rapidly changing computer technology. Aggressive action often is not taken in an area where the small business manager feels insecure. The small business manager needs to become conversant in computer technology and its management to avoid the typical pitfalls and provide rational computer based information system management leadership. In many cases, the burden of responsibility for placing the nerve center of the firm on a computer is delegated to a technician. It is unfair to place such a burden on a technician and it is dangerous to the very existence of the firm that so much power and authority resides with a single person who is not intimately tied to the success of the firm.

A third and critical area is software. Very often a small business manager requires a computer system to conform to his firm's established procedures rather than change the firm's procedures to fit a preprogrammed application package. The process of finding a software package which feels comfortable to a firm with minimum change is the critical issue. Small businesses should not undertake to write their

* If machines are controlled by computers, production to order will be as efficient as continuous production runs because machine set up will be via changing the program rather than costly mechanical alterations.

own software systems. In general, they do not possess the software expertise and the tremendous financial resources that go into writing and documenting their system. Specialized software often ties the firm to hardware and the people who wrote the software. One small firm nearly went bankrupt when its data processing manager died in an accident. No one understood the programs he wrote which were undocumented and no one knew how to run the software on the computer! Standard software would have precluded the problem, since a large group of people would understand the operation of the software system. In general it is best if a small firm uses standard software packages. Modifications to the software should be made only after the firm has used the package for three months, often needed changes prove to be niceties the firm can do without.

COMPUTER ACQUISITION

Nothing can replace the computer educated, articulate manager in the computer acquisition process. However, a measure of security may be achieved by hiring a consultant knowledgeable in the small computer field to assist in the acquisition procedure. Computer sales people are, after all, trying to sell their product. They are an excellent source of useful information, but aggressive sales people will waste valuable time and often appeal to emotional rather than technical issues. A knowledgeable consultant, someone who is in the market every day, can serve as an effective buffer for the small business manager. The final equipment decision and responsibility must lie with the manager. He alone is responsible for the profitability of the firm. An incorrect decision could cost a small firm an amount of money far exceeding the cost of the hardware. A construction firm with which the author is familiar lost ¼ million dollars the year it purchased a computer; the next year it returned to its manual system and has been profitable since. Another firm, a manufacturing company automated its accounting functions and later was unable to access its accounting data for government auditors. The result was a delay in payment on government contracts and a substantial ~~investment of the company president's time over a period~~ of six months. However, if a company uses the following acquisition procedures, the threat of loss will be minimized:

1. If there is no computer expert in your firm, hire a consultant. The consultant should be able to provide you with references, he should have a degree in business administration with a concentration in information systems or computer science coupled with substantial real world experience. The consultant should be able to understand your business problems and be an expert in the automation of business systems.
2. Generate functional specifications for your firm. These should include (1) the system objective, (2) a brief description of what is to be automated, and (3) parameters of the business, such as numbers of customers, inventory items, order volumes, etc. Examples of functional specifications tailored to the needs of specific firms appear in Appendix A.
3. The vendor should be asked to comment on (1) the number of similar systems they have installed and their length of operation, (2) their maintenance capability, and (3) the warranties and terms of purchase or lease.
4. Once the vendors have been narrowed to three to five qualified bidders, visit operating sites similar to the proposed system. During these visits ask questions in the following areas: (1) customer satisfaction, (2) installation delays, (3) length of installation operations, (4) major difficulties during installation, (5) system reliability, (6) service level, (7) modifications to system, and (8) scope and complexity of system.
5. Finally, in selecting the vendor, disregard salesmen's claims. Rely on: (1) facts from information gathered, (2) vendor past performance, and (3) evaluation of people you will work with in system development. Hire a lawyer familiar with computer systems to write an addendum to the vendor's standard contract. See Appendix B for some suggested clauses.
6. Additional system considerations:
   a. Specify a system that is CRT and keyboard driven, cards are costly and unwieldy for small systems.
   b. A cassette based system should never be purchased. Cassettes store ¼ million characters of data that can only be accessed by searching the entire cassette for a particular piece of information. Floppy disks store the same amount of data that can be randomly accessed. A floppy disk system should only be purchased if the data volume to be stored is very small and its growth stagnant. A hard disk costs $10,000 vs. $4,000 for a floppy disk and provides substantial growth benefits.
   c. The critical issue is the availability of software to meet your specific needs. Concentrate on this issue. Hardware is in general not a problem.

APPENDIX A—EXAMPLES OF FUNCTIONAL SPECIFICATIONS.

CPA FIRM/SERVICE BUREAU

Software requirements

1. *Payroll System*—Approximately 50 clients, mode is 5 to 10 employees, five clients with 20 to 50 employees. System must have ability to prepare checks, year-to-date registers, payroll registers, labor distribution by cost center or department, and also prepare W-2's on demand as well as year-end W-2's and quarterly 941-A reporting. A vacation, holiday, and sick leave accrual system is a helpful addition. The payroll reporting subsystem must interface with the general ledger data entry processes to capture by-product payroll data and retain this data for quarterly and annual reporting.
2. *Accounts Receivable System*—Data entered via sales journal and cash receipts journals to produce statements and an aged trial balance.

3. *Accounts Payable System*—Fully integrated system for both cash and accrual based clients. Check writing capability, volume: 6,000 checks per month.
4. *General Ledger System*—Fully integrated with Payroll, A/R, A/P systems must be capable of using the Autotax system chart-of-accounts. Capability of producing subsidiary financial statements.
5. Availability of a standard data input editing capability to include data type field checks, range checks, account code and check digit, etc., and data output control checks is a plus.
6. The current system must be capable of handling 60 clients, having 40 accounts per general ledger account and be expandable in increments to handle 200 clients requiring A/R maintenance, payroll and all standard accounting functions.

System requirements

1. *On-line Interactive System*—Initially a single CRT station, modular and expandable to four to six CRT stations capable of simultaneously accessing system. Software protection of database update an important consideration in maintaining data integrity. Please specify cost of each additional CRT station.
2. *Line Printer*—Low speed, 64 character set, reliability critical.

X-BAR FLYING SERVICE

1. *Data Processing Requirements*
   Processing—The system shall be capable of meeting X-Bar's immediate accounting needs with a fully integrated on-line A/P, A/R, G/L, payroll, inventory, software package.
   Input—Input will be on-line, single item entry with editing capability.
   Output—CRT and line printer (100 to 200 lines per minute).
2. *Future System Requirements*
   Processing—The system shall be capable of software expansion to meet future management control information needs. It must be possible to easily access and restructure data to provide for new management information requirements. It must be possible to process data from at least four separate companies simultaneously; that is, the data may be updated and accessed by remote terminals in at least four different locations. Input/query capability from several local and remote locations. Some terminals may be connected directly to the system while others are connected by a leased telephone line.
3. *Modular Expansion*
   The system must be expandable in modules of hardware, applications software, and operating systems software. Each perspective vendor is expected to quote the basic system and then show how the system can be expanded to meet the future system requirement.
4. *Maintenance*
   Maintenance must be Albuquerque based with one half day response.
5. *Backup*
   An Albuquerque based backup system fully compatible with X-Bar's specified system must be available. Excess time (current and projected) on the backup system must be available.
6. *Training*
   Local training support must be available and easily accessible.
7. *Availability*
   Only a "try before you buy" philosophy will be accepted. Each vendor will be expected to demonstrate the basic system with X-Bar data to satisfaction of X-Bar before purchase. In addition, the vendor must demonstrate a currently operational expanded system that would meet X-Bar's future needs. The system shall be available in 30 to 45 days after the order is received. A similar system and sufficient system time must be available to affect immediate conversion from X-Bar's old system to the new system.
8. *Guarantee*—The vendor must be willing to include a penalty clause and/or initiate an escrow account to cover possible loss to X-Bar due to the system not meeting specifications. The vendor must agree to submit to arbitration should a vendor-buyer disagreement arise.

APPENDIX B—THE STANDARD CONTRACT

The standard vendor contract is not immutable, though that is often the impression presented by the vendor representatives. The truth is that the vendor contract is written to favor the vendor, hence, is inequitable and should be modified to be fair to both parties.[1] The data processing manager and the company executive staff have definite wishes regarding:

(1) Delivery date commitment
(2) Documentation requirements
(3) Trade-in options
(4) Reporting requirements (on a software development contract)
(5) Performance test requirements
(6) Service commitments

They, however, do not have the expertise necessary to translate these requirements into contractual language. A lawyer, on the other hand, normally does not have the expertise to determine what is required. Together the DP manager/corporate executive and negotiating lawyer form a symbiotic relationship. Every vendor will renegotiate a standard contract, if pressed.

Initially, the corporate staff should decide what the critical contract objectives are. For example, if reliable service is the most important consideration, then attention should focus on warranty and maintenance provisions. These contractual objectives must be articulated and translated into appropriate legal language by an attorney.

Some contract considerations follow:*

(1) Warranty—Will the equipment function according to software and hardware specifications?
(2) Terms and Price—Specific conditions under which title the computer and/or software passes to vendor. In a lease contract, who pays the property taxes? Is the lease renewable? Is there an upgrade/downgrade penalty clause?
(3) Delivery and Installation—Delivery dates should be designated with penalties for late deliveries. Provision should be made for damage in shipment.
(4) Acceptance Testing and Maintenance—Under what terms will the vendee accept the equipment? A minimum acceptance period is 30 days. A maintenance clause should specify the maximum response time for service calls.
(5) Training—The type and duration of training should be clearly specified.

In summary, every conceivable event should be included in the contract. Do not accept the verbal agreement with a vendor official, he can be overruled later, or be powerless to follow through if the agreement is not in writing.

* The Auerbach *Data Processing Manual* contains model contracts for computer equipment. Access to this document may be available at a local university library, or a large corporation, university or city data processing department. See also Brandon, Dick H., and Sidney Segelstein, *Data Processing Contracts—Structure, Contents and Negotiation,* VanNostrand Reinhold Company, New York, N.Y., 1976.

REFERENCES

1. "Microcomputers Catch on Fast," *Business Week,* July 12, 1976, p. 50.
2. "The Smart Machine Revolution," *Business Week,* July 5, 1976, pp. 38-44.

Appendix

1. Bucci, R. A., "Beware the Standard Lease," *Datamation,* March 1973, pp. 75-76.

Storage utilization in a self-organizing data base

by P. M. STOCKER
University of East Anglia
Norfolk, England

ABSTRACT

The controlling mechanisms within a self-organizing base are concerned with maintaining the balance between search efficiency and update efficiency, and with resolving conflicting requirements for physical adjacency. This paper shows how the concept of the value of store to a process may be used in the control mechanisms, and discusses the connection between values derived from theory and values measurable in operation.

INTRODUCTION

A proposal for the general structure of a self-organizing data management system was made in Stocker and Dearnley;[1] an account of a pilot implementation was given in Dearnley;[2] and some account of its operation in Dearnley,[3] and Stocker & Dearnley.[4] A second implementation is currently in progress, which follows the general structure of the first, but which includes more sophisticated searching and structuring techniques. In particular, the structures forming the data base are now viewed in a less rigid manner and the self-initiated changes in the structures, in response to changes in usage, are seen more as continuous adaptation than discrete changes.

The data base management system has two principal components. The first determines a near optimum procedure to process a transaction presented to the system, and in almost every instance will necessarily make use of the overall data structure which exists at the instant when the transaction is presented. The second component restructures the base in response to data instance changes, past usage and any known information concerning future usage. Its objective is to minimize overall future cost; that is, user cost, together with any internal cost arising from restructuring activities. The basic processes of the restructuring component are described in Stocker,[5] and are briefly summarized here. The processes in the base are seen as a hierarchy, each monitored by a control routine which adapts it to the usage pattern in which it is currently employed and will be employed in the future. In a sense, each data structure may be viewed as part of a processing plant, with an associated capital and operational cost.

General description of control mechanisms

At the highest level the keys which are to provide direct access to the data are controlled. The process is that of adding a new key or ceasing to maintain a current one. At the next lower level control is maintained over the record-sets which may be accessed by these keys. These record-sets may be regarded loosely as main record-sets and secondary indexes, one or more record-sets being associated with each access key. There are four processes at this level:

(i) The partitioning of the data items associated with a particular key into record-sets, if more than a single record-set is found to be advantageous (i.e., cheaper).
(ii) The migration of a data item from a record-set on one key to a record-set of another key which is in 1-1 correspondence with the first.
(iii) The duplication of data items by including them explicitly in more record-sets than is logically necessary, including secondary sets.
(iv) The specification of which main files shall be indexed by secondary ones.

The control routines associated with these processes periodically review the activities of the system and may cause a process to undertake its activity, that is, move, delete or duplicate a data-item with respect to a record-set.

Below the processes operating at the record-set level are others concerned with the file structure of the record-sets. It is assumed that these files are implemented in terms of "frames," the normal unit of transfer from backing store to core in the file processor. Lower level processes with associated controls are concerned with the inter-frame structure and with the intra-frame structure.

In the present system the *unit of cost is equated to one transfer of a frame between core and backing storage.*

OBJECTIVES

This paper is intended to clarify two aspects of the self-organizing system. The first is the relation between the formal optimization of data structures carried out mathematically under assumptions of random rectangular distribution of data key instances and the more ad hoc procedures which operate on statistics to be measured during operation of the management system. Two examples are chosen, one of a low level control, which operates frequently, and so must have a low overhead. The second control mechanism operates at a high level and, because it is comparatively infrequently invoked, can bear a higher overhead.

The paper is intended also to clarify an important aspect of the data structures. The data structures constructed by the processes described are quite neutral with respect to hierarchic, relational or network definitions of the data in an external user description. The second example is chosen to show how the storage structures react to various forms of usage (not external description). It is hoped that this demonstration will assist in the understanding of the desirable separation between the internal specification of adaptive data base systems and the external view of the data.

OVERFLOW PROCEDURES

Consider a growing sequential file, with index. The file itself may contain actual data or be the lowest level of a multi-level index; see for example, Schneiderman and Goodman.[6] Two possible overflow procedures, when a frame becomes full, are as follows:

(i) Push a number of records to the next frame in sequence,
(ii) Obtain a new frame and divide the contents of the full frame between the two.

If the rate of growth is slow the (i) is attractive since frames will be well filled and overflow will be infrequent. If the growth rate is high (ii) appears attractive.

If strategy (ii) is adopted it can be shown that the average density will be 100 (2 $\log_e 2)^{-1}$=72 percent. The difference from 75 percent is due to the fact that the rate of gain of additions depends upon the page content. For strategy (i) to show an improvement a density higher than this must be attained and the gain from the increased density must more than offset the cumulative nature of the overflow.

In all the processes one should ask, "is the increased complexity justified by the gain?" The gain may be appraised as follows. For a slow-growing file strategy (i) may produce a density approaching 100 percent. This corresponds to a 39 percent saving in disc space over strategy (ii). The saving in access costs will also be 39 percent for a batch of keys which access every frame in the file, but will be zero for a batch where the number of keys is small compared with the number of frames. Finally, strategy (ii) preserves the physical ordering of frames.

The strategy considered here is to partition the file into segments such that strategy (i) is used throughout the segment until it is terminated by a frame following strategy (ii), and hence terminating the overflow propagation.

In the next section a theoretical basis is deduced, leading to the simplified operational algorithm of a later section.

Theoretical basis

Consider a file containing r records having a random rectangular distribution of keys. Consider also a "frame" of that file containing p records and of maximum content q. A batch search supplying m keys of similar distribution is applied to the file.

The probability that the frame is accessed is $1-e^{-mp/r}$. The expected number of keys serviced by the frame is mp/r.

Let Σ denote a sum over a spectrum of values of m (the usage distribution) and let ν be the number of additions to this frame over a time period covered in the usage distribution. Note that ν contains the cumulative overflow from frames earlier in the segment.

Overflow from the frame will cause four additional accesses (read and write overflow frame and index). The cost per query serviced may be written:

$$C=\frac{4\nu/(q-\bar{p})+\Sigma(1-e^{-mp/r})}{(p/r)\Sigma m}, \tag{1}$$

where $\bar{p}$ is the page occupancy after overflow has been performed and p is 'an average value' for page occupancy during operation. Because overflow will tend to cascade down the segment $p=\bar{p}$ is a good approximation.

Segment ends

At a segment end one must have the approximate relation:

$$\frac{4\nu(q-\bar{p})+\Sigma(1-e^{-m\bar{p}/r})}{(\bar{p}/r)\Sigma m}=\frac{\Sigma(1-e^{-0.72mq/r})}{(0.72q/r)\Sigma m}, \tag{2}$$

where it has been assumed that cumulative additions heavily outweigh local ones. With obvious approximation, assuming that $\bar{p}\sim q$, one may obtain:

$$\bar{p}=\frac{0.72q[14\nu+\Sigma(1-e^{-0.72mq/r})]}{\Sigma(1-e^{-0.72mq/r})}. \tag{3}$$

That is, a sequence should terminate if the estimated $\bar{p}/q$ is less than 0.72[1+14 (Number of Additions)/(Number of Accesses)] and this rule will serve for the ad hoc process.

Optimization

Consider the effects of changing $\bar{p}$ to $\bar{p}+1$ in equation (1). Overflow costs per query will increase in the ratio $(q-\bar{p})/(q-\bar{p}-1)$. Search costs will be unchanged for small batches (as can be shown by expanding $e^{-mp/r}$ in powers of m). For large batches they will change in the ratio $\bar{p}/(\bar{p}+1)$. This discrepancy raises a difficulty, but expansion suggests that a satisfactory approximate ratio is given by $1-k/p$, where $k=1-\xi$, ξ being the ratio of accesses to queries serviced.

The previous paragraph omits mathematical detail and is expressed in terms appropriate for the control algorithm.

Control

It is simple to arrange that the data management system record on each frame, when it is accessed, the number of queries serviced and the number of accesses since the last overflow, and to keep weighted values of these for, say, the last three periods between overflow.

When overflow next occurs, the following computation is performed:

1. Form:

$$x=\frac{\bar{p}(q-\bar{p})}{(\bar{p}+1)(q-\bar{p}-1)}\times \text{overflow cost}\ (=4),$$

$$\xi=\text{accesses/keys serviced},$$

$$k=1-\xi,$$

$$y=\left(1-\frac{k}{\bar{p}}\right)\times \text{access cost},$$

$$z=1+14\times(\text{additions/accesses}).$$

2. If $x+y>$(overflow cost + access cost), then decrease $\bar{p}$ by 1 when processing overflow. Otherwise increase $\bar{p}$ by 1.
3. If the new $\bar{p}$ is less than 0.72z treat overflow by strategy (ii), record $\bar{p}$ as 0.72z and reset weighted averages on both frames involved.
4. If new $\bar{p}$ is greater than or equal to 0.72z use strategy (i).

Note that increased packing on backing store may be achieved by adding an additional term proportional to real time and scaling like $p/(p+1)$, to represent backing store rental.

The control scheme achieves local control and the file can adapt to an increase or decrease in additions in a particular key area.

STRUCTURE FORMS

In British Universities, students apply to a national center for admission. A record is maintained under the applicant's unique code number which contains personal data together with a repeating section to contain details of the progress of applications to a number of universities. These records might be of the logical form:

Applicant's Code Name Age
University Code II Course Code
Repeating

The basic logic of the management system may reject the embedded repeating groups and may store such data in two record-sets, thus:

Set A
App. Code Name Age
and
Set B
App. Code Univ. Code Course Code
App. Code Univ. Code Course Code.

Alternatively, it may accept the repeating group in practice, by retaining the two logical record-sets A and B, but storing instances corresponding to the same key in the same frame. In addition, the intrapage control mechanism will suppress some key duplication but that aspect is not mentioned further in this paper.[5]

Statement of the problem

The decision concerning whether the sets A and B should be physically distinct or interleaved is determined by a process specific for this purpose. Three kinds of query access using Applicant's Code as key are possible, those which refer to A alone, B alone, or A and B in combination.

If the size of all query batches is small then the combined structure is optimal (the case where the repeating group for a single key exceeds a frame is ignored). If the batch sizes are larger then the low packing density of A in the interleaved file it may prove more costly than the gain achieved for queries involving the combination.

An approximate process

The number of accesses to a file of n frames to process a batch containing m key instances is:

$$n(1-e^{-m/n}).$$

Thus, to properly cost a number of batches, N_B, one must be able to compute:

$$nN_B-\Sigma n_m e^{-m/n}.$$

In this case, we are concerned with assessing the merits of widely different values of n, and the incremental approach of the preceding section will not suffice. It is necessary, therefore, that statistics be maintained by the management system concerning the batch sizes of queries concerning particular data items. The degree of detail required is dependent on file sizes. A satisfactory division may be an octal one. That is, for each access presenting "Applicant's Code," as key, and for each combination of items sought, a small table is maintained which records a count of the queries in each batch size range, $(8^i, 8^{i\pm1})$. This will be called the batch distribution table, BDT for brevity.

Periodically, a review of the following form is produced. Compute the BDT for the next time period for sets A, B and AB. Compute the likely average number of frames in A, B and AB. Compute the total cost for the next period of

A serving A and AB queries, denoted by C(A), a similar statistic for B, and the cost of AB serving A, B and AB queries. Suppose the system were in the AB state. Form:

$$Q=C(A)+C(B)-C(AB).$$

If Q is positive no further action is taken. If Q is negative, call a procedure which evaluates the cost of the AB→A+B stored data transformation, say, R. It is now necessary to use a further statistic, extremely difficult to obtain, which is the likely time span for which the current usage will continue. Call this T. Let S be the cost, for the next time period, of "interest" and repayment of R over period T. Then if S is less than Q the process controlled is called, that is, the stored data transformation AB→A+B is carried out.

For completeness, the action of another control in this particular problem will be mentioned. Suppose that accesses to B were via the key "University Code," then B would be ordered and indexed on this key. If the items searched for often included "Applicant's Name," an extra access to A via "Applicant's Code" is involved.

This problem may be approached by maintaining usage statistics for the actual file B, as distinct from the BDT's which are system statistics. The file statistics are counts of the hits on data-items in B, either as directs or indirects. That is, they record whether the item was searched for as a required output or as a reference to an output. This is done by the query processor. If the indirect count is high the BDT associated with "University Code" can be consulted and a reason deduced. An appraisal of the value of including "Applicant's Name" in *both* A and B would then be carried out on lines similar to those described eariler in this paper.

CONCLUSIONS

From the foregoing, it will be seen that the concept of processes and control routines provides a system which may produce data structures which correspond to relational concepts (A+B), hierarchic concepts (AB), secondary indexes or CODASYL sets, and which contains controlled data duplication. The concept also more directly provides change towards optimality at lower levels. The author sees the main problem as that of minimizing the number of processes and of finding the most nearly orthogonal set. The simplicity of implementations will be determined by how well this problem is solved.

REFERENCES

1. Stocker, P. M. and P. A. Dearnley, "Self-Organising Data Management Systems," *The Computer Journal,* Volume 16, No. 2, 1973.
2. Dearnley, P. A., "A Model of a Self-Organising Data Management System," *The Computer Journal,* Volume 17, No. 1, 1974.
3. Dearnley, P. A., "The Operation of a Model Self-Organising Data Management System," *The Computer Journal,* Volume 17, No. 3, 1974.
4. Stocker, P. M. and P. A. Dearnley, "A Self-Organising Data Management System," published in *Data Management,* edited by J. W. Klimbie, and K. L. Koffeman, North Holland, 1974.
5. Stocker, P. M., *IFIP-TC-2 Working Conference* on "Modelling in Data Base Management Systems," Nice, North Holland Publishing Co, 1977.
6. Shneiderman, B. and V. Goodman, *A.C.M. Transactions on Data Base Systems,* Volume 1, No. 3, September 1976.

Self-adaptive automatic data base design*

by MICHAEL HAMMER
MIT Laboratory for Computer Science
Cambridge, Massachusetts

ABSTRACT

Physical data base design, the selection of organizational structures and access mechanisms for a data base, is one of the most important responsibilities of a Data Base Administrator (DBA). A DBA often has difficulty in performing this task; he lacks the information needed to choose a design that is well matched to the data base's mode of use.

This paper presents the design principles of an automatic system that has the ability to choose the physical design for a data base and to adapt this design to changing requirements. The components of such a system include: an information gathering module that collects global statistics on the overall usage pattern of the data base; a predictor that projects observed usage statistics into the future; a design evaluator that computes a figure of merit for any proposed design; and a heuristic proposer that synthesizes a small set of candidate designs for detailed consideration. These principles have been applied to the design of a system that selects secondary indices for an inverted file system.

INTRODUCTION

The advent of shared integrated data bases has called into being a new job function, that of the Data Base Administrator (DBA). The DBA is charged with responsibility for a collection of data that is being used in differing ways, for varying reasons, by a disparate community of users; he has authority over an important resource that no single user can (or should) control. It is his task to mediate the conflicting needs of the user community and make decisions regarding the organization and maintenance of the data base. The particular tasks associated with this important position cannot be easily summarized in a job description. By all accounts, one of the DBA's major areas of responsibility is for the physical design of the data base; by this, we mean all decisions pertaining to the structure and representation of the data and its associated access mechanisms.

The physical design of a data base will determine the cost of its use and maintenance: the time it takes to perform retrievals and updates, and the space needed to store the data and its associated auxiliary structures. (A terminological note: We use "update" as a generic term to include insertions and deletions of records as well as the modification of existing records.) In an important sense, physical design issues should be largely invisible to users of the data base: i.e., they should primarily affect only system performance and not the ways that users view the data or write their programs. The latter sort of concerns are usually gathered together under the term "logical design." In some contemporary systems, the choice of a logical schema does have a direct impact on the structure of the physical representation of the data, and so performance issues do impact the users' view of the data. But with the growing trend towards data independence in modern systems, the logical and physical design processes are being increasingly decoupled. Logical design is an exercise in modelling, wherein the DBA attempts to design a logical data structure that effectively captures the fundamental semantics of the application domain and thus enables users to express their transactions with the data base in a natural and convenient way. Physical design can be viewed as the process of reducing this abstract representation to a concrete one, which is expressed in terms of physical data structures and access mechanisms; the only relevant issue at this level should be performance.

Physical design encompasses an extensive set of issues; which ones are relevant in the design of a particular data base depends on the kinds of file structures and access methods that are supported by the data base management system (DBMS) being used. Typical design problems in this domain include: choosing the primary access methods for the files of a data base (e.g., sequential, indexed sequential, or direct); for a sequential file, selecting the major and minor keys by which the file is to be sorted; identifying the set of fields for which to maintain indices, and choosing the structures of these indices.

In most cases, the set of possible physical designs for a given logical data base is very large. No single one of them is the optimal in all circumstances. Rather, one design can be said to be better or worse than another only in the context of a particular pattern of use for the data base. It is well-known that any particular physical design of a data base will enable the efficient execution of certain retrievals but not of others, and that it will entail extensive mainte-

* 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 N00014-76-C-0944.

nance activity for certain updating activities but not for others. Thus, the objective of the physical design process is the selection of a physical representation that will provide good performance in the context of the particular mix of retrievals and updates to which the data base will be subjected.

There is another factor that affects the choice of the physical design for the data base, which might be called the internal characteristics of the data. This includes such issues as the sizes of files, the ranges of values that fields can assume, the distributions of values in fields across a record type, the cardinalities of relationships between record types, the typical numbers of elements in repeating groups; these all affect how a particular physical design will perform for a set of transactions, and so should influence the choice of the physical design.

Large, shared data bases are not static entities. When a data base is used as an autonomous organizational resource, rather than as a set of master files for some particular programs, it develops a highly dynamic life cycle of its own. Its usage patterns can shift dramatically as old applications evolve or are replaced, as new applications emerge, as users acquire increased sophistication and familiarity with the system. Internal characteristics of the data may change as well, reflecting the changing nature of the domain being modelled by the data base. Consequently, the tuning of a data base's physical organization must also be a continual process; physical redesign, to meet changing requirements, is as important as the initial configuration process.

In light of all these issues that influence the choice of a good design, it is not surprising that a physical design chosen by the DBA in an intuitive fashion, based on his qualitative impressions, is likely to result in unsatisfactory performance. But problems also face the DBA in trying to choose a design in a more systematic fashion. First of all, it is difficult for him to obtain an accurate usage pattern for the data base as a whole, since he typically has meaningful interactions only with a limited number of individual users. Secondly, a modern data management system is a complex device, and it is hard for a human DBA to develop a useful intuitive model of its operation; on the other hand, a more precise model would be so complex that the evaluation of any proposed design would be an extremely cumbersome process. Next, even if the DBA were able to assign a figure of merit to any particular design, he would not be able to use this ability to select an optimal structure, because the number of candidate organizations is almost certain to be too large to allow their individual consideration. Finally, because of the evolving character of data base usage and data characteristics, even a well-chosen design is not likely to remain a good one for long. Obtaining meaningful predictions as to what the relevant parameters affecting the design will be like in the future is even harder than determining what they have been in the past.

Therefore, the DBA needs help in choosing the physical design for his data base; and it is natural to look to the data management system itself to assist in (or even subsume) this process. It certainly is the best source for all the information needed to do the task effectively. Since the DBMS processes all user transactions it has the capability to build up an integrated and accurate model of data base usage; it should also be able to detect emerging trends and project them into the future. Similarly, it can obtain summaries of the pertinent internal characteristics of the data during the normal course of its operation. It could also be provided with a detailed model of its own operation, in order to compute useful cost figures for candidate physical designs. What we are proposing is imbuing the data management system with a limited form of "intelligence", thereby enabling it to share in the making of decisions heretofore entirely within the province of humans.

In the remainder of this paper, we describe how a data management system can participate in the physical design process; initially, by means of a set of tools to aid a human DBA, and then through a totally automated data base design facility.

INFORMATION GATHERING AND DESIGN EVALUATION

The basic components of any automated or semi-automated physical design system are a design evaluation module and an information gathering module. The former is used to comparatively evaluate any set of candidate designs for the data base and thereby rationalize the process by which a design is selected. In order to perform such automatic physical design evaluation, the system must have access to information that characterizes the use of the data base and its contents. It is the responsibility of the information gathering module to collect such data. This module can be incorporated into the interface language processor of the DBMS; it can be thought of as watching over the shoulder of the language processor and taking notes on what it sees. The choice of what information it should collect is determined by the design issues to be addressed and the approach being taken to design evaluation by the evaluator module. This choice should also be influenced by considerations of accuracy and efficiency. On the one hand, succinct summary statistics may not provide enough data for the precise analysis of possible designs. On the other, it may be prohibitively expensive to gather and store a large amount of data of a fine-grained character; furthermore, it might be difficult to make effective use of a mass of low-level facts.

Earlier efforts in the area of automatic data base design have utilized rough summary information to describe the use of a data base and its contents.[1,2] This description of a usage pattern in terms of a small number of parameters has been used to characterize an "average" transaction, with respect to which any proposed design could be benchmarked. The difficulty with this approach is that it obscures a great deal of detail that is crucial in tuning a data base design to match its mode of use; much relevant information is lost in describing a usage pattern in terms of a single average transaction.

We believe that for most design decisions, the most effective way to evaluate a proposed design is by examining its aggregate performance for all the individual transactions that comprise the usage of the data base. The system should utilize a model of the operation of the DBMS, together with information about the internal characteristics of the data, to forecast how much processing would be done by the DBMS in the handling of each individual transaction; the sum of these processing costs over all transactions in a usage pattern can be combined with certain other cost figures to achieve an overall figure of merit for the proposed design. Therefore we shall attempt to gather as much detailed information as possible about the individual transactions with the data base and its internal characteristics.

Practically, it would be infeasible to maintain a separate record of every transaction that occurs with the data base; it would also be prohibitively expensive to attempt to analyze each one when considering a proposed design. Therefore, we shall partition the set of transactions into transaction classes such that all transactions in the same class will have the same essential structure. For example, the class of a retrieval request will be determined by the field names and comparison operators occurring in the atomic predicates and by the structure of the total predicate in terms of its logical connectives. The intent is that all transactions in the same class should entail roughly the same amount of processing by the DBMS; while this may not be completely accurate, we believe it represents an acceptable tradeoff between efficiency and accuracy. Then a usage pattern will be expressed by the occurrence frequencies of the various transaction classes; a design will be evaluated by considering its (weighted) performance for the different classes, as predicted by the model of the DBMS. Henceforth, we shall take "transaction" to mean "transaction class."

There are two further points to be observed about the information gathering module. First, it is imperative that the process of gathering information not significantly degrade DBMS performance; the statistics that this module collects ought to be readily available in the normal operation of the DBMS. Secondly, the usage pattern may have to reflect more than just the occurrence frequencies of the various transactions. It may well be that certain users or applications have a higher priority than others, and that their needs and habits should carry more weight in determining the physical design of the data base.

The information gathering module provides the background knowledge essential to choosing a physical design; this information is utilized by the physical design evaluator. This module takes two inputs: the output of the information gatherer (a description of the usage pattern and internal characteristics of the data bases) and a proposed physical design for the data base. Its function is to produce a figure of merit that reflects the suitability of the proposed physical design for a data base with the specified internal characteristics that is being used in the way described by the usage pattern.

The evaluator will be used to compare alternative designs and select the best of any set of candidates. The figure of merit need not be a completely accurate measure of the cost associated with an individual design, but it must allow for reliable comparisons to be made of alternative designs.

The design evaluator will be built around a transaction cost estimator. This estimator is called with a transaction, a proposed physical design for a data base, and a summary of the data base's internal characteristics; it computes a figure that reflects the cost that the DBMS would incur in processing the transaction, if the data base were structured in the proposed way. The units of this estimate could be expected I/O processing, or expected CPU time, or a combination of the two.

The bulk of a (naive) design evaluator could consist, then, simply of an iteration over the set of transactions in the usage pattern; each transaction would be submitted to the cost estimator, and the result tallied into a running total. The value computed by this iteration could be combined with a figure reflecting the costs of storing and maintaining the access structures, in order to achieve a unified figure of merit for the proposed design. We shall discuss later the shortcomings of this naive evaluator and how they may be repaired; however, in the interim, this can serve as a useful model of the structure of this module.

The transaction cost estimator requires a detailed model of the operation of the DBMS. By scanning the structure of a transaction, this module will determine what strategy the DBMS would follow in processing it: what access mechanisms would be utilized and in what order, what kinds of intermediate results would be generated, and so on. The detailed information about the data base's internal characteristics will enable the estimator to forecast such things as the sizes of the various structures the DBMS would have to scan and the number of links it would have to traverse. Thus the estimator will be able to predict the total amount of activity required of the DBMS to handle this transaction. Since the estimator will have to operate efficiently, it may have to ignore some hard-to-compute factors in the processing cost; but the dominant terms should be readily approximated.

Such a cost estimator is an entity of interest in its own right, independent of the problem of self-organizing data bases. Many contemporary data base systems provide a "stand-alone" language interface, which is intended for use by relatively unsophisticated users in posing ad hoc inquiries to the data base in real-time. An unfortunate property of such a facility is that it enables a naive user to ask the system a seemingly simple query whose processing will cost a great deal more (either in terms of elapsed time or other system resources) than the answer is worth to him. A transaction cost estimator could inspect any query and quickly return to the user an estimate of its processing cost, enabling him to abort expensive queries and to better plan his data base usage.

Although the information gatherer and the design evaluator form a coherent unit, either one by itself would still be useful to the DBA: the information gatherer could provide him with data for his use in intuitively designing file structures, or the design evaluator could operate with usage statistics that he supplied it.

Theoretically, the DBA could run the evaluator on the full set of all possible designs and select the one with the best figure of merit; he would then be reasonably certain of having chosen a near-optimal structure. More realistically, the DBA would intuitively choose some small set of structures to investigate more closely, and subject them to careful analysis and evaluation.

REDESIGN AND USAGE PREDICTION

A scenario of the use of these basic DBA design tools is as follows. At the time of data base creation, the DBA generates a pattern that he believes represents the expected usage pattern of the data base, and uses it as input to the evaluator in selecting the initial physical design. He then activates the information gatherer to monitor the actual transactions with the data base and determine its real mode of use. When sufficient statistics have been gathered, the DBA can decide if his initial guess was accurate. If not, he can use the evaluator again and choose another design that better fits the observed usage.

The flaw in this primitive scenario is, of course, that a data base usage pattern is not static, and so even a well-chosen design may become obsolete. Consequently, it is desirable to incorporate the concept of redesign into this picture. To accomplish this, we introduce the notion of redesign points, which could occur on a regular basis or on DBA request (for example, upon his noticing a significant degradation in system performance). At each such point, the statistics gathered since the last redesign point are summarized into a usage pattern, which presumably defines the most recent mode of use. The evaluator can then be used to help the DBA select a physical design optimally suited to this usage pattern. This design is the structure of choice in the current circumstances.

This approach raises some issues that must be addressed. One of these involves the cost of performing a redesign. If it develops that the optimal design for the evidenced usage pattern is different from the existing design, then there is almost certainly going to be some cost associated with the process of transforming the data base from the old physical organization into the new one. The extent of this cost depends on the size of the data base, the kinds of structures involved, and the degree of the difference between the old and new designs. It may be the case that the cost improvement to be gained by using the new design rather than the old will be washed out by the expense of the reconfiguration process. Consequently, this latter cost must also be considered in choosing the physical design. The optimal design is thus defined as that design D for which F(D)=C(D)−C(Do)−T(Do,D) is a maximum, where Do is the existing design, C(D) is the total cost associated with design D for the usage pattern observed since the last design point (as computed by the evaluator), and T(Do,D) is the cost of transforming the data base from design Do to design D. The evaluator can readily be modified to compute F(D) for any proposed design.

The other problem with our redesign scenario is rather more fundamental. In the process described above, we are always designing for the past, optimizing for a usage pattern that is already over. The tacit assumption behind this is that the usage of the future will strongly resemble that of the past. However, it may happen that the redesign points are badly situated with respect to the evolution of the usage pattern, and that statistics gathered since the last design point are dominated by transactions least representative of future usage. This situation might well obtain if redesign points occur when system performance is just beginning to degrade.

For this reason, it is appropriate to be more sensitive to the problem of changing usage patterns and to include in the system a predictor module. This component will interpret the observed statistics and explicitly translate them into an anticipated future usage pattern. The intent is that this module will detect evolving trends in data base use even before they become fully established, and so enable the data base structure to be prepared in advance for the demands to be put on it. A predictor module needs access not only to the usage statistics of the most recent time interval, but to those of earlier periods as well. By analyzing the historical trends of the various parameters that comprise the usage pattern, this component can synthesize a description of expected use for the upcoming interval. It is with respect to this predicted pattern that the alternative physical designs will be evaluated.

A good predictor module must satisfy a number of criteria. It must not be overly vulnerable to chance fluctuations in usage, while still responding to real change. Therefore, it cannot base its decisions purely on recent events, nor on the aggregate of information gathered over the entire history of the data base. Intuitively, some weighted average of the two is desirable. At the same time, the predictor cannot demand access to a great deal of detailed information from all previous time periods, since both the gathering and storage of such information is likely to be prohibitive in cost; nor can it attempt to perform extensive computations for each prediction that it makes. A useful usage pattern is likely to be composed of a large number of parameters, and a lengthy evaluation for each one is impractical.

A promising predictive technique for this application is exponential smoothing.[3] The basic formulation of exponential smoothing in making a forecast of a discrete time series is as follows:

new forecast=a*(actual observation from last period)+(1−a) *(previous forecast)

where a is called a smoothing constant and takes on values between 0 and 1. In essence, this computes a weighted average of all previous observations with the weight decreasing geometrically over successively earlier observations. The rate of response to recent changes can be adjusted simply by changing the smoothing constant; the larger the smoothing constant, the more sensitive is the forecast to recent changes and chance fluctuations. (The value of the smoothing constant a can be selected by the

DBA or can be adaptively chosen by the system itself to minimize the difference between observed and predicted data.) Presumably, the predictor would maintain a time series for each parameter in the usage pattern. In this scheme, only two values need be maintained for each series: the current observation and the previous forecast. The computational requirements of this approach are also minimal.

This basic scheme can be modified in order to make it more sensitive to evolving trends. The revised formulation is as follows:

new average=a*(current observation)+(1−a)*(old average)
current trend=new average−old average
new trend=a*(current trend)+(1−a)*(old trend)
new forecast=new average+((1−a)/a)*(new trend)

The new trend is a smoothed average of the differences between successive basic forecasts (as formulated above), and so represents the direction that these forecasts are taking. The revised forecast is the basic forecast modified by a weighted version of the new trend. Here, too, storage requirements are minimal; space is needed only for the current observation, the old average, and the old trend.

A number of issues must be confronted in attempting to apply exponential smoothing to data base usage prediction. One question is whether such techniques, developed for problem domains like inventory control, are really appropriate for the data base environment. The fundamental problem is determining the nature of the patterns and trends that do occur in the history of use of a centralized data base by a diverse community of users. Certainly, such histories seem to have some aspects that are not congenial to modelling by smoothing techniques. Ultimately the validity of this approach to usage prediction can only be determined by careful study of extensive amounts of appropriate data that have been gathered in an operational environment.

AN AUTOMATIC DESIGN FACILITY

As we have described it, the information gathering module collects the statistics on observed data base use, and passes them to the predictor module, where they are converted into an expected usage pattern. The design evaluator utilizes this prediction in assessing possible designs. What, then, remains the function of the human DBA (with respect to the data base design problem)? It might be said that his first responsibility in this area is that of any human working with an automated decision-making system: to assure that the system's decisions are sensible, that they reflect not only an abstract model but concrete reality as well. Thus the DBA should interpret the statistics that are gathered and the usage that is forecasted, decide if they are plausible, and manually modify them if he feels they are incomplete. Similarly, he must exercise human judgment in his use of the design evaluator, deciding what its results mean in terms of actual data base design. The DBA may need to go beyond the limited contexts of these automated tools, in order to account for hard-to-quantify factors not incorporated in their computations and consequently not reflected in their output. For example, the DBA might decide that the best design is not the one with the lowest cost figure as computed by the evaluator; his decision may be influenced by taste and intuition.

There is a problem with the foregoing argument: while the kind of judgment described above is fundamentally beyond the capacity of a machine, it may also effectively exceed a human's capability as well. With large data bases being used in a wide variety of ways, and with complex data management systems that make clever use of intricate storage structures, a DBA may find that he does not have the subconscious understanding of "what's really going on" needed to transcend the purely quantitative capabilities of an automated design system. He may be forced to accept the decisions of his design aids, because he does not have any real basis for overruling them.

Consequently, if the DBA is provided with an information gatherer, a predictor, and a design evaluator, his role in the physical design process is reduced to deciding which physical designs warrant detailed examination by the evaluator. To be sure, synthesizing a good set of candidate designs is a creative task and an extremely important one. In modern data base systems, the physical design space may have many dimensions, with a large number of alternatives on each axis. The total number of possible organizations for even a simple data base is likely to be astronomical; an exhaustive iteration over all of them is clearly impossible. Thus, human intervention seems necessary in selecting a small set of promising candidates to submit to detailed evaluation.

But here too, a human's performance may be less than acceptable, for the reasons described above; the DBA may not be able to identify any of the near-optimal designs, and may send to the evaluator a set containing only mediocre alternatives. Consequently, we arrive at the notion of a completely automated designer. Previous attempts at automated physical design have attempted to derive a closed-form analytic expression for the cost associated with a design that is parameterized in terms of key properties of the design; this expression is subjected to mathematical optimization techniques in order to yield an optimal design.[4] It is our belief that reducing complex designs to tractable formulae inevitably entails their serious oversimplification. Our approach to automatic design relies on an intelligent module to pick a small set of promising design candidates, which can then be sent to the evaluator for detailed analysis.

Therefore, our automatic design system is comprised of the modules described above, plus a design proposer. At each design point, the proposer will inspect the predicted usage pattern and propose a set of designs to be analyzed by the evaluator. Of course, a trivial proposer could propose all possible designs, but that is clearly unrealistic. Therefore, the proposer must employ heuristics to choose a manageable number of candidate designs. There is the possibility that in some cases such heuristics will not synthesize the mathematically optimal solution; but if the heuristics are well chosen, then in all but the most pathological of contexts, the best design that they do produce should perform nearly as well as the optimal. The synthesis of a very good (if not optimal) design, at low cost, is certainly an acceptable goal for an automatic designer. The competence of a heuristic designer can be verified through experimentation, by comparing its designs in a variety of test cases to those that would be chosen after an exhaustive consideration of all designs.

It is useful to view the problem of automatic physical design as essentially one of navigating through a large search space (that of all possible designs) while looking for the optimal point. This suggests that heuristic search techniques developed for artificial intelligence applications should prove useful in selecting a near-optimal design. Some of these techniques include the following: incremental search through the problem space, with no backtracking and transition made only to improved positions; early termination of the search when a local optimum is reached; decoupling of related design decisions, with a relative ordering placed on them; a rough a priori ranking on alternatives in each design dimension, with primary attention to be paid to high-ranking alternatives.

The foregoing suggests a system design wherein the proposer operates with feedback from the evaluator. That is, the proposer begins with some initial candidate design (which is either a trivial one, or one chosen on the basis of obvious surface properties of the usage pattern, or the actual current design of the data base). The proposer then constructs a small set of variations on this candidate, based on some a priori judgments. These variations are then sent to the evaluator for analysis. That variation which represents the greatest total cost improvement over the current candidate becomes the new candidate design. (If none of its variations represents an improvement, another small set of alternatives may be considered.) The information gained in evaluating these variations determines which variations of the new candidate ought to be considered at the next step. This process continues until a local optimum is reached: a design which is an improvement over its predecessor, but over which none of the considered variations yields a better figure from the evaluator.

In cases of multiple design decisions, the foregoing procedure would be applied sequentially in the different dimensions, based on some general guidelines as to their relative importance.

Experimentation with the heuristics can determine the values of the operating parameters that achieve an acceptable balance between accuracy and efficiency. For example, by limiting the numbers of alternatives that are considered at each step, the algorithm can severely restrict the number of designs that are submitted to the evaluator.

The efficiency of the designer can be further enhanced by modifying the structure of the design evaluator. The basic difficulty with the evaluator as initially described is that it applies the transaction cost estimator to each component of the projected usage pattern; this might be expensive if many of the possible transactions with the data base are expected to occur. One approach to this problem is based on the observation that, when evaluating a design that is a minor variation of the current candidate design, most transactions will have the same cost as they did when the current candidate was evaluated; the reason for this is that most transactions will be processed in exactly the same way for two similar physical designs. Since we are only interested in the comparative evaluations of the candidate and the proposed variation, the evaluator can concentrate on those transactions that have different costs in the two cases. Another approach is to group together similar transactions into a cluster to be represented by a typical transaction; the estimator is then applied to just the representative, and the result multiplied by the size of the cluster. (This is an extension of the concept of grouping transactions into transaction classes.) This can result in a certain inaccuracy, because not all transactions in a cluster will have the same cost. This scheme is probably most effective when infrequently-occurring transactions are clustered together, while the more common ones are still treated individually. The tradeoff between efficiency and accuracy is controlled by the total number of clusters; a good value for this parameter can be obtained through experimentation.

A self-organizing data base system should be able to decide when, as well as how, to redesign a data base to fit changing access requirements; the redesigner ought to be automatically invoked when the current design begins to show signs of degraded performance. Two conditions have to be satisfied in order to signal this situation. First, that the recent pattern of transactions is deviating sharply from the predicted usage pattern to which the current design is tuned. Second, that system performance in the aggregate is proving to be less good than it had been in the past. Some caution must be exercised in invoking the redesigner, so that a transient fluctuation in usage does not cause the (somewhat expensive) process of considering a redesign to be activated. On the other hand, it is too late to wait for degraded performance to be firmly established before beginning the redesign process.

A SYSTEM FOR SECONDARY INDEX SELECTION

The foregoing discussion has been couched in general terms; the system structure described should be applicable to a wide variety of physical design decisions pertinent in different data management systems. We are developing a facility that applies these principles to the problem of selecting secondary indices for data bases managed by an inverted file data management system. A secondary index for a field A is a structure that, for a given value x, provides rapid access to the identifiers (addresses) of all records in the file whose value for A is x. A major design decision for a data base under a system that supports secondary indices is the selection of the appropriate set of fields for which indices ought to be maintained. An index on field A will speed up retrievals involving A, but will slow down updates to A (as well as requiring extra storage space). The accurate determination of whether or not a particular field ought to be indexed depends on a range of considerations beyond its relative occurrence frequencies in retrievals and updates. These include: the selectivity of the field, or the extent to which its presence in a retrieval request cuts down the number of records satisfying the request and so affects the time needed for its processing; the kinds of retrieval predicates in which the field is used; which other fields are known to be indexed; and the details of the DBMS strategies for processing transactions.

Our facility selects indices for an experimental data management system similar in structure to several commercial inverted file systems. The details of this facility are presented elsewhere [6,7]; some of its major points of interest follow.

1. The data management system operates in a paged virtual memory operating system. Since I/O processing is usually the dominant cost in data base systems, the transaction cost estimator computes the expected number of page accesses associated with the processing of the transaction. The overall figure of merit computed by the physical design evaluator is also expressed in units of page accesses; other kinds of costs (e.g., for storage of indices) are converted into this scale.
2. The major information needed by the transaction cost estimator in determining the cost of processing a retrieval is the average selectivity of each field that is used in the predicate and that is indexed in the proposed design. This information can readily be obtained by observing the intermediate results computed by the transaction processor while handling retrieval requests.
3. The relevant usage pattern information is easy to capture during transaction processing.
4. The techniques underlying the cost estimator have been applied to an operational inverted file system, with encouraging results. Using appropriate selectivity information, estimates are produced that approximate very closely the number of page accesses actually performed by the system in processing transactions.
5. The design proposer employs a set of heuristics that enable it to send a relatively small set of candidate designs to the evaluator for analysis. This heuristic proposer has been tested against an exhaustive proposer for a wide range of usage patterns and data characteristics. In virtually all cases, the heuristic proposer finds the optimal design; in the remaining instances, its selection has a total cost within five percent of the true optimum. The number of evaluator calls that the heuristic proposer makes is roughly proportional to the number of fields in the file, compared to an exponential number for the exhaustive version.

SUMMARY

We have presented the principles of a novel approach to the design of self-organizing, adaptive data base systems. This approach differs from earlier efforts in the field by its use of detailed information on data base usage, its concern with evolving trends in the usage pattern, its transaction by transaction analysis of every proposed design, and its reliance on heuristics to synthesize a small set of candidate designs. The keystone of a system built on these principles is a transaction cost estimator, which assesses the cost of performing any specified transaction in the context of a proposed design. An information gathering module acquires sufficient knowledge about the contents of the data base to enable the cost estimator to operate; it also records the global usage pattern of the data base. At each redesign point, all this information is projected into the future, so that the design should be matched to developing requirements. The design evaluator combines the cost figures associated with the transactions in the usage pattern with general maintenance and storage costs to achieve a unified figure of merit for the design. Both the design proposer and the design evaluator make use of heuristics: the former to guide its search through the space of potential designs and thereby restrict the number of calls on the evaluator, and the latter to coalesce transaction classes and reduce the number of calls on the estimator.

These principles have been applied to the design of a system that selects secondary indices for an inverted file DBMS. A transaction cost estimator has been implemented that is efficient and also quite accurate in its forecasts of page accesses; the information that it requires can readily be gathered during normal transaction processing; and its associated heuristic-based proposer performs virtually as well as its exhaustive counterpart, but at dramatically reduced cost. This facility is being extended to address such issues as the selection of sort keys and file partitioning.

REFERENCES

1. Cardenas, A. F., "Evaluation and Selection of File Organization—A Model and System," *CACM*, 16, 9, Sept. 1973, pp. 540-548.
2. Teorey, T. J., and K. S. Das, "Application of an Analytical Model to Evaluate Storage Structures," *Proceedings of the 1976 SIGMOD International Conference on Management of Data*, Washington, D.C., June 1976, pp. 9-19.
3. Brown, R. G., *Smoothing, Forecasting, and Prediction of Discrete Time Series*, Prentice Hall, 1962.
4. Yao, S. B. and A. G. Merten, "Selection of File Organization Using an Analytical Model," *Proceedings of the International Conference on Very Large Data Bases*, Framingham, Mass., September 1975, pp. 255-267.
5. Stocker, P. M. and P. A. Dearnley, "A Self Organising Data Management System," in *Data Base Management*, edited by J. W. Klimbie and K. L. Koffeman, North Holland, 1974.
6. Hammer, M. and A. Chan, "Index Selection in a Self-Adaptive Data Base Management System," *Proceedings of the 1976 SIGMOD International Conference on Management of Data*, Washington, D.C., June 1976, pp. 1-8.
7. Hammer, M. and A. Chan, "Acquisition and Utilization of Access Patterns in Relational Data Base Implementation," in *Pattern Recognition and Artificial Intelligence*, edited by C. H. Chen, Academic Press, 1976.

Overview of the military computer family architecture selection

by WILLIAM E. BURR and AARON H. COLEMAN
US Army Electronics Command
Ft. Monmouth, New Jersey

and

WILLIAM R. SMITH
Naval Research Laboratory
Washington, DC

ABSTRACT

This paper presents an overview of the selection process employed to choose a single Computer Family Architecture (CFA) to be used in a new Military Computer Family (MCF) intended for use in Army and Navy Systems. A joint Army/Navy Selection Committee studied the suitability of a number of architectures, and intensively evaluated three "final candidate" architectures, before selecting one, the PDP-11, for use with the MCF.

INTRODUCTION

Since early 1975 the Center for Tactical Communications Sciences (CENTACS) of the Army Electronics Command, and the Naval Research Laboratory have been engaged in a cooperative project to develop a software compatible family of military computers, based upon a common architecture, and suitable for a wide range of military land, sea, and air applications. That project is known as the Military Computer Family (MCF) Project, and the computer architecture to be used by the MCF is known as the Computer Family Architecture (CFA).

The MCF is based upon a strategy that included the following:

- Selection of architectural design or designers unbundled from implementation or implementers.
- Standardization on architecture design as the foundation on which software investment is made.
- Consideration of commercially successful architectures for which software already exists as candidates for DOD adoption.
- Technology independence, that is the anticipation of multiple implementations of the same architecture, implementations which might differ in technology (e.g., semiconductor vs magnetic memory), environmental specifications (e.g., volume or power constraints), or reliability assurance (e.g., MIL-qualification vs warranties or incentives).
- Multiple sources of supply for the various processors and other modules of the family.
- Support (probably via emulation) of existing software for the principal existing Army and Navy military computers.

The first step in the development of the MCF was the selection of the architecture to be used. The selection was made by an Army/Navy CFA Selection Committee during the period between October 1975 and August 1976 as a result of evaluating and comparing candidate architectures. The CFA committee began with the initial selection of nine candidate architectures, narrowed the initial set to three finalists: the IBM S/370, the DEC PDP-11, and the Interdata 8/32, and finally chose the PDP-11. This paper discusses the basic premises of the CFA selection process, and summarizes the actions of the Selection Committee.

WHAT IS A MILITARY COMPUTER?

Computers are used in the DOD for a wide range of applications. Many administrative, research and laboratory applications are run on the same commercial computers which are used in industrial and business applications. Many military computer applications, however, require "militarized" computers, which can operate in battlefield, shipborne, and airborne environments, and survive exposure to severe shock, vibration, radiation, and thermal stress. The applications for these computers are usually similar to the applications of commercial OEM computers, that is they are usually embedded in some larger system, such as a missile system, or a radar, and the computer itself is just one component, and not necessarily the most important component, of a larger system.

For the purposes of this article and the five articles which follow it, it is the hostile physical environment—not the types of computations, response times, data rates, or throughput requirements—which fundamentally distinguish the commercial OEM application/computer from the military application/computer.

RATIONALE FOR A COMPUTER FAMILY ARCHITECTURE (CFA)

The Army and Navy currently use and maintain an inventory of over one hundred different computer types. Practically all of these machines have a design personality which must be catered to through specialized software and maintenance support. This inventory is regularly justified as the only means of applying processing capability where it is needed with a minimal cost investment. That is, off-the-shelf procurement or specialized designs aimed at specific operational applications is deemed to be the only satisfactory way of meeting the wide range of speed, power, weight, size, etc., requirements imposed by these applications.

There is little or no argument against satisfying a multitude of environmental and processing speed requirements through a combination of machines of varying hardware technological characteristics. Physical constraints ranging over orders of magnitude leave little choice but to meet these head-on with suitable device technologies, if cost-effective weapons systems are to be put into operation. The current proliferation of computer types is more a de facto result of platform and project managers trying honestly to choose the most appropriate machines out of a sea of unrelated available computer types, than a result of uninformed procurement practices. Moreover, platform and project managers face heavy pressures to locally optimize the costs and schedules of their own projects and relatively little direct pressure to reduce the long term life cycle costs of both hardware and, particularly, software.

The greatest penalties arising out of such proliferation are in the efficiency, timeliness and both non-recurring and recurring costs of system software. However, it is not necessary for differences in computer technology requirements to mandate differences in software characteristics. A number of examples of commercial capitalization on this principle are well known—the IBM 360/370 and the Digital Equipment Corporation PDP-11 product lines. What has been gained from this approach is a line of computers of varying processing capabilities but which are software compatible and enjoy the support of a common set of system and applications programs. The main goal, then, of the Computer Family Architecture is to provide the Army and the Navy with a design for a series of computers (a family) with the variety of members necessary to satisfy the requirements of various platform and battlefield applications, while at the same time providing a single software system capability which will serve each and every member of that computer family.

Central to the success of the Computer Family is the selection and precise specification of the family architecture. The term "computer architecture" means quite different things to different people, so a definition is necessary. Here we follow the example of S/360. In an introductory paper on the IBM S/360, Amdahl, et al.,[1] defined computer architecture as: "The term architecture is used here to describe the attributes of a system as seen by the programmer, i.e., the conceptual structure and functional behavior, as distinct from the organization of the data flow and controls, the logical design, and the physical implementation."

This definition of architecture specifically excludes details of hardware implementation. The instructions and registers which programmers "see" are part of the architecture, but the data buses are not. For example, the IBM 360/30 used 8 bit data paths, the 360/40 used 16 bit paths, and the 360/50 used 32 bit paths, but all three are the same architecture, and can execute the same programs. As another example, the PDP-11 Unibus is not an integral part of the architecture (indeed PDP-11's have been built with at least three different bus structures), but the use of dedicated memory locations for communication with I/O devices is an architectural feature, because the programmer does not see the unibus, but he does see the decicated I/O registers.

Selection of the CFA was guided by the principle that the bulk of computer processing improvements over the last two decades have risen out of technology advances rather than out of architectural changes, and that this principle is likely to remain in effect for at least another generation or more of computer systems. It is more promising, then for the Army and Navy to adopt an already successfully demonstrated extant computer architecture, commercial or military, and to use that architecture to reap the benefits of technology advances while enjoying the benefits of software stability. The selection of an existing architecture carries with it an understanding of the strengths and weaknesses of that architecture and also a useful inventory of support and applications software already developed.

OTHER LEVELS OF STANDARDIZATION

The MCF has chosen to standardize at the instruction-set level. Many other levels of standardization are conceptually possible. One attractive alternative might be to standardize on a single Higher Order Language. There are efforts under way to fix a single HOL for DOD, and these efforts do not conflict with the MCF approach. However, there are a number of problems with this approach:

- HOL's are much more complex than instructions sets. Consequently no two different compilers for the same language have ever been truly compatible, as anyone who ever tried to convert any large set of programs from one "standard" FORTRAN or COBOL compiler to another will attest. Differences between supposedly compatible compilers becomes very difficult to resolve when the underlying data types of the host architectures differ.

- No one existing HOL seems satisfactory for all, or even most, tactical military applications.
- Some level of machine or assembly language programming will doubtless be with us for some time. Although the desirability of using HOL's in place of assembly language is recognized in DOD, the time and space constraints placed upon some critical functions will probably force the continued use of assembly level languages for the foreseeable future.

Another alternative might be to standardize on the assembly language, rather than the instruction set. An example of such standardization is found in the Interdata Family of 16 and 32 bit computers (5/6, 6/6, 7/32 and 8/32) which have similar, but not identical 16 and 32 bit architectures, and which rely on a Common Assembly Language and a "smart" assembler program to resolve the differences. This approach is attractive, but requires that the underlying architectures be quite similar. In particular, programs that are to be transferable from one architecture to another have to be written to avoid incompatabilities and thus it may not be possible to take full advantage of either.

A third alternative would have been to settle upon a standard "micro architecture." This approach assumes that all future military computers will be microcoded (not an unlikely assumption), and asserts that it is the internal register-level architecture of the processor, and the microcode which should be standardized. This approach, and the use of read/write microstore, would permit application-tailored macro instruction-sets. This approach, however, has a number of disadvantages, including the following:

- The micro-architecture of a computer is much more directly related to the performance capabilities of the computer, than is the instruction set. It is not clear that a single micro-architecture can effectively satisfy a wide range of performance requirements.
- Micro-architectures are closely related to component technology and hardware design. Since rapidly improving device technology is the driving force in the computer industry, it would be a mistake to select a micro-architecture which is more or less closely tied to contemporary technology.
- The configuration control of the firmware needed to implement a number of tailored instruction sets, or user developed microcode, and of compilers and operating systems for different tailored macroinstruction sets, promises nightmare-like problems in the diverse environment of military laboratories, system centers, project and platform managers, and system developers (contractors).

Standardization at the computer architecture level is the safe, proven and accepted approach. It is the only answer to complete software transportability across a wide range of computer implementations, which has stood the test of time in industry-wide applications. The success of the IBM S/360 and S/370 families, the PDP-11 family, and several other instruction set compatible families, have demonstrated the practicality of this approach. Amdahl and National Semiconductor have proven that independent manufacturers can build compatible versions of a well-defined architecture (the S/370). The CFA approach is based upon the premise that the Army and Navy should try to take maximum advantage of existing commercial technology, rather than try to push it in new directions.

THE SELECTION COMMITTEE

The first task of the Army/Navy cooperative effort was the selection of the computer family architecture to be used. The Naval Research Laboratory (NRL) led this effort for the Navy under the sponsorship of the Naval Air Systems Command. NRL and CENTACS agreed to perform this task.

In order to achieve a wide representation of military computer requirements in this effort, letters were sent to Army and Navy Laboratories, System Centers, and Project Managers inviting them to nominate "candidate" architectures, and to participate in the selection process as members of the CFA Selection Committee. Ten Army and seventeen Navy organizations assigned representatives to participate in the Selection Committee.

The Army and Navy cooperative effort has been entirely voluntary, and was not imposed upon the Army and Navy by DOD. It resulted from the discovery that both the Army and Navy independently had similar efforts under way, from the belief that military data processing requirements in the three services are similar, and from the realization that the combined funding and application bases of the Army and Navy would enhance the success of such a program. Air Force observers have attended the Selection Committee Meetings, and are participating in the next phase of the MCF project, involving systems implementation, or product planning.

SUMMARY OF THE SELECTION PROCESS

The CFA Selection Committee held five meetings between 1 October 1975 and 26 August 1976. The procedure developed by the committee for selecting the architecture is depicted in Figure 1 and may be summarized as follows:

a. *Select Initial Candidates*—The Committee approved a list of nine candidate architectures for examination. Table I shows nine candidate architectures.
b. *Establish Initial Ranking Procedure*—The Committee developed a set of "absolute" and "quantitative" criteria as measures of computer architecture effectiveness for a wide range of military computer-based systems applications.
c. *Evaluate the Candidate Architectures*—Subcommittees were established to evaluate each candidate architecture in accordance with the established absolute

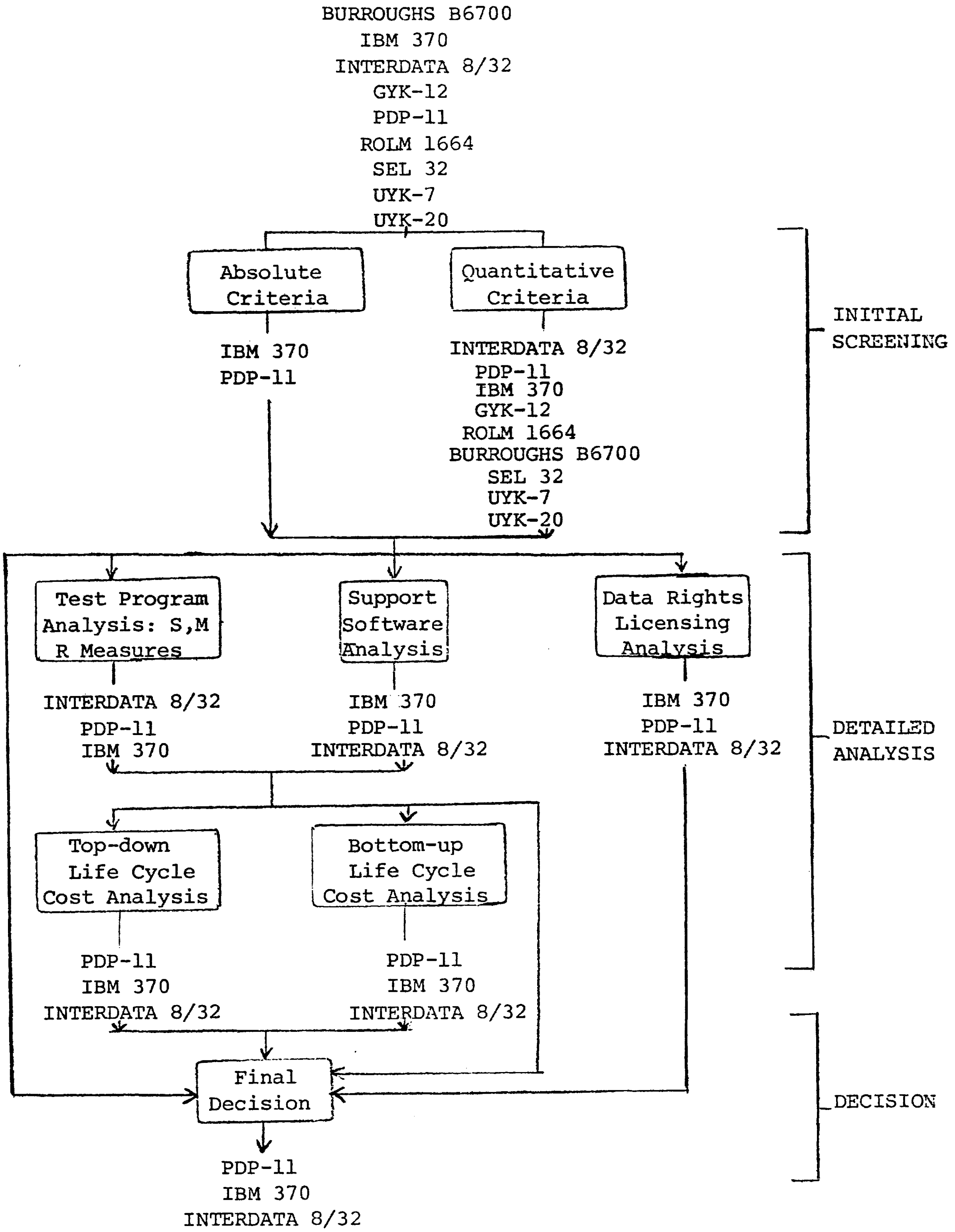


Figure 1—CFA selection process

TABLE I—CFA Candidate Scores on Absolute and Quantitative Criteria

ARCHITECTURE	QUANTITATIVE CRITERIA	ABSOLUTE CRITERIA
INTERDATA 8/32	1.68 (BEST)	Problem with interrupts and traps
PDP-11	1.43	Passed all
IBM S/370	1.36	Passed all
AN/GYK-12	.94	Failed floating point
ROLM/NOVA	.92	Failed virtual memory mapping and interrupts/traps
B6700	.91	Failed protection
SEL-32	.86	Failed virtual memory mapping
AN/UYK-7	.46	Failed floating point
AN/UYK-20	.44 (WORST)	Failed protection

and quantitative criteria. Table I shows the list of nine candidate architectures and their relative performance in this evaluation. Fuller, Stone and Burr[2] describe the criteria and the evaluation process in detail in their paper.

d. *Selection CFA Finalists*—The Selection Committee reviewed the architecture evaluations in detail, and selected three candidate architectures: the IBM S/370, the DEC PDP-11, and the Interdata 8/32 as CFA finalists for further examination.
e. *Describe the Finalists in ISP*—The three final candidate architectures were described in a formal register transfer language, ISP. These ISP descriptions were used to simulate the candidate architectures, and collect the data required for the test program evaluation. Barbacci, Siewiorek, Gordon, Howbrigg, and Zuckerman[4] describe the use of ISP.
f. *Test Program Evaluation*—Just over 100 test program "kernels" were coded by 16 programmers to evaluate the relative efficiency of the three final candidates. The results of this evaluation are summarized in Table II. Fuller, Burr, Shaman, and Lamb[3] describe the test program evaluation in their paper.
g. *Support Software Base Evaluation*—A subcommittee was formed to evaluate the support software bases of the three final candidates. The results of this evaluation are summarized in Table II, and Lieblein, Wagner and Stone[5] describe this evaluation.
h. *Life Cycle Cost Analysis*—A subcommittee was formed to evaluate comparative life cycle costs of the MCF for each of the three final candidates. Two different analyses were performed, one using a "top-down" model and the other using a "bottom-up" model. These evaluations are described by Cornyn, Coleman, Smith and Svirsky.[6]
i. *Licensing*—A series of meetings were held with the manufacturers of the final candidates to establish proposed licensing arrangements for the CFA finalists. Due to the confidentiality of the licensing discussions, they are not reported on here, but they were a significant factor in the final selection.
j. *Final CFA Selection/Recommendation*—All the data acquired in the preceding steps was reviewed and the Committee voted the relative ranking of the CFA finalists.

RESULTS

The Selection Committee held its fifth and final meeting on 24-26 August 1976 at the Naval Underwater System Center, Newport, R. I., for the purpose of selecting the recommended architecture for the MCF. At that meeting the data discussed in the preceding sections of this report was considered at length. The data considered in that discussion is summarized in Table II.

Based upon the data presented in Table II, and upon other concerns specifically considered by the Committee during its discussion of the final selection, the respective strengths and weaknesses of each architecture can be summarized as follows:

a. INTERDATA 8/32. The 8/32 was the highest rated architecture on the Quantitative Criteria, and the Test Program results. The 8/32 has a good interrupt structure for real-time processing. On the other hand, the software base is relatively weak, which consequently compromised its performance in the life cycle cost evaluations. There was a nagging question about how well the state of the machine was preserved after interrupts.
b. IBM S/370. The strongest virtue of the S/370 is its large support software base. The S/370 performed well on the life-cycle cost analyses under assumptions of maximum relative cost of software development. The S/370 is the only architecture demonstrated as an easily virtualized computer in a standard product line.

TABLE II—Final CFA Selection Data Summary

	QUANTITATIVE CRITERIA SCORES	TEST PROGRAM RESULTS		
		R	M	S
8/32	1.68	.83	.85	.83
PDP-11	1.43	.94	.93	1.00
S/370	1.36	1.29	1.27	1.21
	SEE NOTE 1 BELOW	SEE NOTE 2 BELOW		

	SUPPORT SOFTWARE VALUE CURRENTLY AVAILABLE	DEFICIENCY
8/32	$15.3M	$25.9M
PDP-11	$22.2M	$19.1M
S/370	$32.3M	$9.6M

	TOP DOWN COST MODEL (SEE NOTE 3) RELATIVE COST (S/370=1)	BOTTOM UP COST MODEL FOR 1985 RELATIVE COST (S/370=1)	# SYSTEMS PREF. (SEE NOTE 4)
8/32	1.23	1.00	—
PDP-11	1.07	.87	14.5
S/370	1.00	1.00	0.5

NOTE 1: These are relative values with an average value of one for nine CFA candidates. The *higher* the value, the better the architecture.

NOTE 2: These are relative values with an average value of one for the three CFA finalists. The lower the value, the better the architecture. S is a measure of the program storage required for a test routine by each architecture. M is a measure of the processor/memory bandwidth to run a test programs, i.e., # bytes transferred between processor and memory. R is a measure of the internal processor bandwidth required to run a test program.

NOTE 3: Top down cost model analyzed variation of DOD computer resource budget as a function of the selected CFA.

NOTE 4: Bottom up cost model analyzed variation of life cycle cost of 15 tactical data systems as a function of the selected CFA. Relative cost refers to total life cycle cost of all 15 systems. # Systems preferred indicates how many systems would have selected the CFA on the basis of life cycle cost.

On the other hand, its interrupt structure was considered cumbersome for real time control applications. The test program results indicate that the architecture is significantly less efficient than the 8/32 and the PDP-11; this compromised the S/370's performance in the life cycle cost evaluations. There was also concern that small subset versions might not prove cost-effective for low-end applications, and that there was insufficient experience with the S/370 in OEM type applications.

c. PDP-11. The PDP-11 enjoys a good support software base, performed relatively well on the test program evaluations, and has a good interrupt structure for real-time control applications. It enjoys a slight advantage on the cost models for a range of reasonable assumptions. Small scale (microprocessor) implementations are practical and have been built. On the negative side, the 16 bit virtual address space is a limitation and it may be expensive to add a virtual machine capability to the architecture.

The Committee made four final recommendations:

a. The DEC PDP-11 was determined by a vote of 14 to 4 to be the most advantageous architecture for the MCF, the IBM S/370 was ranked second, and the Interdata 8/32 was ranked third.
b. The committee unanimously agreed that a single instruction-set architecture should be selected for the MCF, that the selection of only one architecture is more important than which one of the candidates is selected, and that any one of the three final candidate architectures could provide a satisfactory basis for the MCF.
c. The committee agreed that an effort should be made to relieve the limitations of the selected architecture. In the case of the PDP-11 the major limitation is the small (16 bit) virtual addresss space.
d. A single organizational structure must be established to control the architecture, or major incompatibilities between different implementations will surely result.

REFERENCES

1. Amdahl, G. H., G. A. Blaauw, and F. P. Brooks, "Architecture of the IBM System/360," *IBM Journal of Research and Development,* Vol. 8, April, 1964, pp. 87-101.
2. Fuller, S. H., H. S. Stone, and W. E. Burr, "Initial Selection and Screening of the CFA Candidate Computer Architectures," *AFIPS Conference Proceedings,* Vol. 46, 1977 National Computer Conference.
3. Fuller, S. H., W. E. Burr, P. Shaman, D. A. Lamb, "Evaluation of Computer Architectures via Test Programs," *AFIPS Conference Proceedings,* Vol. 46, 1977 National Computer Conference.
4. Barbacci, M. R., D. P. Siewiorek, R. Howbrigg, R. Gordon, S. Zuckerman, "An Architectural Research Facility: ISP Descriptions, Simulation, Data Collection."
5. Wagner, J., E. Lieblein, J. Rodriguez, and H. Stone, "Evaluation of the Software Bases of the Candidate Architectures for the Military Computer Family," *AFIPS Conference Proceedings,* Vol. 46, 1977 National Computer Conference.
6. Cornyn, J. J., W. R. Smith, W. R. Svirsky, and A. H. Coleman, "Two Life Cycle Cost Models for Comparing Computer Architectures," *AFIPS Conference Proceedings,* Vol. 46, 1977 National Computer Conference.

Initial selection and screening of the CFA candidate computer architectures

by SAMUEL H. FULLER,
Carnegie-Mellon University
Pittsburgh, Pennsylvania

and

HAROLD S. STONE,
University of Massachusetts
Amherst, Massachusetts

and

WILLIAM E. BURR
US Army Electronics Command
Ft. Monmouth, New Jersey

ABSTRACT

The initial selection criteria that were developed and used by the Army/Navy Computer Family Architecture (CFA) committee in their evaluation of alternative computer architectures is presented in this article. These initial criteria were used in this first phase of the CFA evaluation process to reduce the number of computer architectures from the original set of nine to the most promising three or four architectures for the more intensive evaluation discussed in the companion articles.[4,6,8] The machines selected by this initial ranking and screening process for further evaluation were the Interdata 8/32, DEC PDP-11, and the IBM S/370.

INTRODUCTION

The CFA selection committee was concerned with selecting a computer architecture to use in future military (ruggedized) computers and hence wanted to evaluate the merits of the computer architecture independent of any features, or flaws, of existing implementations of the computer. For this reason, the following definition of computer architecture was used by the CFA committee:

> *Computer Architecture:* The structure of the computer a programmer needs to know in order to write any machine-language program that will run correctly on the computer.

With a well specified architecture, details of data bus width, technology (core memory versus semiconductor memory, TTL versus ECL circuits), implementation speed-up techniques, physical size of computer, etc., need not be of concern to the programmer and hence are not a part of the architecture. This separation of architecture and implementation is not a radically new idea.[1] The IBM System/360-370 series, the DEC PDP-11 series, and the Data General NOVA series are just three examples of where this has already been successfully accomplished to a greater or lesser degree.

This article first describes how the CFA selection committee chose the initial candidate architectures for evaluation, and then describes the criteria, the methodology, and the data used in ranking these architectures during the preliminary screening phase of the CFA project. At the point this procedure was formulated, it was known that time and money limitations would preclude doing a detailed analysis on all nine candidates; consequently an initial screening was necessary to limit the field to the three or four "best" candidates that would be subjected to a much more detailed analysis. This more detailed analysis, based on test programs, the support software bases of the architectures, and life cycle cost models is discussed in the accompanying articles.

Many detailed questions arose during the evaluation of these nine initial candidate architectures. It is impossible to review all these questions in this article, but we will discuss here the most important questions that arose, and interested readers are encouraged to refer to Volume II of the final report of the CFA committee for a detailed presentation of how and why each candidate architecture was evaluated as it was.[3]

The mechanism for choosing the nine initial candidate architectures is discussed in the next section. The third and fourth sections then describe the nine absolute and seventeen quantitative criteria, respectively, and show how each of the candidate architectures was ranked on these criteria. The fifth section describes how the CFA committee combined the scores of the candidate architectures for each individual criteria to form a single, composite score for each architecture that reflected the relative importance of the seventeen quantitative criteria.

INITIAL SELECTION OF CANDIDATE COMPUTER ARCHITECTURES

The CFA selection process was initiated in March and April of 1975 when letters were sent to 35 Army and Navy organizations soliciting proposals for candidate computer architectures. As a result of these letters, and discussions at the first two CFA meetings, the following set of nine computer architectures was chosen:

Burroughs 6700
DEC PDP-11
IBM System/370
Interdata 8/32
Litton AN/GYK-12
ROLM Corporation 1664 (AN/UYK-28)*
SEL 32
Univac AN/UYK-7
Univac AN/UYK-20

There were on the order of 100 viable computer architectures in 1975 that might have been considered by the CFA committee for selection.[2] The decision as to what set of architectures would be evaluated remained open from March through December of 1975. The nine architectures listed above were selected for evaluation because they met two essential criteria: (1) the CFA committee agreed the architecture might be a reasonable choice for future military computers, and (2) there was a CFA committee member sufficiently convinced of the value of the computer architecture that he was willing to act as its advocate in the subsequent evaluation phase.

ABSOLUTE CRITERIA

The CFA selection committee specified nine *absolute criteria* that they felt a candidate computer architecture needs to satisfy if it is going to meet the requirements of future military computer systems. All the absolute criteria (with the exception of the subsetability criterion) had to be satisfied by an implementation of the architecture which was operational by 1 January 1976. This eliminated speculative decisions based on promises or potential solutions that looked inviting, but might not come to fruition. Failure to satisfy any absolute criterion resulted in the elimination of the architecture from further consideration. The nine absolute criteria are given below. The formal statement of each criterion is underlined, while explanations and examples are not underlined. Many of the comments that follow the definition of an absolute criteria are the result of the experience gained when the CFA committee evaluated the nine candidate architectures against these criteria.[7] Table I shows which absolute criteria each candidate architecture passed or failed.

Virtual memory support

The architecture must support a virtual to physical address translation mechanism. The intent of this criterion is to take advantage of the widely used feature of many machines that is known as virtual memory. Many advantages accrue to architectures that support virtual address translation mechanisms, the most notable of which is the ability to simplify programming by freeing the programmer of explicit management of his primary memory and providing a mechanism for keeping only the active portions of a program in high-speed memory.

The answers for this criterion listed in Table I are not controversial, except for the AN/UYK-20. This architecture provides the page registers necessary for relocation, but does not limit the ability to change these registers to privileged programs. Some members of the CFA committee felt that preventing user state access to the page registers was a necessary aspect of virtual memory; others disagreed. The full CFA committee voted to fail the AN/UYK-20 on this criteria. The ROLM 1664 and SEL 32 both failed this criterion because each of these architectures provide a mechanism commonly known as "bank switching," which the committee felt was not an adequate memory translation mechanism.

Protection

The architecture must have the capability to add new, experimental (i.e., not fully debugged) programs that may include I/O without endangering reliable operation of existing programs. The intent of this criterion is to provide a mechanism in the hardware for aiding software development, and for preventing certain catastrophic software failures from occurring in the field. Architectures that use a

* The AN/UYK-28 is instruction-set upward-compatible with the Data General NOVA computer architecture. Other ROLM computers that are also compatible with the NOVA architecture are the AN/UYK-19 and AN/UYK-27. The AN/UYK-28 is incompatible with the Data General ECLIPSE computer architecture, Data General's upward-compatible extension of the NOVA.

TABLE I.—Candidate Architecture Value for Absolute Criteria

		Candidate Computer Architectures								
	Absolute Criterion	IBM S/370	Inter-Data 8/32	Rolm AN/UYK-28	DEC PDP-11	Univac AN/UYK-7	SEL 32	Burroughs B6700	Univac AN/UYK-20	Litton AN/GYK-12
1	Virtual Memory	Y	Y	N	Y	Y	N	Y	N	Y
2	Protection	Y	Y	Y	Y	Y	Y?	N	N	Y?
3	Floating Point	Y	Y	Y	Y	N	Y	Y	Y	N
4	Interrupts/Traps	Y	?	Y	Y	Y	Y	Y	Y	Y
5	Subsetability	Y	Y	Y	Y	Y?	Y	Y?	Y	Y?
6	Multi Processor	Y	Y	Y	Y	Y	Y	Y	Y	Y
7	I/O Controllability	Y	Y	Y	Y	Y	Y	Y	Y	Y
8	Extensibility	Y	Y	Y	Y	Y	Y	Y	Y	Y
9	Read-Only Code	Y	Y	Y	Y	Y	Y	Y	Y	Y
	SUMMARY	Y	?	N	Y	N	N	N	N	N

Y	Yes, Meets Criteria
N	No, Fails Criteria
Y?	Yes (but with some reservations)
?	Unresolved

privileged mode to protect vital registers and system resources generally meet this criterion.

The AN/UYK-20 failed this criterion because it lacks memory protection; any user can modify the contents of the relocation registers, and thereby read and write any word in memory. Another generic way for an architecture to fail the protection criterion is for a program to have the ability to put the machine into a noninterruptible state for an indefinite time. Architectures that permitted nonterminating instructions were carefully examined to identify if these were, or were not, interruptible.

Floating-point support

The architecture must explicitly support one or more floating-point data types with at least one of the formats yielding more than 10 decimal digits of significance in the mantissa. The significance measure was determined as representative of the most stringent requirements actually encountered.

The AN/GYK-12 failed this criterion because it does not support floating point operations. The AN/UYK-7 failed because it supports a single, 64-bit floating point format with only 31 bits (9.2 decimal digits) of mantissa. Because this is so close to the borderline, one might reconsider requirements on significance to determine how firm the 10 decimal digit criterion is. (Had the AN/UYK-7 looked like an otherwise excellent architecture, it is likely that the committee would have relaxed the floating point absolute criterion for it.)

Interrupts and traps

It must be possible to write a trap handler that is capable of executing a procedure to respond to any trap condition and then resume operation of the program. For example, if the processor receives a page-fault trap from the address translation unit, it must be able to request the appropriate page be brought in from secondary storage and then resume execution. If resumption of execution is logically impossible (e.g., an attempt to store an operand into a read-only segment of virtual memory or fetch an instruction with a parity error) then the trap handler should be able to abort the program with an indicator of which instruction and/or operand caused the termination.

A similar requirement exists for interrupts: the architecture must be defined such that it is capable of resuming execution following any interrupt (e.g., power failure, disk read error, console halt).

Another intent of this criterion is to permit extensions and subsets of an architecture to operate correctly so programs can be upward or downward compatible. The subsets and extensions may differ drastically in size, cost, and performance, but every program written for the native architecture can run on the subset or extended machine.

The Interdata 8/32 had difficulty satisfying this criterion since it has variable length instructions, and there is no way after a trap or an interrupt to tell whether the instruction which was being executed was a 16, 32, or 48 bit instruction. This may be a problem when it is desirable to correct the cause of the fault, and then re-execute (or resume) the instruction. Due to uncertainties in the definition of the Interdata 8/32 architecture, the CFA committee was not able to resolve whether or not the Interdata 8/32 satisfied this criterion.

Subsetability

At least the following components of an architecture must be able to be factored out of the full architecture:

a. Virtual-to-Physical Address Translation Mechanism
b. Floating Point Instructions and Registers (if separate from general purpose registers)

c. Decimal Instructions Set (if present in full architecture)
d. Protection Mechanism

Implementations of the architectures on small machines for dedicated applications must not be required to include features of the architecture intended for use on larger, multiprogrammed, multi-application configurations. Existence of such subsets did not have to be demonstrated in an operational implementation of the architecture.

Because there was no operational method for testing subsetability, we could not challenge positive replies for any of the nine candidate architectures. However, the B-6700 and the AN/UYK-7 have not been subsetted in the sense of the criterion, so that their subsetability is more speculative.

In order to retain program compatibility across the implementations of the architecture, this criterion was extended to include the following requirement: The trap mechanism of the architecture must be defined such that instructions in the full architecture, but not implemented in the subset machine, trap on the subset machine and that it be possible to write trap routines for the subset machine that allow it to interpretively execute those instructions not implemented directly in hardware (or firmware) and then resume execution. (This is an elaboration of absolute criterion 4.)

Multiprocessor support

The architecture must support some form of "test-and-set" instruction to allow for the communication and synchronization of multiple processors. The intent of this criterion is to be sure that the basic architecture can support multiprocessor configurations.

Input/output controllability

A processor must be able to exercise absolute control over any I/O processor and/or I/O controller. The interpretation of the criterion proved rather difficult. While all architectures necessarily permitted individual devices to be started and queried for status, there were varying degrees of control exercisable with respect to stopping the devices. It is reasonable to stop all input/output, or to stop selected devices. All architectures had some way of stopping a single device and stopping all devices, but how they did it varied widely in efficiency.

Extensibility

The architecture must have some method for adding instructions to the architecture consistent with existing formats. There must be at least one undefined code point in the existing opcode space of the instruction formats. All nine candidate architectures have unused instructions, so all passed this criterion.

Read-only code

It must be possible to execute programs from read-only storage. This criterion is intended to permit an added degree of reliability by permitting programs to be stored in a nonvolatile read-only memory. However, a program can be rewritten to be read-only on any of the nine architectures, even if that architecture does not support special types of instructions to facilitate this. It might have been more meaningful to examine this question quantitatively.

Table I shows the score of each candidate architecture on each of the absolute criteria. Note that none of the nine architectures failed to meet the last five criteria: subsetability, multiprocessor support, I/O controllability, extensibility, and read-only code. This is in part the case because we limited our evaluation to reasonably successful architectures, but is partly the result of not defining these criteria precisely enough prior to applying them to the candidate architectures. For example, by not clearly defining how to test for the practical subsetability of an architecture, we made it virtually impossible for an architecture to fail this criterion. Subsequent studies would be well advised to consider more precise definitions of these (and any additional) absolute criteria before evaluating alternative architectures against them.

QUANTITATIVE CRITERIA

In addition to the absolute criteria, the CFA committee specified seventeen quantitative criteria that they felt would be helpful in the initial screening process. A number of these quantitative criteria measure attributes of a computer architecture better measured by benchmarks, or test programs.[4] However, the CFA committee recognized that it did not have the resources to run benchmarks on all nine candidate architectures and therefore proceeded with the use of these quantitative criteria to help select three or four candidate architectures, out of the original nine candidate architectures, for more intensive study via test programs.

The quantitative criteria are described below and the score of each architecture on the quantitative criteria is given in Table II.

Virtual address space

V_1: The size of the virtual address space in bits.
V_2: Number of addressable units in the virtual address space.

Two aspects of these measures were open to interpretation. The CFA committee settled on the following interpretation for treating bank switching: the virtual address for a machine with bank switching is the address within a bank. The effect of bank switching is to increase the size of the physical rather than the virtual address.

The second interpretation centered on the notion of "addressable unit." There are several degrees of addressa-

TABLE II.—Candidate CFA Values for Quantitative Criteria

		Candidate CFA's								
#	Quantitative Criteria	IBM S/370	Inter- Data 8/32	Rolm AN/UYK-28	DEC PDP-11	Univac AN/UYK-7	SEL 32	Burroughs B6700	Univac AN/UYK-20	Litton AN/GYK-12
1	V_1**	27	27	20	20	24	22	24	20	20
2	V_2**	27	27	20	19	24	22	20	17	20
3	P_1**	27	27	22***	25	23	26***	24	20	29
4	P_2**	27	27	22***	24	23	26***	20	17	29
5	U	.371	.355	.039	.043	.15	.450	.019	.125	.219
6	CS_1	1344	1632	1008	1168	992	304	306	1328	1008
7	CS_2	576	576	112	144	448	288	204	336	752
8	CM_1	3168	1120	1882	736	1472	768	408	2256	1344
9	CM_2	1312	32	544	480	1472	704	408	720	1088
10	K	1	0	0	1	0	0	0	0	0
11	B_1	17,300	185	13,800***	14,700	346	75	90	400	30
12	B_2*****	16,000	14	169	311	147	23	207	8	6
13	I	64	16	48	16	128	64	169	80	32
14	D	15	27	20	19	18	22	18	20	20
15	L	6192	560	114	112	2112	288	255	—	1376
16	J_1	1904	2368	1360	1040	1280	960	459	1408	1344
17	J_2	1136	1280	320	400	1280	960	459	640	1088

** These values are of the form 2^x where x = indicated data except for B6700 which is of the form $3(2^x)$.
*** With memory bank switching. **** Includes Novas. ***** Millions of dollars.

bility. An item may be fully addressable in the sense that it can be accessed by the address produced by an effective address computation. The committee also decided, however, that instructions such as the IBM S/370 Test Under mask, and the OR Immediate allowed the testing and setting of individual bits, and provided a minimum addressable unit of 1 bit.

Physical address space

P_1: The size of the physical address space in bits.
P_2: The number of addressable units in the physical address space.

Where bank switching has been implemented, the physical address measures include all the banks of memory available. For computers with virtual address translation, the physical address is the address resulting from the virtual-to-physical address translation. The physical address space is defined apart from any implementation, since the physical address space size is defined by the effective address calculation process or the virtual address translation process and need not be equal to the largest memory configuration yet delivered.

Fraction of instruction space unassigned

It is important to select an architecture that will allow reasonable growth over its expected lifetime. Let U be defined as the fraction of the instruction space in the architecture that is unassigned. Specifically:

$$U = \sum_{1 \leq i < \infty} u_i 2^{-i} \tag{1}$$

where u_i is the number of unassigned instructions of length i.

Size of central processor state

The amount of information that must be stored or loaded upon interrupt and/or context swapping is clearly an important factor in the response of real time systems and in the overhead of multiprogramming systems. Let the processor state be defined as all the bits of information in a processor that must be saved in order to be able to restart an interrupted process at a later date. Processor states normally include the accumulators, index registers, program counter, condition codes, memory mapping registers, interrupt mask registers, etc.

C_{s_1}: The number of bits in the processor state of the full architecture.
C_{s_2}: The number of bits in the processor state of the minimum subset of the architecture (i.e., without Floating Point, Decimal, Protection, or Address Translation Registers).
C_{m_1}: The number of bits that must be transferred between the processor and primary memory to first save the processor state of the full architecture upon interruption and then restore the processor state prior to resumption. This measure differs from

C_{s_1} above in that "register bank switching," where provided for in the candidate architectures, may eliminate the need to save some registers in primary memory, while the instruction fetches required to save the state are included in C_{m_1} but not in C_{s_1}.

C_{m_2}: The measure analogous to Cm1 for the minimum subset of the architecture.

These measures give an approximation to the complexity of the implementation of the architectures, as well as a measure of the responsiveness of the architectures to worst-case context changes for interrupt processing.

If an architecture provides for several sets of certain registers to provide fast switching or multiple contexts, and if a program uses only one such register set when it runs in one context, then only one set of these registers is used in calculating C_{s_1}.

Usage base

B_1: Number of computers delivered as of the latest date for which data exists prior to 1 June 1976.

B_2: Total dollar value of the installed computer base as of the latest date for which data exists prior to 1 June 1976.

These two measures are meant to be approximate indicators of the existing software and programmer experience base. A single individual determined the value of these measures for all candidate architectures from standard sources.

I/O initiation

I: The minimum number of bits which must be transferred between main memory and any processor (central, or I/O) in order to output one 8-bit byte to a standard peripheral device.

Although this measure was intended to give some insight into the responsiveness of an architecture, it is very difficult to construct an interpretation of the measure that serves this purpose well. The measure counts relatively few bits for some architectures, and this, in turn, makes the measure very sensitive to changes of a few bits. The I measure is also sensitive to several assumptions about exactly what actions are to be performed in doing the input/output operation, and where parameters for the operation are found. Unfortunately, this sensitivity made the I measure very arbitrary, and a rather inexact measure of input/output responsiveness. The precise, and somewhat lengthy, definition of I is given in Reference 3.

Virtualizability

K: is unity if the architecture is virtualizable as defined in Reference 5, otherwise, K is zero.

The intent of this criterion is to capture the concept of virtual machines that has been used to advantage in some commercial computer systems (e.g., IBM's VM/370). An architecture that supports virtual machines provides a mechanism for a privileged, stand-alone program to run as an unprivileged task and produce the results identical to those it produces as a privileged program. The importance of this idea is that an operating system can be run in user mode as a subsystem of another operating system.

The definition of virtual machine as provided by Popek and Goldberg in their article in *CACM*[5] is a very strict definition that guarantees that any operating system that can run stand-alone on architecture X, can also run on architecture X in nonprivileged mode. If an architecture fails this definition it may still support virtual machines in a more limited sense.

Direct instruction addressability

D: The maximum number of bits of primary memory which one instruction can directly address given a single base register, which may be used but not modified.

Large displacement fields in instructions generally simplify programming because they reduce the need to set base registers and to maintain addressability. Because an architecture may have several different instruction formats, each with different displacement field formats, the committee required that the format selected for this measure be the one used for standard LOAD and STORE operations, or the equivalent thereof. This eliminated anomalies, like the MOVE CHARACTER LONG in the IBM S/370 architecture, from consideration.

Maximum interrupt latency

Let L be the maximum number of bits which may need to be transferred between memory and any processor (central processor, I/O controller, etc.) between the time an interrupt is requested and the time that the computer starts processing that interrupt (given that interrupts are enabled). This may be interpreted as a measure of the longest noninterruptible instruction or sequence of instructions. Architectures with nonterminating noninterruptible instructions have infinite L measures and are so indicated in Table II.

Subroutine linkage

J_1: The number of bits which must be transferred between the processor and memory to save the user state, transfer to the called routine, restore the user state, and return to the calling routine, for the full architecture. No parameters are passed.

J_2: The analogous measure to S1 above for the minimum architecture (e.g., without Floating Point registers).

This measure gives an indication of the size of overhead that might be encountered in doing subroutine calls in the worst case for the biggest and smallest machines in the family. The bits counted here are related to the count in CS_1, CS_2, CM_1, and CM_2. By presumption, the bits that are stored for J_1 are exactly those for CS_1, except that it is not necessary to save the protection registers, memory map registers, interrupt mask, and other registers that determine the global context for a program. Architectures with small processor states or that have LOAD/STORE MULTIPLE instructions show up well on these measures.

COMPOSITE SCORE OF THE QUANTITATIVE CRITERIA

After applying the quantitative criteria just discussed, the CFA committee had to determine how the performance of the candidate architectures on these criteria would be used to screen out all but three or four of the architectures for further consideration in the test program and software evaluation phases of the study. Clearly, the candidate architectures should be ordered relative to each of the seventeen quantitative criteria and these independent orderings studied to detect weaknesses and strengths of the competing architectures. However, some summary measure was ultimately needed to assist the committee in its selection of the final architectures to undergo more intensive study. A variety of thresholding and weighing schemes were proposed, but the particular scheme that follows was the scheme chosen by the CFA committee.

Relative weighing of criteria

Each voting organization of the CFA committee was given 100 points to distribute among the various measures to indicate their relative importance to the organization. The weight for criterion x, W[x], was defined as the total number of points given criterion x by all the voting CFA organizations, divided by the total number of points handed out. The weights for the quantitative criteria based on responses from 24 voting CFA committee members is given in Table III.

TABLE III.—Quantitative Criteria Composite Weights

Criterion	Army Weights	Navy Weights	Full CFA Committee Weights
V_1	.0412	.0444	.0433
V_2	.0438	.0575	.0529
P1	.0425	.0706	.0612
P2	.0387	.0637	.0554
U	.0513	.0644	.0600
CS1	.0587	.0375	.0466
CS2	.0675	.0219	.0371
CM1	.0700	.0544	.0596
CM2	.0713	.0319	.0450
K	.0500	.0587	.0558
B1	.0450	.0244	.0313
B2	.0200	.0281	.0254
I	.0875	.1419	.1238
D	.0912	.1081	.1025
L	.0812	.0969	.0917
J1	.0637	.0626	.0629
J2	.0762	.0331	.0475

Normalization

When attempting to combine these quantitative measures into a composite measure we faced two problems:

a. The measures are defined such that good computer architectures maximize some measures and minimize others. Specifically, the measures that a computer architecture should maximize are: V_1, V_2, P_1, P_2, U, K, B_1, B_2, and D; while the measures that should be minimized are: CS_1, CS_2, CM_1, CM_2, I, L, J_1, and J_2. Let our composite measure be a maximal measure and transform all minimal measures to maximal measures by taking the reciprocal: X'=1/X.

b. Measures that inherently involve large magnitudes are not necessarily more important than smaller measures. For example, V_1 is on the order of 10^4 to 10^9 while K is either 0 or 1.

To resolve this problem of differing scale, the values for the quantitative criteria were normalized by dividing each value by the average value of the criterion over the set of nine architectures. For example, the nine measures for criteria I are (64, 16, 48, 16, 128, 64, 169, 80, 32), the average value is 68.6, and the normalized measures are (0.93, 0.23, 0.70, 0.23, 1.87, 0.93, 2.47, 1.17, 0.47).

Normalized measures have the attractive properties that they all lie in the range (0,M); have a mean across the set of M architectures of unity; and the standard deviation of the set of normalized measures is in the interval (0, $M^{0.5}$). We could have taken the normalization process a step further and adjusted the spread of each measure so that the measure gave a standard deviation of unity (or some other constant) across the set of architectures being evaluated. We did not do this for all measures. Some measures were better "discrimination functions" than others and we did not want in general to lose this information by further normalization. However, the committee agreed that it is important to normalize the standard deviation of some of the measures; specifically, V_1, V_2, P_1, P_2 and D were normalized to have a mean *and* standard deviation of unity. These measures may differ by several orders of magnitude between candidate architectures, but the CFA Committee did not feel that the utilities, as expressed by the measures, differ by orders of magnitude.

Scaling and composition of the quantitative measures

In order to combine the individual measures the committee used a simple, linear sum of each normalized measure X scaled by its corresponding weighing coefficient W[X]. The weighing coefficients have been defined so that they sum to unity and hence the composite measure A is in fact a normalized measure with a mean of 1. Using the weights given in Table III and the values of the quantitative criteria given in Table II we get the composite measures for the candidate architectures shown below in Table IV.

There was some valid concern by members of the CFA committee about the role of the weighing of the measures, the normalization of the measures, and the measures themselves in the selection of finalists. However, upon detailed examination of the results we found that, given the weights applied by the committee as an indication of the importance of idealized concepts, the finalists selected are very insensitive to the exact details of the selection procedure. Almost any reasonable methodology for measuring the key concepts quantitatively would select the same finalists.

SUMMARY

This article has presented the nine absolute criteria and the seventeen quantitative criteria used by the CFA committee in their initial screening on the initial candidate computer architectures. The scores for each of the candidate architectures are given in Tables I and II for the absolute and quantitative criteria, respectively. Only the IBM S/370 and PDP-11 architectures passed all the absolute criteria. The Interdata 8/32 architecture is not well defined with respect to trap handling and there remains some question as to whether it meets the requirements of the interrupt and trap handling criteria. The remaining six candidate architectures failed one or more of the absolute criteria specified by the CFA committee. A weighing scheme was developed by the CFA committee for the quantitative criteria and the composite scores of the nine candidate architectures are given in Table IV. The quantitative criteria showed that the Interdata 8/32, PDP-11, and IBM S/370 lead the other architectures by comfortable margins. These results were used by the CFA committee to reduce the field of candidate architectures to three finalists—the IBM S/370, the PDP-11, and the Interdata 8/32—for more thorough evaluation.

This article has indicated some of the areas where we had difficulty applying the criteria and the final report of the CFA committee goes into these difficulties, and their resolution, in much greater detail.[3] The fact remains, however, that if we had to compare a set of computer architectures again, we would need to go through a similar "initial screening" process; it is just too costly and time-consuming to expect to be able to evaluate more than a small set of architectures via any more comprehensive means such as benchmarking. The absolute and quantitative criteria used by the CFA committee have the attractive property that they can be determined directly from the definition of the computer architecture (or from a survey of computer installations for criteria B_1 and B_2). Reflecting back on the history of the CFA project, we estimate that it took from two to five man-days to evaluate each of the computer architectures against the criteria discussed in this article, plus a two day meeting of the entire CFA committee to resolve differences of interpretation, and it took from six to nine man-months to evaluate each of the computer architectures via the set of test programs, support software evaluation, and life cycle cost models in the subsequent stages of the CFA project.

TABLE IV.—Ranking Based on the Quantitative Criteria

Architecture	Score
Interdata 8/32	1.68
PDP-11	1.43
IBM S/370	1.36
AN/GYK-12	0.94
ROLM	0.92
B6700	0.91
SEL-32	0.86
AN/UYK-7	0.46
AN/UYK-20	0.44

ACKNOWLEDGMENTS

The criteria and methodology described in this article benefited from the ideas and criticism of many members of the CFA committee. R. Estell, L. Haynes, and N. Tinkelpaugh, as members of the selection criteria subcommittee, each made important contributions to the formulation of the initial screening process.

REFERENCES

1. Amdahl, G. M., G. A. Blaauw, and F. P. Brooks, "Architecture of the IBM System/360," *IBM Journal of R and D* 8, April 2, 1964, pp. 87-101.
2. *Computer Review* (formerly *Computer Characteristics Review*) GML Corporation, Lexington, MA, 02173, 1975.
3. Fuller, S. H., H. S. Stone, and W. E. Burr, "Selection of Candidate Computer Architectures and Initial Screening," Volume II of *Computer Family Architecture Selection Committee Final Report*, Naval Research Laboratory, Washington, D.C. 20375. December 1, 1976.
4. Fuller, S. H., W. E. Burr, P. Shaman, and D. A. Lamb, "Evaluation of Computer Architectures via Test Programs," *AFIPS Conference Proceedings*, Vol. 46, 1977 National Computer Conference.
5. Popek, G. J., and R. P. Goldberg, "Formal Requirements for Virtualizable Third Generation Architectures," *Communications of the ACM*, Vol. 17, No. 7, July 1974, pp. 412-421.
6. Smith, W. R., J. J. Cornyn, A. H. Coleman, W. Svirsky, R. Estell, P. Sabin, "Life Cycle Cost Models for Comparing Computer Family Architectures," *AFIPS Conference Proceedings*, Vol. 46, 1977 National Computer Conference.
7. Stone, H. S., "An Audit of the Selection Criteria for Computer Family Architecture," CFA memorandum, January, 1976. Distributed at the 18-20 February CFA meeting.
8. Wagner, J., B. Lieblein, J. Rodriguez, H. S. Stone, "Evaluation of the Candidate Architectures for the Military Computer Family," *AFIPS Conference Proceedings*, Vol. 46, 1977 National Computer Conference.

Evaluation of computer architectures via test programs*

by SAMUEL H. FULLER, PAUL SHAMAN and DAVID LAMB
Carnegie-Mellon University
Pittsburgh, Pennsylvania

and

WILLIAM E. BURR
U.S. Army Electronics Command
Ft. Monmouth, New Jersey

ABSTRACT

This article presents the evaluation of the Computer Family Architecture (CFA) candidate architectures via a set of test programs. The measures used to rank the computer architectures were S, the size of the test program, and M and R, two measures designed to estimate the principal components contributing to the nominal execution speed of the architecture. Descriptions of the twelve test programs and definitions of the S, M, and R measures are included here. The statistical design of the assignment of test programs to programmers is also discussed. Each program was coded from two to four times on each machine to minimize the uncertainty due to programmer variability. The final results show that for all three measures (S, M, and R) the Interdata 8/32 is the superior architecture, followed closely by the PDP-11, and the IBM S/370 trailed by a significant margin.

INTRODUCTION

While there are many useful parameters of a computer architecture that can be determined directly from the principles of operation manual, the only method known to be a realistic, practical test of the quality of a computer architecture is to evaluate its performance against a set of benchmarks, or test programs. In a previous article,[1] we presented a set of absolute and quantitative criteria that the CFA committee felt provided some indication of the quality of the candidate computer architectures. It is important to emphasize, however, that throughout the discussion of these criteria it was understood that a benchmarking phase would be needed, and that many of the quantitative criteria were being used to help construct a reasonable "prefilter" that would help to reduce the number of candidate computer architectures from the original nine to a final set of three or four. As described in the preceding article, this initial screening in fact reduced the set of candidate computer architectures to three: the IBM S/370, the PDP-11, and the Interdata 8/32.

The concept of writing benchmarks or test programs, is not a new idea in the field of computer performance evaluation and is generally considered the best test of a computer system.[2-4] For the purpose of the CFA committee, we define a *test program* to be a relatively small program (100 to 500 machine instructions) that was selected as representative of a class of programs. The CFA committee's test program evaluation study described here had to address the central problems facing conventional benchmarking studies:

a. How is a representative set of test programs selected?
b. Given limited manpower, how are programmers assigned to writing test programs in order to maximize the information that can be gained?

We faced an additional problem because we evaluated computer architectures, independent of any of their specific implementations. In other words, when evaluating particular computers, time is the natural measure of how fast a test program can be executed. However, a computer architecture does not specify the execution time of any instructions and so an alternative to time must be chosen as a metric of execution speed.

This article explains how the CFA committee addressed the above questions and presents the results of the test program evaluation of the three candidate architectures. The next section describes how the 12 test programs used in the evaluation process were selected. The third section explains the measures of architecture performance that were used in this study. The fourth section explains how 16 programmers were assigned from six to nine programs

* This work was supported in part by the US Army Research Office Grant DAAG29-76-G-0299, and in part by the Advanced Research Projects Agency under Contract F44620-73-C-0074, which is monitored by the Air Force Office of Scientific Research.

each, in order to get a set of slightly over 100 test program implementations that were used to compare the relative performance of the candidate architectures. The principal results of the test program evaluation are presented in the sixth section and Appendix A contains the actual S, M, and R measurements of all of the test programs. For the actual specifications of the test programs, details of the evaluation process beyond the scope of this article, and a chronology of the CFA test program study see Reference 5.

TEST PROGRAM SPECIFICATION

Alternative approaches

A number of alternative test program specifications were considered by the CFA committee. A tempting proposal was to use test programs written in a Higher-Order Language (HOL). This had the advantage of allowing a single HOL source program to be used for all the architectures to be tested. This also would have permitted the use of existing benchmark programs, which were available from several sources (FCDSSA, and NADC), and which were extracted from "real" military systems. One disadvantage of this approach was that no one language, even FORTRAN, was available on all the nine initial candidate architectures and those languages developed for use in tactical military applications (e.g., JOVIAL, CMS-2, CS-4, and TACPOL) were each available on only a few of the candidate architectures. There are FORTRAN IV and COBOL compilers available for each of the three final candidate architectures; however, neither FORTRAN nor COBOL are widely used in tactical military applications. The major disadvantage, however, was that there is no practical way to separate the effects of compiler quality from the effects of architectural efficiency, and the object of the test program study was to measure only the architecture. The results obtained from HOL test programs would necessarily involve a significant undetermined component, which would be due to variations in the efficiency of compilers that are unlikely to be extensively used in tactical military applications, and because these unmeasurable compiler effects might well mask genuine differences in the intrinsic efficiencies of the architectures.

Using standard (Machine-Oriented) assembly language for the test programs was the obvious alternative to the use of Higher Order Languages, but it had several obvious disadvantages. First, each program would have to be recoded for each machine, adding to the effort involved. Moreover, this introduced programmer variability into the experiment, and previous studies have shown programmer variability to be large (variation of factors of 4:1 or more are commonly accepted). Finally, it is much more expensive to code in assembly language than in Higher Order Languages, and this would limit the size or number of the test programs. *Nevertheless,* the committee felt that there were ways to limit, separate, and measure these programmer effects, while there was no practical way to limit or separate the effects of compiler efficiency. It was therefore decided that the test programs would, of necessity, be coded in assembly language.

Guidelines for test programs specification

The Test Program Subcommittee attempted to establish a strategy for defining and coding the test programs that would minimize the variability due to differences in programmer skill. The strategy devised was as follows:

a. The test programs would be small "kernel" type programs, of not more than 200 machine instructions. (In the end, a few test programs required more than 200 instructions.) It was felt that only small programs could be specified and controlled with sufficient precision to minimize the effects of programmer variability. Moreover, resources were not available to define, code, test, and measure a significant set of larger programs.
b. The programs were defined as structured programs, using a PL/I-like Program Definition Language (PDL) and then "hand translated" into the assembly languages of the respective architectures.
c. Programmers were not permitted to make *algorithmic* improvements or modifications, but rather were required to translate the PDL descriptions into assembly language. Programmers were free to optimize their test programs to the extent possible with highly optimizing compilers. This "hand translation" of strictly defined algorithms was expected to reduce variations due to programmer skill.
d. All test programs except the I/O Interrupt test programs were coded as reentrant, position-independent (or self-relocating) subroutines. This was believed to be consistent with the best contemporary programming practice and provides a good test of an architecture's subroutine and addressing capabilities.

Selection of the twelve test programs

The CFA committee appointed a subcommittee responsible for developing a set of test program specifications consistent with the guidelines just discussed. This subcommittee defined a set of 21 test programs that were intended to be broadly representative of the basic types of operations performed by military computer systems. The CFA committee reviewed these 21 test programs, committee members were asked to rank the relevance of these test programs to the applications of their particular organization, and it was agreed that the top 12 programs would be the basis of the test program study. (The rationale for using 12 test programs is explained in a later section, where the statistical design of the test program assignments is presented.) The full specification of the 12 selected test programs is given in Reference 5 and a brief description of these test programs is given below.

A. *I/O kernel four priority levels,* requires the processor to field interrupts from four devices, each of which has its own priority level. While one device is being processed, interrupts from higher priority devices are allowed.

B. *I/O kernel, FIFO processing,* also fields interrupts from four devices, but without consideration of priority level. Instead, each interrupt causes a request for processing to be queued; requests are processed in FIFO order. While a request is being processed, interrupts from other devices are allowed.

C. *I/O device handler* processes application programs' requests for I/O block transfers on a typical tape drive, and returns the status of the transfer upon completion.

D. *Large FFT* computes the fast Fourier transform of a large vector of 32-bit floating point complex numbers. This benchmark does exercise the machine's floating point instructions, but principally tests its ability to manage a large address space. (Up to one half of a million bytes may be required for the vector.)

E. *Character search,* searches a long character string for the first occurrence of a potentially large argument string. It exercises the ability to move through character strings sequentially.

F. *Bit test, set, or reset* tests the initial value of a bit within a bit string, then optionally sets or resets the bit. It tests one kind of bit manipulation.

G. *Runge-Kutta integration* numerically integrates a simple differential equation using third-order Runge-Kutta integration. It is primarily a test of floating-point arithmetic and iteration mechanisms.

H. *Linked list insertion* inserts a new entry in a doubly-linked list. It tests pointer manipulation.

I. *Quicksort* sorts a potentially large vector of fixed-length strings using the Quicksort algorithm. Like FFT, it tests the ability to manipulate a large address space, but it also tests the ability of the machine to support recursive routines.

J. *ASCII to floating point* converts an ASCII string to a floating point number. It exercises character-to-numeric conversion.

K. *Boolean matrix transpose* transposes a square, tightly-packed bit matrix. It tests the ability to sequence through bit vectors by arbitrary increments.

L. *Virtual memory space exchange* changes the virtual memory mapping context of the processor.

The specifications, written in the Program Definition Language, were intended to completely specify the algorithm to be used, but allow a programmer the freedom to implement the details of the program in whatever way best suited the architecture involved. For example, in the ASCII-to-floating-point benchmark, program J, the PDL specification included the statement:

NUMBER←integer equivalent of characters POSITION
to J-1 of A1 where character J of A1 is "."

This description instructs the programmer to convert the character substring POSITION, POSITION+1, . . . , J−1, to an integer and store the result in the integer NUMBER. It left up to the programmer whether he would sequence through the string character-by-character, accumulating an integer number until he found a decimal point, or perhaps (on the S/370) use the Translate-and-Test (TRT) instruction to find the decimal point, and then use PACK and Convert-to-Binary (CVB) to do the conversion. It did forbid him to accumulate the result as a floating point number directly, forcing him to convert to an integer and then to floating point.

Procedures for writing, debugging, and measuring the test programs

The test programs were written by seventeen programmers at various Army and Navy laboratories and at Carnegie-Mellon University. A set of reasonably comprehensive instructions and conventions were needed to insure that the various programmers produced results that could be compared in a meaningful way. A later section of this article discusses the assignments made to the programmers, and shows how these assignments were made to minimize the distortion of the final conclusions due to variations between programmers. In addition, we also agreed that it was not sufficient to just write the test programs in assembly language. We instructed each programmer that all of the test programs that he wrote had to be assembled and run on the appropriate computer.* Test data was distributed to the programmers, and a test program was defined to be debugged for the purposes of the CFA committee's work if it performed correctly on the test data.

S, M AND R: MEASURES OF AN ARCHITECTURE'S PERFORMANCE

Very little has been done in the past to quantify the relative (or absolute) performance of computer architectures, independent of specific implementations. Hence, like it or not, we had little choice but to define measures of architecture performance for ourselves.

Fundamentally, performance of computers is measured in units of space and time. The measures that were used by the CFA Committee to measure a computer architecture's performance on the test programs were:

Measure of Space

S: Number of bytes used to represent a test program.

Measures of Execution Time:

M: Number of bytes transferred between primary mem-

* The exceptions were test programs A, B, C, and L since they all require the use of privileged instructions and it was impractical to require programmers to get stand-alone use of all the candidate machines. In these four cases, an "expert" on a test program was designated and he was responsible for reading in detail all implementations of the test program and returning the test programs to the programmer for correction if he detected any errors.

ory and the processor during the execution of the test program.

R: Number of bytes transferred among internal registers of the processor during execution of the test program.

All of the measures described in this section are measured in units of 8-bit bytes. A more fundamental unit of measure might be bits, but we faced a number of annoying problems with respect to carry propagation and field alignment that make the measurement of S, M, and R in bits unduly complex. Fortunately, all the computer architectures under consideration by this committee are based on 8-bit bytes (rather than 6, 7, or 9-bit bytes) and hence the byte unit of measurement can be conveniently applied to all these machines.

Test program size

An important indication of how well an architecture is suited for an application (test program) is the amount of memory needed to represent it. We define $S_{i,j,k}$ to be the number of 8-bit bytes of memory used by programmer i to represent test program j in the machine language of architecture k. The S measure includes all instructions, indirect addresses, and temporary work areas required by the program.

The only memory requirement not included in S is the memory needed to hold the actual data structures, or parameters, specified for use by the test programs. For example, in the Fourier transform test program S did not include the space for the actual vector of complex floating-point numbers being transformed but it did include pointers used as indices into the vector, loop counters, booleans required by the program, and save-areas to hold the original contents of registers used in the computation.

Processor execution rate measures

In selecting among computer architectures, as opposed to alternative computer systems, we are faced with a fundamental dilemma: one of the most basic measures of a computer is the speed with which it can solve problems, yet a computer architecture is an abstract description of a computer that does not define the time required to perform any operation. (In fact, it is exactly this time-independence that makes the concept of a computer architecture so attractive!) Given this dilemma, one reaction might be to ignore performance when selecting among alternative computer architectures and leave it to the engineers implementing the various physical realizations to worry about execution speed. However, to adopt this attitude would invite disaster. In other words, although we were evaluating architectures, not implementations, it was essential that the architecture selected yield cost/effective implementations, i.e., the architecture must be "implementable."

The M and R measures defined below were developed to measure those aspects of a computer architecture that will most directly affect the performance of its implementations.

Processor memory transfers

If there is any single, scalar quantity that comes close to measuring the "power" of a computer system, it is the bandwidth between primary memory and the central processor(s).[6-8]

This measure is not concerned with the internal workings of either the primary memory or the central processor; it is determined by the width of the bus between primary memory and the processor and the number of transfers per second the bus is capable of sustaining. Since processor/memory bandwidth is a good indicator of a computer's execution speed, an important measure of an architecture's effect on the execution speed of a program is the amount of information it must transfer between primary memory and the processor during the execution of the program. If one architecture must read or write 2×10^6 bytes in primary memory in order to execute a test program and the second architecture must read or write 10^6 bytes in order to execute the same test program, then, given similar implementation constraints, we would expect the second architecture to be substantially faster than the first.

The particular measure of primary-memory/central-processor transfers used by the CFA Committee is called the M measure. $M_{i,j,k}$ is the number of 8-bit bytes that must be read or written from primary memory by the processor of computer architecture k during the execution of test program j as written by programmer i.

Clearly, there are implementation techniques used in the design of processors and memories to improve performance by attempting to reduce processor/memory traffic, i.e., cache memories, instruction lookahead (or behind) buffers, and other buffering schemes. However, with the intention of keeping our measure of processor/memory traffic as simple, clean, and implementation-independent as possible, none of these buffering techniques were considered. At the completion of one instruction, and before the initiation of the next instruction, the only information contained in the processor is the contents of the registers in the processor state.

Table I shows an example of a small IBM S/370 instruction sequence which should help to illustrate the calculation of M. The instructions are the basic loop of a routine for calculating the inner product of two single precision floating point vectors of length 10.

Registers transfers within the processor

The processor/memory traffic measure just described is our principal measure of a computer architecture's execution rate performance. However, it should not be too surprising that this M measure does not capture all we might want to know about the performance potential of an architecture. In this section a second measure of architec-

TABLE I—M Measure for IBM 370 Inner Product Example

			M	Comments
(1)	LA	2,10(0,0)	4	Set R2 to 10, the length of the vectors.
(2)	LA	3,XVEC	4	Load R3 with starting address of X vector.
(3)	LA	4,YVEC	4	Load R2 with starting address of Y vector.
(4)	SDR	2,2	2	Clear floating point reg. 2.
				Use it to accumulate inner product.
(5)	SR	7,7	2	Clear R7
				Use it as index into floating point vectors.
(6) LOOP	LE	4,0(7,3)	8	Load X(i) into floating point register 4.
(7)	ME	4,0(7,4)	8	Multiply X(i) by Y(i).
(8)	ADR	2,4	2	Sum := Sum+X(i)*Y(i).
(9)	LA	7,4(0,7)	4	Increment index by 4 bytes.
(10)	BCT	2,LOOP	4	Decrement loop count and branch back if not done
			26	(Loop Total)
			260	(Loop (6-10)* 10)
(11)	STO	2,SUM	12	Store double precision result in SUM.
			288	Grand Total

ture performance is defined: R—register-to-register traffic within the processor. Whereas the M measure looks at the data traffic between primary memory and the central processor, R is a measure of the data traffic internal to the central processor. The fundamental goal of the M and R measures was to enable the CFA committee to construct a processor execution rate measure from M and R (ultimately an additive measure: aM+bR, where the coefficients a and b can be varied to model projections of relative primary memory and processor speeds). An unfortunate but unavoidable property of the R measure is that it is very sensitive to assumptions about the register and bus structure internal to the processor; in other words, the "implementation" of the processor.

The definition of R is based on the idealized internal structure for a processor shown in Figure 1. By using the register structure in Figure 1 we do not imply that this is the way processors ought to be built. On the contrary, the structure in Figure 1 has a much more regular data path structure than would be practical in contemporary processors. There exist both data paths of marginal utility and nonexistent data paths that, if present, could significantly speed up the processor. This structure was selected because the very regular data path, ALU, and register array structure helped simplify our analysis.

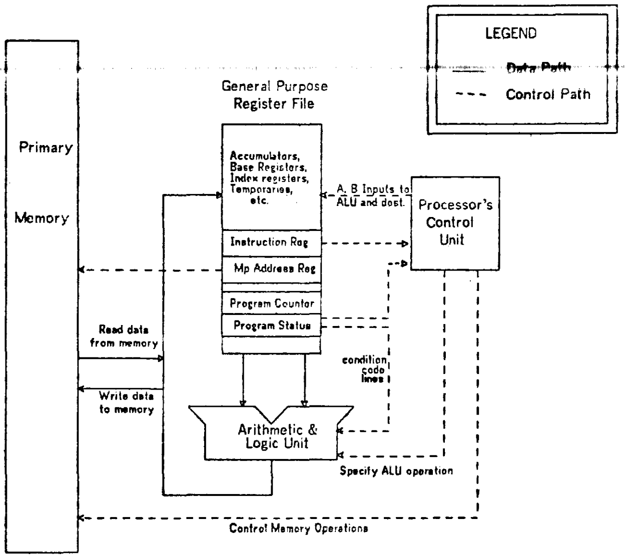


Figure 1—Canonical processor organization for R measure computations

$R_{i,j,k}$ is defined as the number of 8-bit bytes that are read to and written from the internal processor registers during execution of test program j on architecture k as written by programmer i.

ALU Operations—The ALU in Figure 1 is allowed to perform any common integer, floating point, or decimal arithmetic operation; increment or decrement; and perform arbitrary shift or rotate operations.

Only Data Traffic Measured—All data traffic is measured in R and no control traffic measured. Figure 1 is intended to specify what will be defined to be control traffic and what will be data traffic for the purposes of the R measure. The R measure does not count the following "control" traffic:

(1) The setting of the condition codes by the ALU (or control unit) and the use of the condition codes by the ALU. The only time that movement of data into or out of the Program State Word will be counted in the R measure is when a Load PSW instruction is performed or a trap or interrupt sequence moves a new PSW into or out of the PSW register.

(2) Bits transmitted by the control unit to activate or otherwise control the register file, ALU, or memory unit, are not counted in the R measure.

(3) Reading of the Instruction Register by the control unit as it decodes the instruction to determine the instruction execution sequence is not counted in the R measure. In other words, the Instruction Register

(with the exception of displacement fields) will be for most practical purposes a write-only register as far as the R measure is concerned.

(4) Loading the Memory Address Register is counted in the R measure, but use of the contents of the Memory Address Register to specify the address of data to be accessed in primary memory is not counted.

Virtual Address Translation—The virtual to real address translation process is not counted in the R measure. In other words, the final memory address in the MAR is a virtual address and the work involved in translating this virtual address to a real address is not included in the R measure.

The definition of the R measure was the center of considerable discussion within the CFA committee. The full set of rules that are necessary to completely define the R measure is too voluminous to present here; readers interested in the details of the R measure are referred to Volume III of the CFA Selection Committee's final report.[5] Figure 2 illustrates the calculation of the R measure for an IBM/370 add instruction.

STATISTICAL DESIGN OF TEST PROGRAM ASSIGNMENTS

The test program phase of the CFA evaluation process involved comparison of twelve test programs on three machines. Approximately sixteen programmers were available for the study and a complete factorial design would have required each programmer to write all of the test programs on each of the machines (for a total of 576 programs). This was clearly not feasible with the given time and resource constraints, and, consequently, a fractional design (or several fractional designs) had to be selected. Fractional factorial designs are discussed in Reference 9. The fractional designs to be described below incorporate balance in the way test program, machine, and programmer combinations are assigned.

It was necessary to consider designs which required each programmer to write test programs for all three machines. Otherwise, comparisons among the machines could not be separated from comparisons among the programmers. A desirable design would have instructed each programmer to write a total of six or nine different test programs, one third of them on each of the three machines. For most of the programmers in the study time limitations precluded this type of design, and some compromise was required. The compromise design selected also had to allow for precise comparisons among the three competing architectures. A type of design that meets both of these objectives is the nested factorial.[10]

The test program part of the study actually involved the use of three separate experimental designs, henceforth referred to as Phase I, Phase II, and Phase III. Nested factorial designs were used for Phase I and Phase III. Phase II was a one-third fraction of a 3^4 factorial design. Phase I was used to study test programs A through H, those deemed to be of primary interest. Phase III was used to study test programs I through L. Phase II included test programs A-B, E-H, and J-L. Plans of the three designs are depicted in Figure 3.

The Phase I design is a pair of nested factorials, each involving four programmers. Each programmer was asked to write two test programs for all three machines. Each of the eight test programs in Phase I appears once on each machine in each of the nested factorials. When this design was originally formulated, the plan included requiring programmers to write their six test programs in a preassigned randomly selected order, so as to eliminate possible biases due to learning during the course of completing the assignments. This procedure was discarded, however, when the programmers objected because of the varying availability of the three machines for debugging. Programmers were instructed to complete the assigned jobs in conformity with their typical practices and working habits with regard to order, consultation with other individuals, and other such considerations. Programmers in the study were not permitted to consult with each other, however, on any substantive matters concerning their designated assignments. All programmers were instructed to keep diaries of their work on the experiment.

As noted above, the Phase I design was formulated with the goal of obtaining maximum possible information about differences between the competing architectures. With the given Phase I design, comparisons among the three architectures are not confounded by effects of either test programs or programmers. The Phase I design called for 48 observations and was viewed as the most important of the three designs formulated.

The design termed Phase III was formulated according to the same plan as was Phase I, except that four test programs and four programmers were utilized.* The Phase III design contains half as many observations as the Phase I design and thus gives statistical results of less precision. The test programs in the Phase III design are of lesser interest than those in Phase I. The four programmers in Phase III are distinct from the eight in Phase I.

Together the Phase I and Phase III designs provide a view of all three machines and the operation of all twelve test programs selected for consideration. A third experiment, labelled Phase II, was also planned. This was viewed as an auxiliary effort and was to be completed only if it was clear that the programmers assigned to it would not be needed to aid in the completion of Phase I and Phase III. the Phase II design called for three programmers to write nine different test programs, three on each of the three machines. The programmers assigned to Phase II were able to devote enough time to the test program study to permit use of a design which required them to write nine different programs. Some comparisons among programs not possible in Phase I and Phase III could be made, and the statistical results of Phase II could be compared to those of the other two experiments. The design used was the 3.4.3 plan in Reference 11. This was made possible by dividing the factor

RX, RS, & SI INSTRUCTION INTERPRETATION

	R	Comment
IR<0:15> ← Mh[MAR]	2	Get halfword in instruction register
MAR ← MAR + 2	3	Incrementation counts only 1 byte
IR<15:31> ← Mh[MAR]	2	Get rest of instruction in IR
PC ← PC + 4	3	Increasing Program Counter
address interpretation	-	
instruction execution	-	
MAR ← PC	6	Set up MAR for next instruction

TOTAL	16	

RX ADDRESS CALCULATION

	R	Comment
1. B2 = 0, X2 = 0		
MAR ← IR<20:31>	5	Read 12 bits from the IR
2. B2 = 0, X2 > 0		
MAR ← IR<20:31> + R[x2]<8:31>	8	
		Add 12 bits from IR to 24 bits from index
3. B2 > 0, X2 = 0		
MAR ← IR<20:31> + R[B2]<8:31>	8	
4. B2 > 0, X2 > 0		
MAR ← IR<20:31> + R[B2]<8:31>	8	
MAR ← R[x2] + MAR	9	Full 24 bit (3 byte) addition

TOTAL	17	

EXAMPLE INSTRUCTION: A R4,DISP(R2,R7)

RX Add Instruction	R
RX instruction interpretation	16
address inti rpretation	17
MBR ← Mw[MAR]	4
R[R1] ← R[R1] + MBR	12
	--
TOTAL	49

Figure 2—IBM S/370 R measure example

representing test programs, which appears at nine levels, into two pseudofactors,[10] each at three levels. One of the Phase II programmers also participated in the Phase I design. The only duplicate assignment, however, was test program G on the IBM S/370.

ANALYSIS OF TEST PROGRAM RESULTS

This section describes the experimental results and statistical analysis of the test program data. We shall first discuss the Phase I experiment, then the Phase III experiment, and then the analysis combining data from Phase I and III. Finally, the Phase II experiment will be described.

Phase I models

A possible model for the nested factorial designs in Phase I is

$$y_{ijk}=C+P_i+T_{ij}+M_k+PM_{ik}+TM_{ijk}+e_{ijk}, \qquad (1)$$

$$i=1, 2, 3, 4, j=1, 2, k=1, 2, 3.$$

In this equation y_{ijk} is some response (i.e., an S, M or R measure) generated by the ith programmer writing the jth test program on the kth machine. Also,

C = constant, termed the grand mean
P_i = effect due to the ith programmer
T_{ij} = effect of the jth test program assigned to the ith programmer
M_k = effect of the kth machine
PM_{ik} = interaction between the ith programmer and the kth machine
TM_{ijk} = interaction between the jth test program written by the ith programmer and the kth machine
e_{ijk} = a random error term, assumed to be normally distributed with mean 0 and variance not dependent on the values of i, j, and k.

The Phase I experiment may also be modelled in a manner somewhat different from that just described. In Phase I there are two factors at eight levels each, programmers and test programs, and one factor at three levels, machines. The two eight-level factors may each be replaced by three pseudofactors at two levels each. Then we are concerned with a complete factorial experiment involving $3*2^6=192$ total observations. The actual Phase I experiment is a $^1/_4$ fraction of this. A model may be fit using dummy variables to account for various effects and interactions.

Transformation of the data

Examination of the S, M, and R data values collected clearly shows there is wide variation in the data from one test program to another, e.g., especially for the M and R measures. Various statistical considerations suggest that some transformation of the raw data prior to analysis is desirable. A technical discussion of transformation of statistics is given in Reference 12 which illustrates use of the methodology in various contexts.

In the CFA study the purpose of a transformation of the data is to stabilize variance, so that an additive model such as (eq. 1) will hold for each of the designs. Specifically, the model (eq. 1) assumes that the variance of the error term e_{ijk} is independent of i, j, and k. Under this assumption inferences which follow from analysis of variance (ANOVA) calculations, as described below, are valid.

A variance stabilizing transformation is frequently suggested by consideration of the experimental situation and prior understanding of the variation to be expected in the data. For example, consider the M and R measures. Suppose some programmers each write two test programs and the average run time of the second one is k times the average run time of the first. Then if the standard deviation of the M or R readings is V for the first test program, it can be expected to be proportional to kV for the second test program. In other words, the variability (standard deviation) in run times is directly proportional to the average run time. The accuracy of this conjecture may be tested by examination of the data, but clearly there is strong intuitive support for it. Consider the Runge-Kutta test program. Its M and R measures are dominated by the computation of the inner loop performing the step-wise solution of the differential equation. Variations in M and R measures will be a result of alternative encodings of this inner loop. Average M and R measures will be doubled if the number of iterations requested is doubled. Moreover, doubling the number of iterations will also cause the differences between the different Runge-Kutta programs to double. When the standard deviation of the test data is directly proportional to the mean, a logarithmic transformation will stabilize the variance, that is, remove the dependence of the variance on the size of the test program.[12]

The model of (eq. 1) may be termed an *additive* model. When a logarithmic transformation is used for the data, y_{ijk} in (eq. 1) becomes the logarithm of the response, such as the M or the R reading. In this case a *multiplicative* model in fact underlies (eq. 1) and we write

$$z_{ijk}=\chi\pi_i\tau_{ij}\mu_k\pi\mu_{ik}\tau\mu_{ijk}\epsilon_{ijk}, \qquad (2)$$

$$i=1, 2, 3, 4, j=1, 2, k=1, 2, 3.$$

The connection between (eq. 1 and eq. 2) is

$$\ln z_{ijk}=y_{ijk}$$
$$\ln \chi=C,$$
$$\ln \pi_i=P_i,$$
$$\ln \tau_{ij}=T_{ij},$$
$$\ln \mu_k=M_k,$$
$$\ln \pi\mu_{ik}=PM_{ik},$$
$$\ln \tau\mu_{ijk}=TM_{ijk},$$
$$\ln \epsilon_{ijk}=e_{ijk}.$$

Thus, use of the logarithmic transformation on both sides of (eq. 2) yields (eq. 1), and the multiplicative model (eq. 2) may be viewed as the meaningful basic underlying model.

Similarly, consideration of the underlying properties of the S measure suggest a square root transformation is appropriate to stabilize its variance. This transformation arises because the variance, rather than the standard deviation, of the S measure can be expected to be proportional to kV.[5] Use of the square root transformation would imply use of the model in (eq.1) with y_{ijk} denoting the square root of the measured S value.

It should be noted that the square root and logarithmic

Phase	Programmer	A	B	C	D	E	F	G	H	I	J	K	L
I	14	all							all				
	1			all		all							
	2		all					all					
	9				all		all						
	11				all	all							
	12	all					all						
	13		all						all				
	17			all				all					
II	3	370	11			832	11	11	832		832	370	370
	4	11	832			370	832	832	370		370	11	11
	17	832	370			11	370	370	11		11	832	832
III	5									all	all		
	8											all	all
	6									all		all	
	7										all		all

Figure 3—Layouts of Phase I, II, and III designs "all" designates all three machines

transformations are only two of a large number of possible transformations. A particular family of transformations takes a response z and transforms it according to z^a for an $a>0$. With an appropriate interpretation, the logarithmic transformation corresponds to the limiting value $a \to 0$. This family of power transformations is discussed in detail in Reference 13.

Statistical analysis of phase I data

ANOVA calculations were performed on both halves of the Phase I experiment for $\sqrt{S}$, ln M, and ln R values. In each analysis the sample variance of the 24 values was decomposed into sums of squares attributable to variations among programmers test programs, machines, programmer-machine interactions, and test program-machine interactions. The proportions of the total variance due to the various sums of squares are given in Table II. The ANOVA calculations indicate that test program and programmer variations account for most of the variation in the data in the case of the M and R measures, and that machine

TABLE II—Estimates of Machine Comparisons and 95 percent Confidence Intervals, Phase I

Measure / Comparison of Machines	$\sqrt{S}$	ln M	ln R
M_3-M_1	−.586 (−3.696, 2.524)	.018 (−.430, .466)	.012 (−.449, .474)
M_3-M_2	−3.535 (−6.645, −.425)	−.655 (−1.103, −.207)	−.717 (−1.178, −.255)
M_2-M_1	2.949 (−.161, 6.059)	.673 (.225, 1.121)	.729 (.267, 1.191)
$\frac{1}{2}(M_1+M_3)-M_2$	−3.242 (−5.936, −.548)	−.664 (−1.052, −.276)	−.723 (−1.122, −.323)

model (eq. 1):
M_1: effect of PDP-11
M_2: effect of IBM S/370
M_3: effect of Interdata 8/32

differences are relatively small. Machine differences are more noticeable for the S measure.

Using dummy variables, we also fit models using the formulation discussed earlier. In each model 24 parameters were fit, leaving 24 degrees of freedom to measure experimental error. Estimates of the variance of the error term in the model (eq. 1) are 18.175, 0.377, and 0.400 for $\sqrt{S}$, ln M, and ln R, respectively. The actual data values for the S, M, and R measures are given in the Appendix, and these estimates of variance reflect the magnitude of the experimental error component in the model (eq. 1). Table II shows estimates of various machine comparisons for the Phase I data. A 95 percent confidence interval is quoted below each estimate. The 95 percent confidence intervals which do not cover the value 0 correspond to comparisons statistically significant at level $0.05(=1-.95)$. Thus at level .05 the Interdata 8/32 is superior to the IBM S/370 on all measures. The PDP-11 is adjudged superior to the IBM S/370 at level .05 on two of the measures and barely misses being superior when $\sqrt{S}$ is considered. Moreover, the IBM S/370 is inferior to the average performance of the other two machines on all measures. It is worth noting that these comparisons among the competing architectures are based upon consideration of test programs A through H only. It is reasonable, however, to view the eight programmers in Phase I as representative of a larger population of programmers.

Table III displays estimates of the effects M_k and μ_k for the various measures. The μ_k estimates are obtained by exponentiating the estimates of M_k and are appropriate for the logarithmic models only. Estimates have been included for architecture comparisons obtained from the model (eq. 1) with the response ln S. These are also given in Tables V and VII. Use of the ln S model leads to estimates which are qualitatively similar to those obtained from the $\sqrt{S}$ model, and it permits more convenient comparisons of the three architectures. Since the effects noted in Table III are differential values, a value of 0 is neutral for M_k and a value of 1 is neutral for μ_k. The figures in Table III are consistent for the different measures and transformations. The IBM S/370 is noticeably worse than the other two architectures. For all but the ln R response, the Interdata 8/32 appears to be modestly better than the PDP-11.

One may interpret the last three lines of Table III in the following way. The ln M measure results indicate the IBM S/370 requires 155.7 percent as many processor/memory transfers to "execute" programs A through H as the average of the three machines, while the PDP-11 and Interdata 8/32 require 79.5 percent and 80.9 percent, respectively.

TABLE III—Estimates of Machine Effects in Models (eq.1) and (eq.2), Phase I

Measure	$\sqrt{S}$	ln S	ln M	ln R
Machine Effects				
M_1	−.788	−.148	−.230	−.247
M_2	2.161	.354	.443	.482
M_3	−1.374	−.205	−.212	−.235
μ_1		.862	.795	.781
μ_2		1.425	1.557	1.619
μ_3		.815	.809	.791

M_1, μ_1: effects for PDP-11
M_2, μ_2: effects for IBM S/370
M_3, μ_3: effects for Interdata 8/32

TABLE IV—Estimates of Machine Comparisons and 95 percent Confidence Intervals, Phase III

Comparison of Machines \ Measure	$\sqrt{S}$	ln M	ln R
M_3-M_1	−3.806 (−8.780, 1.168)	−.295 (−1.000, .410)	−.348 (−.988, .291)
M_3-M_2	−1.585 (−6.559, 3.389)	.099 (−.606, .804)	−.027 (−.666, .613)
M_2-M_1	−2.221 (−7.195, 2.753)	−.394 (−1.099, .311)	−.321 (−.960, .318)
$\frac{1}{2}(M_1+M_3)-M_2$	.318 (−3.990, 4.626)	.247 (−.364, .858)	.147 (−.407, .701)

M_1: effect of PDP-11
M_2: effect of IBM S/370
M_3: effect of Interdata 8/32

Phase III models and results

The models for Phase III experiments are the same as in (eq. 1) and (eq. 2), except that the subscript i assumes the values 1 and 2 only. Estimates of the variance of the error term in the Phase III version of model (eq. 1) are based on eight degrees of freedom and are 18.606, 0.374, and 0.308 for $\sqrt{S}$, ln M, and ln R, respectively.

Table IV is the analog of Table II, and Table V the analog of Table III. None of the confidence intervals shown in Table IV fails to cover the value 0. However, it is apparent that the PDP-11 performed noticeably worse than the other two machines in Phase III. Also, there is very little difference between the IBM S/370 and the Interdata 8/32 in Phase III.

The relatively poor performance of the PDP-11 in Phase III appears to be due to its inability to handle test program I, quicksort. Certainly part of the explanation for the poor performance of the IBM S/370 in Phase I can be attributed to test program A, I/O kernel with four priority levels. In

TABLE V—Estimates of Machine Effects in Models (eq.1) and (eq.2), Phase III

Measure	$\sqrt{S}$	ln S	ln M	ln R
Machine Effects				
M_1	2.009	.133	.229	.223
M_2	−.212	.042	−.165	−.098
M_3	−1.797	−.174	−.066	−.125
μ_1		1.142	1.257	1.250
μ_2		1.043	.848	.907
μ_3		.840	.936	.882

M_1, μ_1: effects for PDP-11
M_2, μ_2: effects for IBM S/370
M_3, μ_3: effects for Interdata 8/32

the next section results from Phase I and Phase III are combined to produce overall estimates of machine effects and overall comparisons of the machines.

Combination of Phase I and Phase III results

Let Θ_I denote an estimate of a machine effect or comparison, such as M_1 or M_3-M_1, in Phase I. Let Θ_{III} denote the estimate of the same effect or comparison in Phase III. In the previous two sections such estimates were given, as well as some confidence intervals. The purpose of this section is to present estimates of the form

$$\alpha\theta_I+(1-\alpha)\theta_{III}, \qquad (3)$$

where α is chosen to minimize the variance of the resulting linear combination and $0<\alpha<1$. Table VI shows estimates of machine comparisons and 95 percent confidence intervals. The value of α for each column in the table is given along the top border. In all columns more weight is given to the Phase I data. Table VII gives estimates of machine effects with Phase I and Phase III data combined.

All of the confidence intervals for M_3-M_2 in Table VI fail to cover the value zero. Thus, the evidence suggests that the Interdata 8/32 performs better than the IBM S/370 on all three measures, S, M, and R. Also, the IBM S/370 tends to be worse than the average of the other two machines.

The estimates of μ_k in Table VII provide a summary of the Phase I and Phase III data. The IBM S/370 requires 120.8 percent as much storage as the average of all three machines for the twelve test programs studied. According to the ln M measure estimate, the IBM S/370 required 126.6 percent as many processor/memory transfers to "execute" the test programs as the average of the three machines. The other figures in the lower part of Table VII are interpreted similarly.

Phase II models and results

Analysis of variance calculations were performed on data arising from the Phase II design. Some of the results for responses $\sqrt{S}$, ln R, and ln M are summarized in Table VIII. This table indicates the proportions of the total variance attributable to various sums of squares. The variance was split into sums of squares each with two degrees of freedom. Since two of the factors in the design were in fact pseudofactors at three levels each to account for the nine test programs, several sets of sums of squares were combined. There is some aliasing in the design involving the second-order interactions.

TABLE VI—Estimates of Machine Comparisons and 95 percent Confidence Intervals, Phase I and Phase III Data Combined

Comparison of Machines \ Measure	$\sqrt{S}$ α=.67	ln M α=.66	ln R α=.61
M_3-M_1	−1.649	−.088	−.128
	(−4.119, .821)	(−.442, .266)	(−.517, .261)
M_3-M_2	−2.892	−.399	−.448
	(−5.362, −.422)	(−.753, −.045)	(−.837, −.059)
M_2-M_1	1.243	.310	.320
	(−1.227, 3.713)	(−.044, .664)	(−.069, .708)
$\frac{1}{2}(M_1+M_3)-M_2$	−2.067	−.354	−.384
	(−4.207, .073)	(−.661, −.047)	(−.721, −.047)

M_1: effect of PDP-11
M_2: effect of IBM S/370
M_3: effect of Interdata 8/32

TABLE VII—Estimates of Machine Effects in Models (eq.1) and (eq.2), Phase I and Phase III Data Combined

Measure	$\sqrt{S}$	ln S	ln M	ln R
Machine Effects	α=.67	α=.47	α=.66	α=.61
M_1	.135	.001	−.075	−.064
M_2	1.378	.189	.236	.256
M_3	−1.514	−.189	−.163	−.192
μ_1		1.001	.928	.938
μ_2		1.208	1.266	1.292
μ_3		.828	.850	.825

M_1, μ_1: effects for PDP-11
M_2, μ_2: effects for IBM S/370
M_3, μ_3: effects for Interdata 8/32

Estimates of differential effects in a model comparable to (eq. 1) for the three machines can also be given. For the $\sqrt{S}$ measure they are −.952 for the PDP-11, 1.605 for the IBM S/370, and −.653 for the Interdata 8/32. For the ln M measure the values are −0.691, 0.508, and 0.183 for the machines quoted in the same order, and the figures are −.662, .538, and .123 for the ln R measure. Thus, the experimental results for this phase tend to rank the machines with the PDP-11 first by a substantial margin, and the Interdata 8/32 ranks second. However, it should be noted that test program A was included in the Phase II design, and test programs D and I were not.

SUMMARY

This article has described how the test program phase of the CFA study was developed, what methodologies were used, and what were the results of the study. We began

TABLE VIII—Phase II ANOVA Calculations Proportion of Variance Attributable to Each Sum of Squares

Measure		$\sqrt{S}$	ln M	ln R
Sum of Squares	Degrees of freedom			
Programmers	2	.027	.018	.026
Test Programs	8	.623	.653	.660
Machines	2	.132	.076	.068
Programmers × Machines	2	.039	.053	.047
Test Programs × Machines	8	.132	.124	.121
Test Programs × Programmers	4	.047	.076	.078

with a discussion of the twelve test programs used in this study and how the CFA committee selected these twelve from a larger set of test programs as most representative of the expected applications of military computers. A Program Definition Language (PDL) was used to clearly specify these test programs so that it was clear to the programmers exactly what algorithm was to be implemented yet also indicate to what extent we expected the programmer to optimize the coding of the test programs to take advantage of the features of the architecture under test.

The third section of this article defined the three measures of performance used to evaluate the candidate computer architectures on each test program:

S: The number of bytes used to represent a test program
M: The number of bytes transferred between primary memory and the processor during execution of the test program
R: The number of bytes transferred among internal registers of the processor during execution of the test program

The test programs were assigned to programmers based on a statistical design involving three phases, denoted as I, II, and III. In Phase I eight programmers were assigned two test programs to implement on each of the three machines. Phase III was a smaller version of Phase I, involving only four programmers. Phase II was a somewhat more complex design that involved each of three programmers writing nine different test programs, three on each machine. Phase II was intended to give some information on the interaction between particular test programs and machines that was not available with much precision from Phases I and III.

The principal results of the test program study that were passed along to the life-cycle cost models[13] was the composite performance of the candidate architectures for Phases I and III on the set of 12 test programs. An analysis of Variance (ANOVA) procedure was used to determine the overall relative performance of the three candidate machines. Unity indicates average performance and the lower the score on any of the measures, the better the machine handled the set of test programs.

In other words, the test program results indicate that the IBM S/370 needs 46 percent more memory than the Interdata 8/32 to represent the set of test programs (or 21 percent more than the average of the three architectures) and the PDP-11 is essentially average in its use of memory.

Considering the test program results in a little more detail, in Phase I the data revealed the IBM S/370 to be significantly worse than the other two machines on S, M, and R measures at a significance level of 0.05 (i.e., the 95 percent confidence intervals all failed to include the point where the IBM S/370 equals the performance of the other machines). Moreover, the overall performance of the PDP-11 was virtually identical to that of the Interdata 8/32. Some part of the poor performance of the IBM S/370 can be traced to test program A (the priority I/O kernel). In Phase III alone, none of the comparisons among the three machines was significant at the 0.05 level because of the small number of data points (24). However, the PDP-11 was noticeably the worst of the three machines on all three measures. The IBM S/370 dominated the Interdata 8/32 with regard to the M measure, the Interdata was better for the S measure, and there was little difference between the two for the R measure. The relatively poor performance of the PDP-11 appeared to be due to the quicksort test program, test program I, which worked with a list much larger than the 64K byte virtual address space of the PDP-11.

TABLE IX—Average Performance of the Architectures on the 12 test Programs

ARCHITECTURE	*S*	*M*	*R*
PDP-11	1.00	0.93	0.94
IBM S/370	1.21	1.27	1.29
Interdata 8/32	0.83	0.85	0.83

Statistical results from Phases I and III were combined. In this analysis the ranking of the three machines from best to worst on the three measures was: Interdata 8/32, PDP-11, and IBM S/370. The average performance of the three architectures in Phases I and III is given in Table IX.

The outcome of Phase II largely corroborates the results of the other two experiments. The ranking of the three machines, from best to worst is: PDP-11, Interdata 8/32, IBM S/370. This ranking prevails for all three measures, S, M, and R. It is important to recall (See Table III) that Phase II included test program A, for which the IBM S/370 performs relatively poorly, and does not include test programs D and I, which are relatively difficult to implement on the PDP-11, because they have large data structures. Because of the magnitude of the experimental error in these test programs and the relatively small number of data points in Phase II (27), we were not able to detect any test program/architecture interactions that were statistically significant.

ACKNOWLEDGMENTS

During the specification of the test programs and development of the S, M, and R measures, we had helpful discussions with many individuals related to the CFA project. Mario Barbacci, Lynn DeNoia, Robert Gordon, David Parnas, John Shore, Daniel Siewiorek, and William Smith. We are especially indebted to a group of graduate students at Carnegie-Mellon University who proved crucial to the successful completion of the full set of test programs. Three of these students, Navindra Jain, George Mathew, and Leland Szewerenko were particularly helpful through their continued effort on behalf of this project.

REFERENCES

1. Fuller, S. F., H. S. Stone, and W. E. Burr: "Initial Selection and Screening of the CFA Candidate Computer Architecture," *AFIPS Conference Proceedings*, Vol. 46, 1977 National Computer Conference.

2. Lucas, H. C., "Performance Evaluation and Monitoring," *ACM Computing Surveys*, 3, 3, 1971, pp. 79-91.
3. Bernwell, N. (editor), *Benchmarking: Computer Evaluation and Measurement*, John Wiley & Sons, New York, 1975.
4. Wichmann, B. A., *Algol 60 Compilation and Assessment*, Anderson Press, New York, 1973.
5. Fuller, S. F., W. E. Burr, P. Shaman, and D. Lamb: *Evaluation of Computer Architectures via Test Programs*. Volume III of *Computer Family Architecture Selection Committee Final Report*, Naval Research Laboratory, Washington, D.C., 1 December 1976.
6. Bell, C. G. and A. Newell, *Computer Structures: Readings and Examples*, McGraw-Hill, New York, 1971.
7. *Computer Review*, GML Corporation, Lexington, Mass., 1976.
8. Stone, H. S. (editor), *Introduction to Computer Architecture*, Science Research Associates, Chicago, 1975.
9. Davies, O. L. (editor), *Design and Analysis of Industrial Experiments*, 2nd ed., Oliver and Boyd, Edinburgh, 1971.
10. Anderson, V. L. and R. A. McLean, *Design of Experiments, a Realistic Approach*, Marcel Dekker; Inc., New York, 1974.
11. Connor, W. S. and M. Zelen, *Fractional Factorial Experiment Designs for Factors at Three Levels*, National Bureau of Standards, Applied Mathematics Series 54, 1959.
12. Rao, C. R., *Linear Statistical Inference and its Applications*, 2nd ed., John Wiley & Sons, New York, 1973.
13. Cornyn, J. J., W. R. Smith, W. R. Svirsky, and A. H. Coleman, "Two Life-Cycle Cost Models for Comparing Computer Architectures," *AFIPS Conference Proceedings*, Vol. 46, 1977 National Computer Conference.
14. Box, G. E. P. and B. R. Cox, "An Analysis of Transformations," *The Journal of the Royal Statistical Society*, Series B, Vol. 26, 1964, pp. 211-252.

APPENDIX A-S, M, AND R MEASURES FOR EACH TEST PROGRAM

On the following pages are actual measurements for each of the test programs written for the CFA program. The unit of measurement for all data is (8-bit) bytes. The number in brackets following each measurement is the identifying number of the programmer who wrote and debugged the particular test program. Data followed by an "A" are auxilary data points. Data followed by a "*" were associated with programming assignments not completed in time to be used by the CFA Committee and the pseudo-values shown were used in the ANOVA calculation (when the actual data points became available at a latter date, insertion of the real values for these programs had no significant effect on the results).

INDIVIDUAL S MEASURES

Test Program	Computer Architecture		
			Interdata
	IBM S/370	PDP-11	8/32
A. Priority I/O	216[3]	48[4]	26[12]
Kernel	286[12]	32[12]	28[14]
	742[14]	32[14]	26[17]
B. FIFO I/O	372[2]	133[2]	144[2]
Kernel	465[13]	124[3]	142[4]
	308[17]	246[13]	98[13]
C. I/O Device	192[1]	132[1]	176[1]
Handler	252[17]	216[17]	241[17]

INDIVIDUAL S MEASURES—Continued

Test Program	Computer Architecture		
			Interdata
	IBM S/370	PDP-11	8/32
D. Large FFT	454[11]	766[11]	550[11]
	454[9]*	766[9]*	402[9]
			402[17]A
E. Character	104[1]	88[1]	120[1]
Search	92[4]	136[11]	144[3]
	154[11]	90[17]	168[11]
F. Bit Test, Set,	144[9]	68[3]	82[4]
Reset	122[12]	78[9]	90[9]
	116[17]	86[12]	98[11]A
			98[12]
G. Runge-Kutta	202[2]	184[2]	166[12]
Int.	238[17]	172[3]	158[4]
		248[17]	232[11]A
			190[17]
H. Linked List	144[4]	162[13]	148[3]
Insertion	228[13]	182[14]	198[13]
	176[14]	194[17]	164[14]
I. Quicksort	340[6]	940[6]	426[6]
	407[5]	1534[5]	524[5]
J. ASCII to Float-	256[4]	164[5]	206[3]
Pt.	441[5]	208[7]	238[5]
	241[7]	172[17]	204[7]
K. Boolean Matrix	224[3]	174[4]	156[17]
	267[6]	232[6]	130[6]
	284[8]	284[8]	180[8]
L. Virtual Memory	292[3]	254[4]	328[17]
Exchange	382[7]	250[7]	310[7]
	414[8]	378[8]	334[8]

INDIVIDUAL M MEASURE

Test Program	Computer Architecture		
			Interdata
	IBM S/370	PDP-11	8/32
A. Priority I/O	212[3]	28[4]	28[12]
Kernel	354[12]	24[12]	32[14]
	522[14]	24[14]	
B. FIFO I/O	424[2]	208[2]	192[2]
Kernel	920[13]	188[3]	226[4]
	434[17]	296[13]	114[13]
C. I/O Device	328[1]	309[1]	426[1]
Handler	304[17]	290[17]	279[17]
D. Large FFT	10810[11]	14746[11]	10886[11]
	10810[9]*	14746[9]*	8560[9]*
			8560[17]A
E. Character	854[1]	730[1]	958[1]
Search	940[4]	770[11]	1044[3]
	1724[11]	520[17]	1021[11]
F. Bit Test, Set,	378[9]	162[3]	222[4]
Reset	358[12]	178[9]	176[9]
	238[17]	152[12]	296[11]A
			276[12]

INDIVIDUAL M MEASURE—Continued

Test Program	Computer Architecture		
			Interdata
	IBM S/370	PDP-11	8/32
G. Runge-Kutta Int.	141074[2]	102662[2]	100062[2]
	228056[17]	94960[3]	100042[4]
		176960[17]	117984[11]A
			138414[17]
H. Linked List Insertion	228[4]	204[13]	224[3]
	304[13]	218[14]	260[13]
	264[14]	240[17]	238[14]
I. Quicksort	1024[5]	14960[5]	2968[5]
	1008[6]	2756[6]	1732[6]
J. ASCII to Float-Pt.	241[4]	292[5]	363[3]
	437[5]	275[7]	423[5]
	433[7]	283[17]	334[7]
K. Boolean Matrix	832[3]	582[4]	384[6]
	909[6]	776[6]	566[8]
	896[8]	932[8]	640[17]
L. Virtual Memory Exchange	532[3]	541[4]	721[7]
	532[7]	566[7]	1058[8]
	645[8]	945[8]	780[17]

INDIVIDUAL R MEASURES

Test Program	Computer Architecture		
			Interdata
	IBM S/370	PDP-11	8/32
A. Priority I/O Kernel	947[3]	108[4]	166[12]
	2146[12]	106[12]	166[17]
	3052[14]	106[14]	214[14]
B. FIFO I/O Kernel	2222[2]	1096[2]	698[2]
	4583[13]	810[3]	937[4]
	2226[17]	1419[13]	482[13]

INDIVIDUAL R MEASURES—Continued

Test Program	Computer Architecture		
			Interdata
	IBM S/370	PDP-11	8/32
C. I/O Device Handler	1789[1]	1480[1]	1902[1]
	1729[17]	1416[17]	1391[17]
D. Large FFT	62904[11]	70512[11]	60446[11]
	62904[9]*	70512[9]*	50045[9]*
			50045[17]A
E. Character Search	5603[1]	4348[1]	5885[1]
	5549[4]	4326[11]	3139[3]
	10239[11]	3091[17]	5767[11]
F. Bit Test, Set, Reset	1674[9]	832[3]	891[4]
	1542[12]	917[9]	887[9]
	1212[17]	801[12]	1167[12]
			1281[11]A
G. Runge-Kutta Int.	845966[2]	724372[2]	696085[2]
	1203952[17]	665529[3]	696049[4]
		1012727[17]	777846[11]A
			874923[17]
H. Linked List Insertion	950[4]	1025[13]	834[3]
	1741[13]	1087[14]	1049[13]
	1137[14]	1210[17]	965[14]
I. Quicksort	7618[5]	74278[5]	13315[5]
	7540[6]	15205[6]	9609[6]
J. ASCII to Float-Pt.	1330[4]	1726[5]	2100[3]
	2578[5]	1512[7]	2270[5]
	2226[7]	1716[17]	1897[17]
K. Boolean Matrix	5576[3]	3180[4]	2216[6]
	5661[6]	3905[6]	3154[8]
	5277[8]	4446[8]	3945[17]
L. Virtual Memory Exchange	1931[3]	2616[4]	2539[7]
	1934[7]	2911[7]	4573[8]
	2529[8]	4226[8]	2643[17]

An architectural research facility—ISP descriptions, simulation, data collection*

by MARIO BARBACCI and DANIEL SIEWIOREK
Carnegie Mellon University
Pittsburgh, Pennsylvania

and

ROBERT GORDON and ROSEMARY HOWBRIGG
Naval Underwater Systems Center
New London, Connecticut

and

SUSAN ZUCKERMAN
Naval Research Laboratory
Washington, DC

ABSTRACT

The objectives of this paper are twofold. In the first place we discuss some issues related to the formal description of computer systems and how these issues were handled in a specific project, the selection of a standard computer architecture for the Army/Navy Computer Family Architecture (CFA) project. The second purpose is to present a methodology for automatically gathering architectural data which can be used for evaluation and comparison purposes. We will not discuss the rationale behind the selection of specific test programs and the statisical experiment set up to ascertain the influence of the programmers, the test programs, and the machine architecture on the results. These issues and the actual results of the experiment belong in a companion paper.[9]

Formal descriptions of three candidate architectures (IBM S/370, Interdata 8/32 and DEC PDP-11) were written in ISP, a computer description language. The ISP descriptions of the three architectures were used to simulate the execution of assembly language test programs. The measurements collected during the program simulations were stored into a data base for post-processing. Automating the data collection process not only eliminated tedious and potentially error prone hand calculations, but also provided the means to gather information about dynamic program behavior, information that would be almost impossible to calculate manually.

INTRODUCTION

There have been many attempts to specify computer architectures in some formal notation. The CFA project included, to our knowledge, the first attempt to describe the complete instruction set of several large, commercially available architectures. The candidate architectures were the IBM S/370, DEC PDP-11, and the Interdata 8/32. The experiment described in this paper involved the preparation of formal computer descriptions, the execution of machine language programs under an instrumented simulator, and the collection of data used to evaluate the architectures. Three aspects of the experiment are important to observe: (1) we did not implement specific simulators, tailored for each architecture; the system used in this project is a general purpose computer simulator driven by a formal machine description; (2) we executed a large number of test programs,* each ranging from less than a dozen instructions to several hundred instructions; (3) we used real programs that had been executed on actual physical machines and then used to initialize the simulators.

The Naval Research Laboratory selected ISP[6] as the notation to formally describe the candidate machines. This decision was based on the availability of expertise and software support at CMU and in the fact that ISP was better suited than other candidate notations for describing a computer architecture, independently of timing and other

* The work reported in this paper was sponsored in part by the Naval Research Laboratory, the National Science Foundation (under grant GJ32758X), and the Defense Advanced Research Projects Agency, ARPA (under contract F44620-73-C-0074).

* A total of 114 simulation runs were executed. They correspond to a total of 70 different programs (some of which called for several test cases, in other instances a test case had to be divided into separated sub-cases). The 70 programs were divided as follows: 26 for the PDP-11, 22 for each of the IBM S/370 and Interdata 8/32.

implementation issues.* This however, does not imply that ISP is free of blemishes. Some of its virtues and defects are discussed in Reference 3. In this paper we will point out some characteristics of the notation that prevent a complete separation between architectural and implementation details.

Volume IV of the final report of the CFA committee[7] includes the ISP descriptions of the three candidate architectures and more information about the writing and debugging of ISP descriptions. It also discusses the issue of the correctness of the ISP descriptions and other matters which could not be covered in a short paper.

The second section of this paper presents a brief introduction of ISP through a simplified version of the IBM S/370 ISP description. The third section discusses the separation of architecture vs. implementation details. The fourth section describes the Architectural Research Facility. The fifth section describes the collection of architectural data from the simulation of ISP descriptions and the sixth section concludes the paper by outlining the areas in which future work could benefit from the use of the Architecture Research Facility.

A TYPICAL ISP DESCRIPTION

The ISP notation was developed to formalize the information normally given in basic machine manuals and to supplement or, if possible, eventually replace the "programming reference manuals." Hence its essential requirements were readability, completeness, flexibility, and brevity.

The original notation was introduced for descriptive purposes and, in the context of a book,[6] certain ambiguities were permitted. For more formal uses, the notation had to be revised and a language named ISPL was developed between 1973–1975.[4] Further developments on the notation continue at CMU, and a language tentatively named ISPS is being implemented. For the remainder of this paper we shall refer exclusively to ISPL, the dialect used in the description of the CFA architectures.

The example shown in Figure 1 is derived from the IBM S/370 ISP description. We will only present the main declarations and the instruction interpretation cycle.**

The control flow for all instructions in Figure 1 follows a well defined path. The main body of the ISP description is defined by the Run procedure which continuously performs a loop of instruction cycles (IFetch followed by IExec). After an instruction has been executed, a special section of code (INT) is executed. INT checks for the presence of exceptional conditions (errors or external interrupts) and performs the proper context switching to handle these conditions.

The instruction fetch section (IFetch) reads the first half-word of the instructions and from the first two bits (Instr⟨Ø⟩ and Instr⟨1⟩) it computes the length of the instruction (PSW⟨32:33⟩) and updates the program counter (PSW ⟨40:63⟩). IFetch then proceeds to read one or two more half-words, the rest of the instruction.

The instruction execution section (IExec) uses the first two bits of the instruction (Instr⟨Ø:1⟩) to select an instruction-type specific section. The RR, RX, RSSI, and SS sections handle the corresponding instruction types. RX, RSSI, and SS begin by computing the effective address of the operand(s). After this step is completed the next 6 bits of the instruction (Instr⟨2:7⟩) are used to select a "routine" which describes the behavior of the instruction.

If any errors are detected during the instruction cycle (address boundary errors, illegal operations, storage protections, etc.) the rest of the instruction is aborted and the proper error code is set in the PSW. This premature termination allows the interrupt handler (INT) to take care of the situation (the usual mechanism is to switch PSWs thus automatically starting the execution of interrupt specific system routines).

We have tried to keep the example as simple as possible by avoiding any details beyond those necessary to follow the example. In particular, the reader might have noticed that we were making explicit references to fields of the Instruction Register (Instr) and the Program Status Word (PSW). It is clear that when we deal with large descriptions such explicit references tend to become cumbersome and error prone.* The following section deals with the issues of how to improve the readability and writability of ISP descriptions by using abstractions like pseudo-registers, procedures, temporary registers, etc.

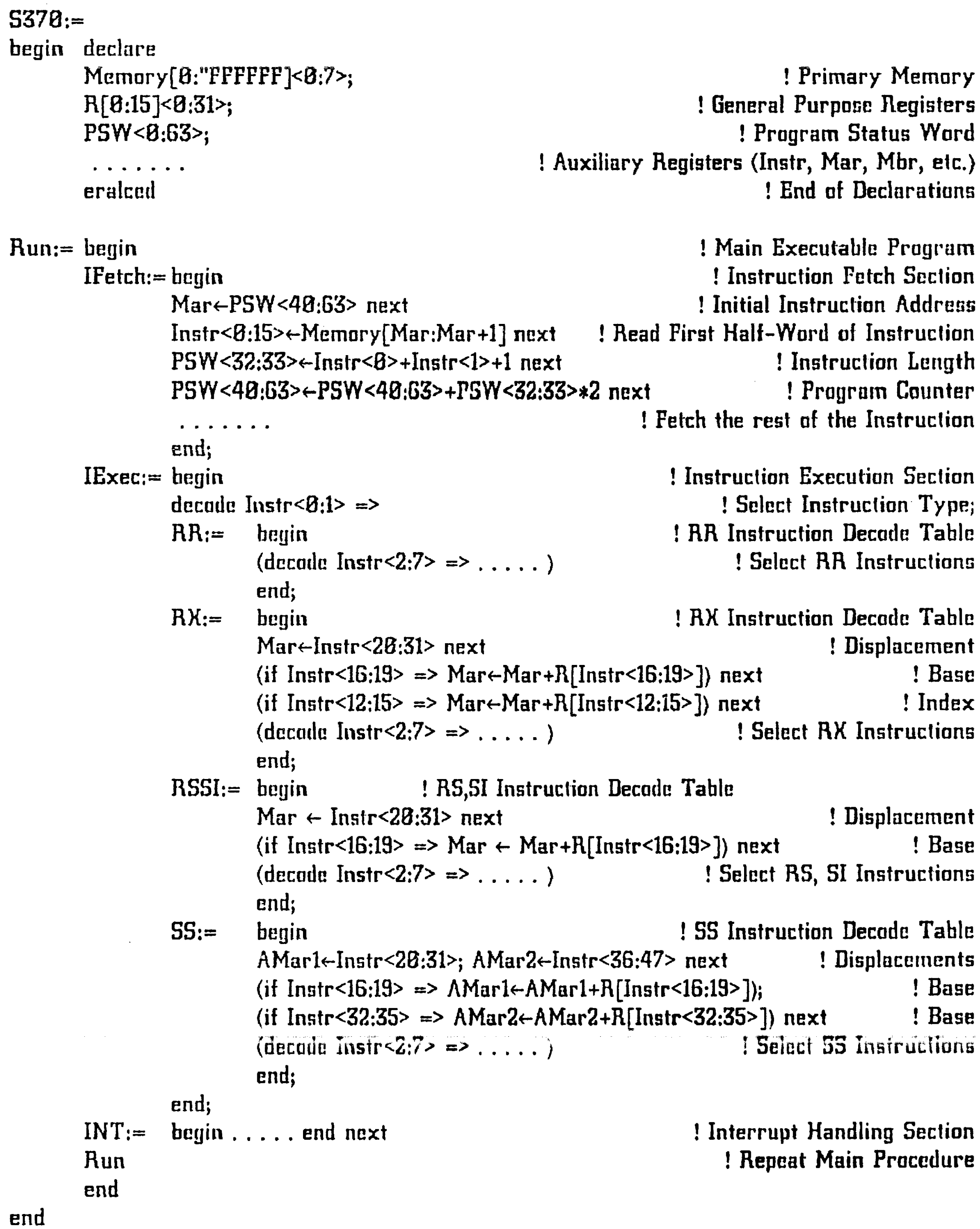
ABSTRACTIONS AND IMPLEMENTATION DEPENDENCIES

ISP can be viewed as a programming language for a specific class of algorithms, i.e., Instruction Set Processors or Architectures. Ideally, a language to describe architectures should avoid the specification of any implementation details. Any components introduced beyond these are unnecessary for the programmer of the machine and might even bias the implementor working from the description. While these items must appear in a description of an implementation, a problem arises when describing a family of machines where the abstractions and/or algorithms may vary across members of the family. The rest of this section illustrates this problem.

* The CFA selection committee adopted the definition of architecture proposed by the designers of the IBM S/360: "The term *architecture* is used here to describe the attributes of a system as seen by the programmer, i.e., the conceptual structure and functional behavior, as distinct from the organization of the data flow and control, the logical design, and the physical implementation."[1]

** In order to keep the examples within the space limitations of this paper, we have taken some minor liberties with the syntax of ISPL. These alterations should not overly confuse readers familiar with ISPL.

* Even though some portions of the Architectures were left out of the ISP descriptions, notably the Floating-Point Instructions, the ISP descriptions used in this project are non-trivial computer programs. Each description takes between 30 and 40 pages of code. The size of the descriptions (1445 lines for the PDP-11, 2345 lines for the Interdata 8/32, and 2132 lines for the IBM S/370) reflects the size of the instruction set, not necessarily the complexity of the architecture.



```
S370:=
begin  declare
       Memory[0:"FFFFFF]<0:7>;                                   ! Primary Memory
       R[0:15]<0:31>;                                  ! General Purpose Registers
       PSW<0:63>;                                            ! Program Status Word
       .......                      ! Auxiliary Registers (Instr, Mar, Mbr, etc.)
       eralced                                            ! End of Declarations

Run:= begin                                             ! Main Executable Program
       IFetch:= begin                                 ! Instruction Fetch Section
              Mar←PSW<40:63> next                   ! Initial Instruction Address
              Instr<0:15>←Memory[Mar:Mar+1] next ! Read First Half-Word of Instruction
              PSW<32:33>←Instr<0>+Instr<1>+1 next              ! Instruction Length
              PSW<40:63>←PSW<40:63>+PSW<32:33>*2 next             ! Program Counter
              .......                            ! Fetch the rest of the Instruction
              end;
       IExec:= begin                              ! Instruction Execution Section
              decode Instr<0:1> =>                     ! Select Instruction Type;
              RR:=   begin                          ! RR Instruction Decode Table
                     (decode Instr<2:7> => ..... )          ! Select RR Instructions
                     end;
              RX:=   begin                          ! RX Instruction Decode Table
                     Mar←Instr<20:31> next                          ! Displacement
                     (if Instr<16:19> => Mar←Mar+R[Instr<16:19>]) next        ! Base
                     (if Instr<12:15> => Mar←Mar+R[Instr<12:15>]) next       ! Index
                     (decode Instr<2:7> => ..... )          ! Select RX Instructions
                     end;
              RSSI:= begin           ! RS,SI Instruction Decode Table
                     Mar ← Instr<20:31> next                        ! Displacement
                     (if Instr<16:19> => Mar ← Mar+R[Instr<16:19>]) next      ! Base
                     (decode Instr<2:7> => ..... )      ! Select RS, SI Instructions
                     end;
              SS:=   begin                          ! SS Instruction Decode Table
                     AMar1←Instr<20:31>; AMar2←Instr<36:47> next   ! Displacements
                     (if Instr<16:19> => AMar1←AMar1+R[Instr<16:19>]);        ! Base
                     (if Instr<32:35> => AMar2←AMar2+R[Instr<32:35>]) next    ! Base
                     (decode Instr<2:7> => ..... )          ! Select SS Instructions
                     end;
              end;
       INT:=  begin ..... end next                     ! Interrupt Handling Section
       Run                                                ! Repeat Main Procedure
       end
end
```

Figure 1—A simplified version of the IBM S/370 ISP description

Abstractions

An ISP description written using only the architectural components would not only be unreadable but also unwritable. Some form of abstraction is required. The following subsections demonstrate this point by introducing pseudo-registers, procedures, and temporary registers. These abstractions may or may not have a counterpart in some or all physical implementations of the ISP description.

Pseudo-Registers

When writing an ISP description for a real machine it immediately becomes apparent that describing everything in terms of just the components of the architecture would lead to a cumbersome and unreadable description. The concept of a pseudo-register to rename a frequently used field of a register greatly relieves this problem. For example, consider the PDP-11 which has an autoincrement addressing mode. During the address computation an architecture register, pointed to by a subfield of the current instruction, must be incremented. Dealing only with components of the architecture would yield an expression like: R[M[Pc]⟨2:∅⟩]←R[M[Pc]⟨2:∅⟩]+2 where M[Pc] represents the current instruction in memory, pointed to by the program counter. Introducing the pseudo-register Ir (instruction register) for the current instruction would yield: R[Ir⟨2:∅⟩]←R[Ir⟨2:∅⟩]+2. We could further define a pseudo-register, Dr (for destination register), for the frequently used three bit subfield Ir⟨2:∅⟩, as in: R[Dr]←R[Dr]+2.

The pseudo-registers may suggest a register (e.g., Ir) or a set of wires (e.g., Dr) in some physical implementation. In reality they may have no physical correspondence at all. In any event, pseudo-registers are a useful and necessary abstraction for readable (and writable) ISP descriptions. However creating pseudo-registers for infrequently used fields or using obscure names may defeat the usefulness of this abstraction leading to reader confusion and excessive page flipping to find definitions.

Procedures

Just as there are frequently used register fields in a machine description, there are frequently used sequences of operations. Forming these operations into procedures greatly enhances readability.

For example, consider operand fetching. Every machine has a more or less complicated effective address calculation that is performed when accessing these operands. A memory reference to a destination operand might appear as: M[Dest] where Dest is a procedure for calculating the effective address of the destination operand. Without procedures the same reference for the PDP-11 would appear as shown in Figure 2. The situation would further be aggravated if the effective address had to be processed by some form of memory management which provides for address translation and rights checking. These operations would have to be performed in the description on top of the effective address calculation. It should be noted that many minicomputers and all larger computers have some form of memory management.

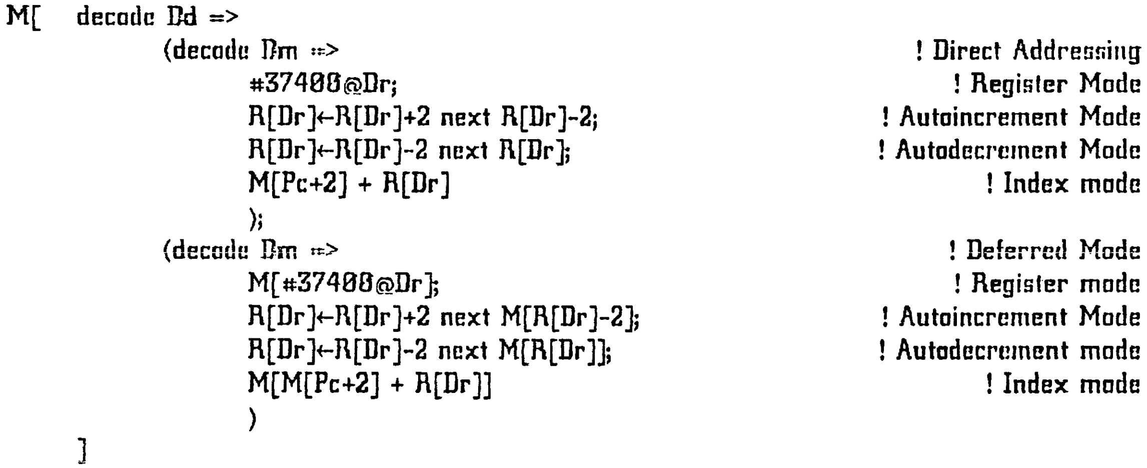
Temporaries

Occasionally readability is improved by introducing a temporary register in cases where the operands before and after the operation are required or a complex result is used repeatedly. Figure 3 shows a portion of the memory management procedures for the PDP-11.

The Read procedure shows the translation of a virtual address into a physical address. A temporary Memory Address Register (Mar) initially contains the virtual address (the result of the effective address calculation) which is then translated into a physical address in the line that reads:

Mar←(PAR[Temp]⟨11:∅⟩+Mar⟨12:6⟩) @ Mar⟨5:∅⟩ next

The PAR (Page Address Register) and PDR (Page Data Register) arrays contain the necessary address translation information. A bounds check is performed before the actual memory fetch from physical memory. Without the temporary variable Mar the Read procedure would be substantially complicated by having to replace every appearance of the temporary by the complex expression given above. Of course, the temporary variable may or may not have a counterpart in some implementation.



```
M[   decode Dd =>
          (decode Dm =>                                      ! Direct Addressing
               #37400@Dr;                                        ! Register Mode
               R[Dr]←R[Dr]+2 next R[Dr]-2;                  ! Autoincrement Mode
               R[Dr]←R[Dr]-2 next R[Dr];                    ! Autodecrement Mode
               M[Pc+2] + R[Dr]                                      ! Index mode
               );
          (decode Dm =>                                          ! Deferred Mode
               M[#37400@Dr];                                     ! Register mode
               R[Dr]←R[Dr]+2 next M[R[Dr]-2];               ! Autoincrement Mode
               R[Dr]←R[Dr]-2 next M[R[Dr]];                 ! Autodecrement mode
               M[M[Pc+2] + R[Dr]]                                   ! Index mode
               )
     ]
```

Figure 2—Inline effective address calculation

```
Read:=begin
        Temp ← Mar<15:13> next
        Mar ← (PAR[Temp]<11:0> + Mar<12:6>) @ Mar<5:0> next    ! Compute Physical Address
        (if not PDR[Temp]<2:1> => Abort) next
        (if (Mar<12:6> gtr PDR[Temp]<14:8>) and not PDR[Temp]<3> => Abort) next
        (if (Mar<12:6> lss PDR[Temp]<14:8>) and PDR[Temp]<3> => Abort) next
        .......                                        ! Read from Physical Memory
        end;
```

Figure 3—A portion of the PDP-11 memory management

Implementation dependencies

There are multiple examples of details that must be specified in an implementation description but do not belong in an architecture description. Typically, these are features that exhibit model dependencies. For instance, in the specification of the interrupt handling facility of a computer system, it could be the case that because of cost/performance requirements, different models must respond to simultaneous interrupts in different orders. An ISP description must by its very nature describe a specific order of interrupt trapping, thus losing a degree of freedom that one might wish to provide the machine implementors.

Figure 4 shows how the specific order in which simultaneous interrupts are fielded is build into an ISP description. Individual bits of INTVEC indicate the presence of a pending interrupt of a given priority. When only one interrupt is pending the proper context switching will take place. When more than one is pending there will be multiple context swaps and lower priority interrupts will be delayed to be processed later (the "new PSW" associated with a low priority interrupt will be stored into the "old PSW" position associated with a higher level interrupt).

It is not clear whether having to be specific about ordering of interrupts or similar events is a bad practice. Although one can claim that machine designers will be constrained in their choice of designs, the fact still remains that somebody must write the interrupt handling software, and for these programmers the order of interrupt fielding is important. This type of dilemma occurs quite often when dealing with ISP descriptions. The solution might be simply to write model-dependent ISP procedures whenever this conflict arises and then indicate in the ISP description which version of a given procedure must be implemented for a given model.

Another problem with implementation dependencies is that the definition of the input/output behavior of an instruction might actually imply a particular implementation. For example, consider the PDP-11 Subtract instruction. The carry condition code (C) is set according to the borrow during the subtraction. The PDP-11 Processor Handbook

```
Int:=   begin
        Temp←PSW<32:33> next                           ! Save Instruction Length
        (if INTVEC<0> AND PSW<13> =>                ! Handle Priority (1) Interrupts

                .......
                ) next
        (if INTVEC<1> =>                            ! Handle Priority (2) Interrrupts

                .......
                ) next
        (if INTVEC<2> =>

                .......
                ) next
        (if INTVEC<3> AND PSW<0:7> =>               ! Handle Priority (3) Interrupts

                .......
                ) next
        (if INTVEC<4> AND IOMSK =>                  ! Handle Priority (4) Interrupts

                .......
                ) next
        PSW<16:31>←0; PSW<32:33>←Temp ! Reset Instruction Length & Interrupt Code
        end;
```

Figure 4—Explicit interrupt processing order in the IBM S/370

```
5 - 3 = 2 (no borrow)   3 - 5 = -2 (borrow)

   0101                    0011        Subtracting
   0011                    0101
   ----                    ----
 0 0010                  1 1110
borrow                  borrow

   0101                    0011        Adding Two's Complement
   1101                    1011
   ----                    ----
 1 0010                  0 1110
carry                   carry
```

Figure 5—Implementation dependant condition code setting

describes the setting of the C bit as:

"C condition code is cleared if there was a carry from the most significant bit of the result, set otherwise."

This definition implicitly assumes that subtraction is implemented by forming the two's complement and adding. Figure 5 illustrates the situation. Consider four-bit numbers and the two methods to perform subtractions, by using a subtractor, and by using an adder after forming the two's complement.

In the adder case, the carry is the complement of the borrow which is exactly the definition given by the PDP-11 Processor Handbook. The ISP description of the setting of C becomes:

```
C←(dest−source)⟨16⟩;                      ! Subtraction
C←NOT (dest+NOT(source)+1)⟨16⟩;            ! Addition
```

As in the previous example (the order of interrupt handling), a complete algorithm had to be given. In this case, the subtractor/borrow algorithm is preferred since it presupposes only the properties of the two's complement number system. However, if an alternate implementation (such as forming the two's complement and adding) is utilized, then the implementor should be aware of possible changes in other algorithms in the ISP description.

THE ARCHITECTURE RESEARCH FACILITY

The facility used for the data collection phase of the CFA project is depicted in Figure 6. Reference 4 explains in full detail the features of the ISP compiler and simulator. Some familiarity with their capabilities is needed in order to

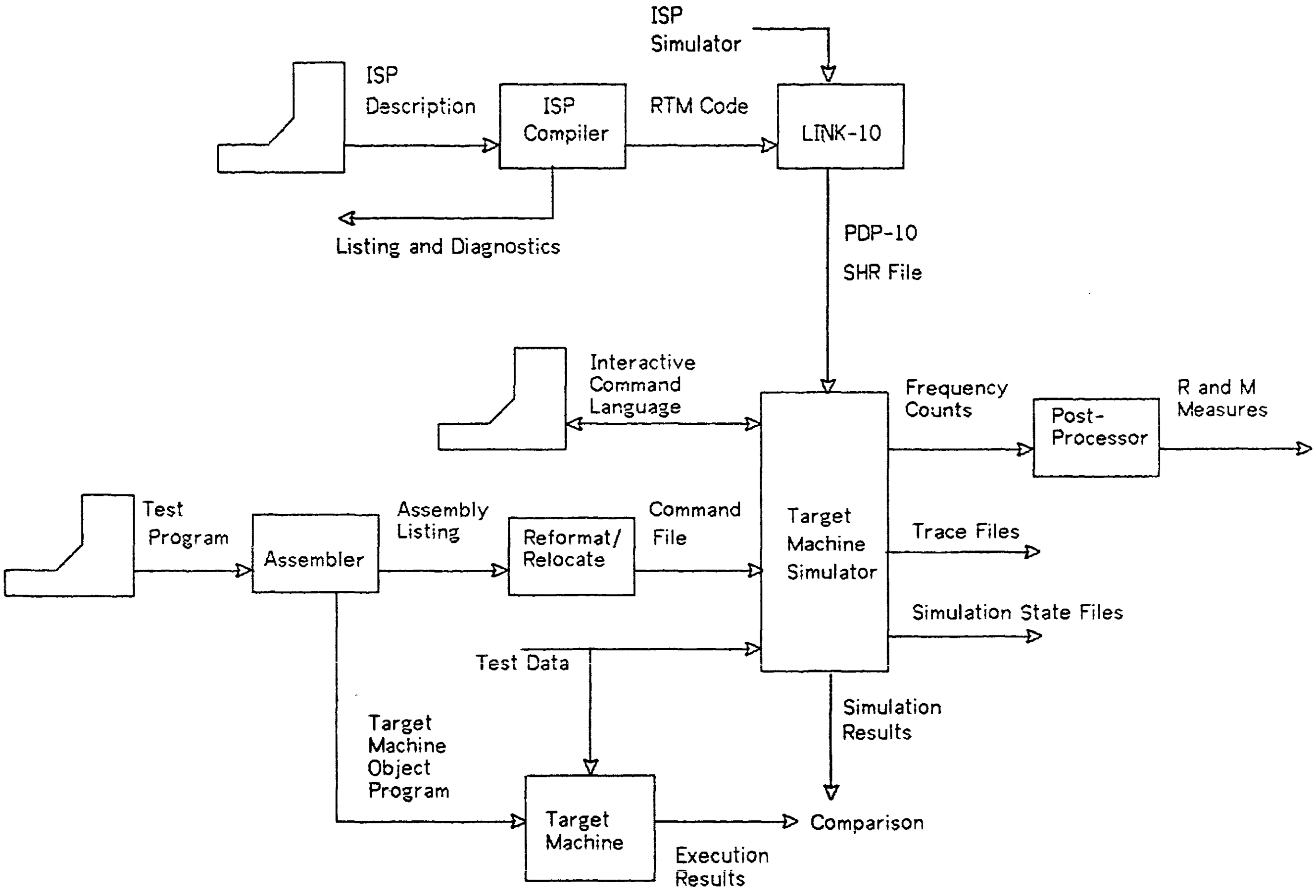


Figure 6—Test program execution under ARF

understand the data collection phase described later. The following paragraphs attempt to satisfy this need.

The ISP compiler produces code for a hypothetical machine, dubbed the Register Transfer Machine (RTM). The "object code" produced by the compiler can be linked together with a program which is capable of interpreting RTM instructions. This separation between the ISP description, the RTM code, and the RTM interpreter allows the simulation of arbitrary, user defined architectures. The result of linking the RTM code with the RTM interpreter is a running program, a simulator.

The simulator accepts commands from a teletype or user designated command file. The state of the simulator can be dumped to a command file which can be read at a future date when the simulation is continued. Command files can also be used to load programs and data into the simulated target machine memory and registers.

Debugging of test programs and ISP descriptions

Most of the test programs were debugged and run on the real machines. Other programs were executed exclusively under the simulator. The latter included those programs using privileged instructions that were not directly available to non-system programmers (e.g., interrupt and I/O handlers). Results from the actual runs, whenever available, were used to check the simulated execution.

Only minor modifications and corrections to the ISP descriptions were performed during the data collection phase. The largest unforeseen problem with the test programs was presented by the memory management feature of the PDP-11 which was based on the PDP-11/40. The test programs which made use of this feature had been tested on a PDP-11/45 which uses different Unibus addresses for the memory management registers. This difference required minor modifications in the test programs. Most other problems were of a simpler nature and required only a few minutes to correct. It should be noted here that the simulator facility was also used to debug some programs for the Interdata 8/32 before they were executed on the real machine. This was dictated by the fact that no 8/32 was available near CMU and a large turn-around time (several days) would have complicated the debugging of the test programs.

Preparation of simulation tests

The ISP simulator provides commands for the loading and initialization of the simulated machine memory and internal registers. The single most important feature of the command language which permitted the fast execution and collection of statistics was the ability to read command files containing the test programs to be executed. The command language cannot handle programs in symbolic form (assembly language); it requires the preassembly of the programs into absolute, numeric, code. To get around this problem, a set of utilities was developed at CMU which permitted the transformation of assembly listings prepared by the real machine's assembler into simulation command files. This operation was performed off-line as shown in Figure 6.

Figures 7 and 8 show the transcript of a typical session using the ISP simulator. The session consists of running one of the test programs (Bit Test, Set, and Reset) on the PDP-11. The input for a simulation session consists of several files prepared off-line. These files include: the test program (derived from the assembly listing), a driver (simulation commands used to initialize the parameters for the test program), and finally, a command file with a list of those ISP procedures which must be "opaqued" (these are the procedures during which the activity counters are disabled). A typical command file, derived from an assembler listing is shown in Figure 9. This was the test program used in the sample simulator session shown in Figures 7 and 8.

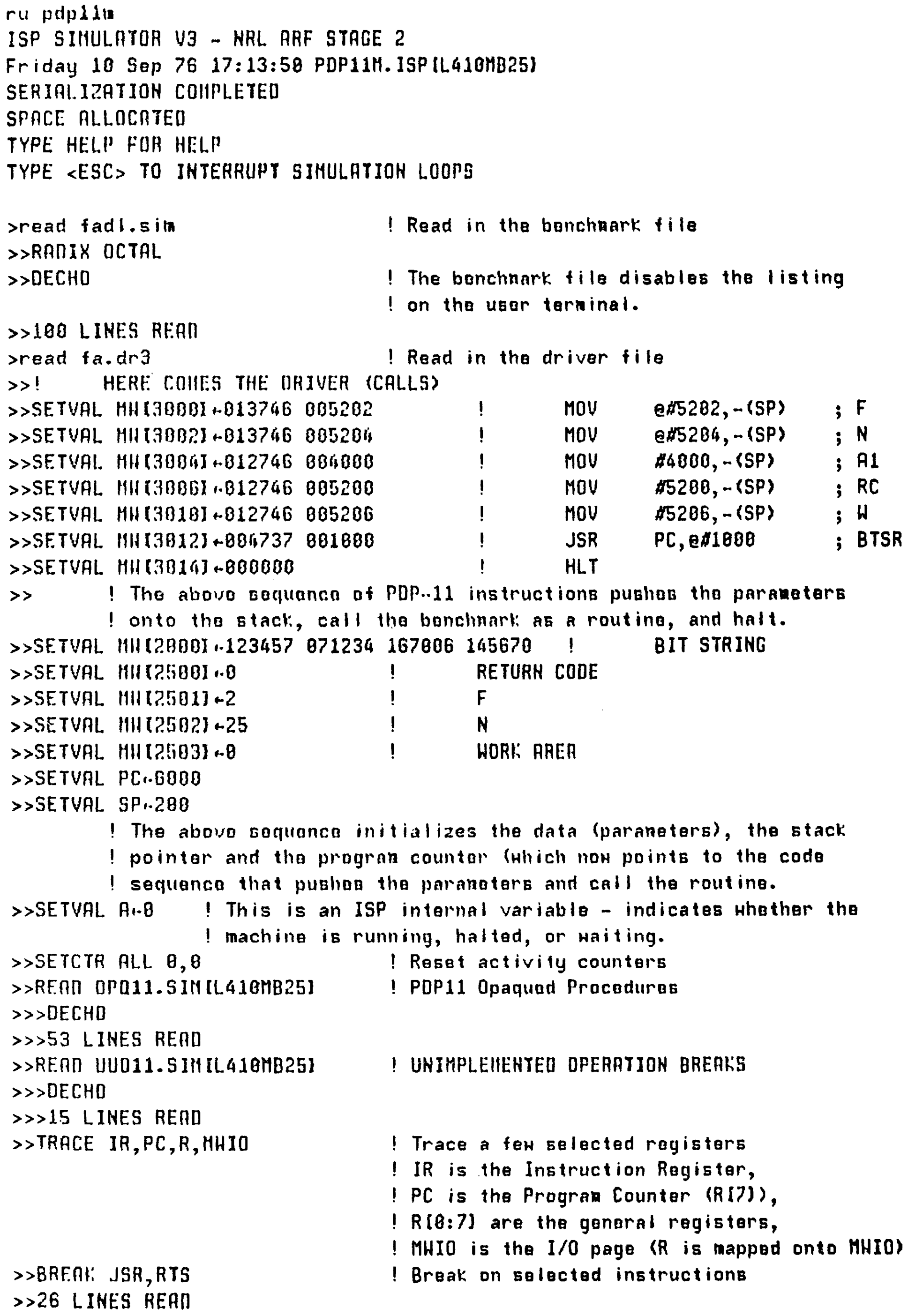
Instrumentation

The ISP simulator permits the instrumentation of an ISP description by associating activity counters with each of the machine registers and memories. These counters allow the collection of statistics indicating the number of times each component of the machine is read from or written into. A separate counter is kept for each label in the ISP description. Labels are included in the ISP descriptions to identify machine instructions, addressing modes, loops (used to describe vector-like instructions like move character on the S/370), as well as other ISP procedures. During the execution of the test programs, a data base was created by collecting dumps of the counters after each test case was completed. The files containing the counters were then processed by other, off-line, programs in order to arrive at the M and R measures described in a later sction.

Artificial labels in the ISP descriptions

Certain modifications not normally needed were made to the ISP descriptions to aid in the collection of data during the running of the test programs for the CFA project. Several labels and "do-nothing" procedures were added to identify certain phases in the instruction interpretation algorithm and to measure selected events (e.g., different addressing modes). The labels added to count these events are clearly not part of the architecture or even the implementation.

Figure 10 shows an example extracted from the S/370 ISP Description. It shows the use of artificial labels to identify different addressing modes for the RX instruction set. According to the definition of the S/360 and S/370 architectures, the RX instructions can specify both a base and an index register to be added together with the displacement field of the instruction to compute the address of the memory operand. The architecture further specifies that R[Ø], when specified as either a base or index register does not take place in the effective address calculation, i.e.,



```
ru pdp11m
ISP SIMULATOR V3 - NRL ARF STAGE 2
Friday 10 Sep 76 17:13:50 PDP11M.ISP[L410MB25]
SERIALIZATION COMPLETED
SPACE ALLOCATED
TYPE HELP FOR HELP
TYPE <ESC> TO INTERRUPT SIMULATION LOOPS

>read fadl.sim                  ! Read in the benchmark file
>>RADIX OCTAL
>>DECHO                         ! The benchmark file disables the listing
                                ! on the user terminal.
>>100 LINES READ
>read fa.dr3                    ! Read in the driver file
>>!     HERE COMES THE DRIVER (CALLS)
>>SETVAL MW[3000]←013746 005202        !     MOV    @#5202,-(SP)    ; F
>>SETVAL MW[3002]←013746 005204        !     MOV    @#5204,-(SP)    ; N
>>SETVAL MW[3004]←012746 004000        !     MOV    #4000,-(SP)     ; A1
>>SETVAL MW[3006]←012746 005200        !     MOV    #5200,-(SP)     ; RC
>>SETVAL MW[3010]←012746 005206        !     MOV    #5206,-(SP)     ; W
>>SETVAL MW[3012]←004737 001000        !     JSR    PC,@#1000       ; BTSR
>>SETVAL MW[3014]←000000               !     HLT
>>      ! The above sequence of PDP-11 instructions pushes the parameters
        ! onto the stack, call the benchmark as a routine, and halt.
>>SETVAL MW[2000]←123457 071234 167006 145670   !      BIT STRING
>>SETVAL MW[2500]←0             !      RETURN CODE
>>SETVAL MW[2501]←2             !      F
>>SETVAL MW[2502]←25            !      N
>>SETVAL MW[2503]←0             !      WORK AREA
>>SETVAL PC←6000
>>SETVAL SP←200
        ! The above sequence initializes the data (parameters), the stack
        ! pointer and the program counter (which now points to the code
        ! sequence that pushes the parameters and call the routine.
>>SETVAL A←0     ! This is an ISP internal variable - indicates whether the
                 ! machine is running, halted, or waiting.
>>SETCTR ALL 0,0                ! Reset activity counters
>>READ OPO11.SIM[L410MB25]      ! PDP11 Opaquod Procedures
>>>DECHO
>>>53 LINES READ
>>READ UUO11.SIM[L410MB25]      ! UNIMPLEMENTED OPERATION BREAKS
>>>DECHO
>>>15 LINES READ
>>TRACE IR,PC,R,MWIO            ! Trace a few selected registers
                                ! IR is the Instruction Register,
                                ! PC is the Program Counter (R[7]),
                                ! R[0:7] are the general registers,
                                ! MWIO is the I/O page (R is mapped onto MWIO)
>>BREAK JSR,RTS                 ! Break on selected instructions
>>26 LINES READ
```

Figure 7—Initialization of a simulation run

R[Ø] should be specified whenever one of these two components (base or index) is missing. In the above example four dummy in-line procedures where introduced to count the number of times each possible combination of base/index modes occurs. Thus RXØØØØ is "executed" whenever R[Ø] is specified as both the base and the index register. RXØØX2 is "executed" whenever R[Ø] is used as the base register and any of R[1:15] is used as the Index register. RXB1ØØ is "executed" whenever R[Ø] is specified as the index register and any of R[1:15] is specified as the base register. Finally, RXB1X2 is "executed" whenever R[Ø] is not specified as either the base or index registers. NOP is a dummy procedure which does not have any side effects.

ARCHITECTURE PARAMETERS

As a means of comparing architectures, three measures were defined for the CFA project:[9]

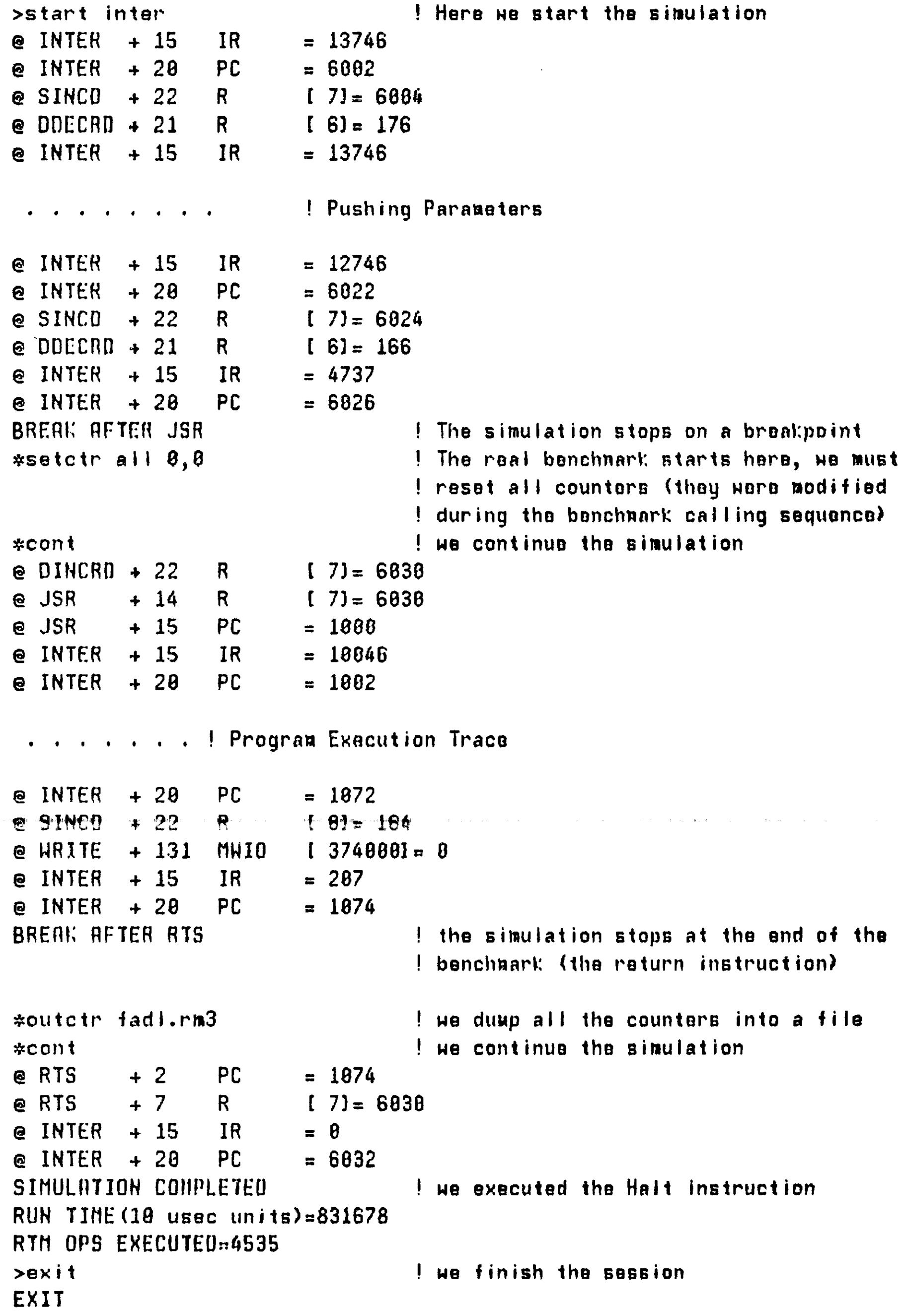
Measure of Space

S The number of bytes used to represent a test program.

Measures of Execution Time

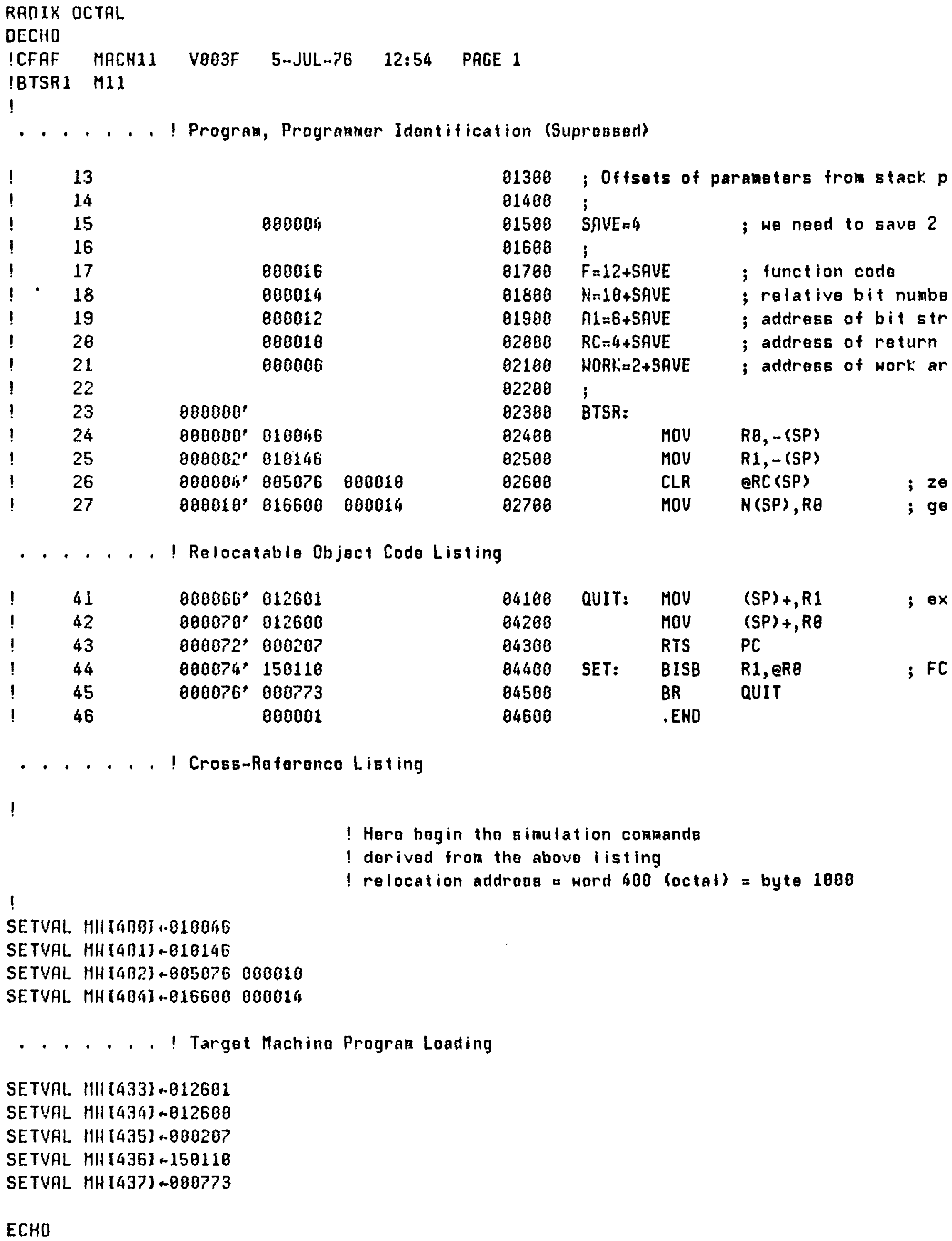
M The number of bytes transferred between primary memory and the processor during the execution of the test programs.

R The number of bytes transferred among internal registers of the processor during execution of the test program.



```
>start inter                         ! Here we start the simulation
@ INTER  + 15   IR      = 13746
@ INTER  + 20   PC      = 6002
@ SINCO  + 22   R       [ 7]= 6004
@ DDECRD + 21   R       [ 6]= 176
@ INTER  + 15   IR      = 13746

 . . . . . . . .        ! Pushing Parameters

@ INTER  + 15   IR      = 12746
@ INTER  + 20   PC      = 6022
@ SINCO  + 22   R       [ 7]= 6024
@ DDECRD + 21   R       [ 6]= 166
@ INTER  + 15   IR      = 4737
@ INTER  + 20   PC      = 6026
BREAK AFTER JSR                      ! The simulation stops on a breakpoint
*setctr all 0,0                      ! The real benchmark starts here, we must
                                     ! reset all counters (they were modified
                                     ! during the benchmark calling sequence)
*cont                                ! we continue the simulation
@ DINCRD + 22   R       [ 7]= 6030
@ JSR    + 14   R       [ 7]= 6030
@ JSR    + 15   PC      = 1000
@ INTER  + 15   IR      = 10046
@ INTER  + 20   PC      = 1002

 . . . . . . . . ! Program Execution Trace

@ INTER  + 20   PC      = 1072
@ SINCO  + 22   R       [ 8]= 104
@ WRITE  + 131  MWIO    [ 374000]= 0
@ INTER  + 15   IR      = 207
@ INTER  + 20   PC      = 1074
BREAK AFTER RTS                      ! the simulation stops at the end of the
                                     ! benchmark (the return instruction)

*outctr fadl.rm3                     ! we dump all the counters into a file
*cont                                ! we continue the simulation
@ RTS    + 2    PC      = 1074
@ RTS    + 7    R       [ 7]= 6030
@ INTER  + 15   IR      = 0
@ INTER  + 20   PC      = 6032
SIMULATION COMPLETED                 ! we executed the Halt instruction
RUN TIME(10 usec units)=831678
RTM OPS EXECUTED=4535
>exit                                ! we finish the session
EXIT
```

Figure 8—Program execution trace



```
RADIX OCTAL
DECHO
!CFAF   MACN11   V003F   5-JUL-76   12:54   PAGE 1
!BTSR1  M11
!
 . . . . . . . . ! Program, Programmer Identification (Supressed)

!     13                                   01300   ; Offsets of parameters from stack p
!     14                                   01400   ;
!     15               000004              01500   SAVE=4          ; we need to save 2
!     16                                   01600   ;
!     17               000016              01700   F=12+SAVE       ; function code
!  •  18               000014              01800   N=10+SAVE       ; relative bit numbe
!     19               000012              01900   A1=6+SAVE       ; address of bit str
!     20               000010              02000   RC=4+SAVE       ; address of return
!     21               000006              02100   WORK=2+SAVE     ; address of work ar
!     22                                   02200   ;
!     23          000000'                  02300   BTSR:
!     24          000000' 010046           02400           MOV     R0,-(SP)
!     25          000002' 010146           02500           MOV     R1,-(SP)
!     26          000004' 005076  000010   02600           CLR     @RC(SP)          ; ze
!     27          000010' 016600  000014   02700           MOV     N(SP),R0         ; ge

 . . . . . . . . ! Relocatable Object Code Listing

!     41          000066' 012601           04100   QUIT:   MOV     (SP)+,R1         ; ex
!     42          000070' 012600           04200           MOV     (SP)+,R0
!     43          000072' 000207           04300           RTS     PC
!     44          000074' 150110           04400   SET:    BISB    R1,@R0           ; FC
!     45          000076' 000773           04500           BR      QUIT
!     46                  000001           04600           .END

 . . . . . . . . ! Cross-Reference Listing

!
                                  ! Here begin the simulation commands
                                  ! derived from the above listing
                                  ! relocation address = word 400 (octal) = byte 1000
!
SETVAL MW[400]←010046
SETVAL MW[401]←010146
SETVAL MW[402]←005076 000010
SETVAL MW[404]←016600 000014

 . . . . . . . . ! Target Machine Program Loading

SETVAL MW[433]←012601
SETVAL MW[434]←012600
SETVAL MW[435]←000207
SETVAL MW[436]←150110
SETVAL MW[437]←000773

ECHO
```

Figure 9—A command file derived from an assembly listing

The S measure is a static parameter which can be computed independently of the ISP description. For the purposes of this paper we will restrict the discussion to the other two measures. The actual computation of the M and R measures was done through a semiautomatic process. The raw data collected from the simulator was used to count frequencies of instructions and addressing modes. These counters were multiplied by certain hand calculated factors in order to arrive at the M and R measures for each test program. Ideally, the ISP simulator should perform the

```
RX:=    begin
        Mar←Instr<20:31> next
        (decode (Instr<16:19> NEQ 0)@(Instr<12:15> NEQ 0)=>
        \00     RX0000:=      (NOP);                        ! No Base, No Index
        \01     RX00X2:=      (NOP);                        ! No Base, Indexing
        \10     RXB100:=      (NOP);                          ! Base, No Index
        \11     RXB1X2:=      (NOP)                           ! Base, Indexing
                ) next
        (if Instr<16:19> => Mar←Mar+R[Instr<16:19>]) next
        (if Instr<12:15> => Mar←Mar+R[Instr<12:15>]) next
        (decode Instr<2:7> =>
        . . . . . .                                 ! Select RX Instructions
                )
        end;
```

Figure 10—Use of artificial labels

entire operation and this would be a better approach, less subject to human errors. We had to use the hand computed factors due to our inability to determine the influence of the ISP writing style on the architecture parameters as defined above. The actual results of the experiments (the M and R measures for each test program) are presented in Reference 9.

The exact methodology for writing ISP descriptions so that the M and R measures can be calculated automatically has yet to be developed. It is clear, however, that a careful control of the counting mechanism is paramount to the collection of meaningful data. During the data collection phase we made use of the following techniques towards this goal.

Opaqued procedures

A simulator command allows the selective masking of in-line and off-line procedures. Masking or opaquing a procedure inhibits all activity counts inside the body of the procedure.

Certain operations, such as incrementing the program counter after an instruction, or the setting of the condition codes as a result of an instruction, do not affect the R measure and should not be counted. This is typical of those actions which, in a reasonable implementation, would be done using ad hoc circuitry, separate from the main operational units of the machine. These operations could be implemented by combinational logic (e.g., setting condition codes from ALU lines), special registers (e.g., using a counter instead of a simple register for the program counter), or even complex sequential networks (e.g., the virtual address translation can be performed using its own arithmetic units and data paths).

Operations like those described above can be easily marked by adding artificial labels to the ISP description and then disabling the counters while the selected operation is being performed.

Pseudo-Register chains

Every component declared in an ISP description has activity counters associated with it. When a register is defined in terms of another register, such as: Pc⟨15:Ø⟩: =R[7]⟨15:Ø⟩; a redefinition chain is established. Accesses higher up in the chain increment all counters lower in the chain but not vice versa. In the above example an access of the Pc causes the register file counter for R to be incremented but accessing R[7] does not increment the program counter (Pc). By establishing appropriate redefinition chains, distinction between access types can be maintained. One variation of this technique is the use of "shadow" registers. For example two instruction registers can be defined: Ir⟨15:Ø⟩:=Irl⟨15:Ø⟩; where Irl is the shadow register. The loading of the Ir from memory is to be counted in the R measure, however, the combinational logic decoding of the instruction and effective addressing mode is not to be counted. The former is performed on Ir, the latter on Irl thus distinguishing the two different types of accesses.

Memory access procedures

Modern machines provide the user with an address space defined in terms of small units of information, typically 8-bit bytes. For convenience, however, the architectures also define larger access units in multiples of bytes. Thus, the IBM S/370 provides bytes, half-words, full-words, and double-words. Since the physical memory is the same, the ISP description must declare the different address spaces by building a redefinition chain in which the different address spaces are declared as "pseudo-memories" so that the M measure component of each address space is properly accounted for.

Machines like the PDP-11 add some more complexity to the issue of having multiple address spaces. The PDP-11 architecture defines the concept of an I/O page as a reserved portion of the address space, not necessarily implemented as a physical memory. Addresses in the upper 4K bytes of the PDP-11 are used to address I/O devices, machine registers, etc. Addresses in the I/O page must be handled differently when computing the M measure. If one attempts to include in-line address checks in the ISP description, the description quickly becomes bulky and unreadable. A satisfactory solution is simply to define memory access procedures (Read and Write), which can then be properly instrumented, thus enabling the automatic computation of the M measure.

Temporary Registers

The automatic computation of the R measure is more difficult. In an ISP description there are three types of registers to consider: architectural, standard implementation, and temporaries. Architectural registers and certain standard implementation registers (instruction register, memory address register, and memory buffer register) can be handled using the same techniques used to automate the M measure (declaration chains and encapsulating procedures). Handling temporary registers presents a more difficult problem. The number, type, and manipulation of temporary registers are a matter of writing style.

Architecture parameters which are based solely on architecture registers while ignoring temporary registers introduced for clarity might overlook hidden computations performed on these registers. Unlike the memory, architectural registers, and standard implementation registers, a tightly defined writing style cannot be developed for temporary registers. One solution would be to use well-known expression optimization techniques[10] on the ISP description to uniformly minimize the temporary register activity. Hopefully the optimization would lead to similar results for equivalent algorithms.

Architectural parameters should be independent of the experience, style, and objectives of the ISP writer. This will then guarantee that the ISP descriptions which make use of abstractions (pseudo-registers, procedures, and temporary registers, etc.) to enhance clarity and readability will not be penalized. By the same token, no advantage should be derived from the use of "clever" programming tricks which might attempt to bias the measurements.

ADVANTAGES OF AN ARCHITECTURAL RESEARCH FACILITY

Although for the purposes of this paper we have presented the uses of the ISPL compiler and simulator in the context of a specific project, we should point out the wider range of applications in which a system like ARF* can be of great value.

A simulator as a training tool

In this paper we described how machine language test programs can be executed under the simulator. The implied assumption during the data collection phase was that we were dealing with correct, finished programs. With no extra effort the ISP simulator can be a powerful training device for novice programmers. Speed of simulation is not an issue in this application. Programmers learning a new machine language tend to spend long hours single-stepping via the machine console. An interactive simulator can easily satisfy the needs of these users, while providing much better diagnostic and debugging facilities than a computer console (did you ever see a "help" button on a machine?). ISP descriptions exist for the following machines: DEC PDP-8, PDP-10, PDP-11, IBM S/370, Interdata 8/32, and Intel 8080.

Architecture evaluation

The S, M, and R measures are by no means the only set of architecture parameters one might wish to evaluate. Nothing in the ISP simulator depends upon this particular set of parameters. The instrumentation in the simulator allows counting every event we care to define by simply labelling the event. There is no need to create new procedures which might impact the organization or readability of the description; even a single register transfer operation can be labelled and counted. More details on architecture evaluation can be found in Reference 5.

Experimentation

Once the initial effort of writing an ISP description is accomplished, only moderate effort is required to perturb it to reflect proposed or actual changes in the architecture. Thus the effect of a modification in an architecture can be measured and studied before any funds are committed to the development of a new machine. By a careful design of the ISP description it is possible to pattern a description along the lines of the organization of the physical machine. Thus one would be able to measure and evaluate different models of the architecture. For instance, functional units and data paths can be represented by separate procedures in the ISP description. An ISP description could then be parameterized to invoke these procedures in different order, concurrently or sequentially, with or without intermediate steps, etc. as the different models differ in their implementation. An example might be determining the effect of a cache memory on the apparent instruction execution speed in high performance implementations.

* Soon after the data collection phase was completed, a new, more powerful version of ARF was undergoing final testing. This new system, designated ARF III,[8] was developed by the Naval Research Laboratory. ARF III and its successors will continue to evolve and will be used to model and verify the chosen CFA.

Machine relative software

As the number of different architectures coming into existence increases every year, it is becoming more and more expensive to develop the necessary software support base that allows the effective use of these machines. The availability of user micro-programmable machines enlarges the space of possible architectures to the point that automatic software generation systems will become a necessity. Tools that operate relative to a computer description could represent a significant breakthrough in the manner that computer systems (hardware/software) are designed and evaluated. The Advanced Research Projects Agency (ARPA) of the Department of Defense is currently sponsoring this area of research at CMU and elsewhere.[2]

In the future one can foresee hardware and software automation systems that take as input computer descriptions, and language and problem specifications; and from these, generate operating systems, compilers, and other support and application software automatically. Other areas of current research include automatic diagnostic generation, microcode generation, machine verification, etc.

Formal computer descriptions will play an increasing and important role in the evaluation, procurement, verification, and programming of computers. The ARF facility is a step in this direction.

REFERENCES

1. Amdahl, G. M., G. A. Blaauw, and F. P. Brooks, "Architecture of the IBM System/360," *IBM Journal of Research and Development*, Vol. 8, No. 2, April 1964, pp. 87–101.
2. Barbacci, M. R. and D. P. Siewiorek, *Some Aspects of the Symbolic Manipulation of Computer Descriptions,* Department of Computer Science, Carnegie-Mellon University, July 1974.
3. Barbacci, M. R., "A Comparison of Register Transfer Languages for Describing Computers and Digital Systems," *IEEE Transactions on Computers,* Vol. C-24, No. 2, February 1975, pp. 137–149.
4. Barbacci, M. R., *The Symbolic Manipulation of Computer Descriptions: ISPL Compiler and Simulator, Technical Report, Department of Computer Science, Carnegie-Mellon University, 1976.*
5. Barbacci, M. R., S. H. Fuller, and D. P. Siewiorek, *A Methodology for Comparative Computer Architecture,* Department of Computer Science, Carnegie-Mellon University, 1977.
6. Bell, C. G. and A. Newell, *Computer Structures: Readings and Examples,* McGraw-Hill Book Company, New York, 1971.
7. *Computer Family Architecture Selection Committee Final Report,* Naval Research Laboratory, Washington, D.C., December 1976. A collection of volumes describing each aspect of the CFA project.
8. Elovitz, H. S. and R. A. Parker, *The Architecture Research Facility (ARF) User's Guide*, Technical Memorandum 5403-472, Naval Research Laboratory, Washington, D.C., October 1976.
9. Fuller, S. H., W. Burr, P. Shaman and D. Lamb, "Evaluation of Computer Architectures Via Test Programs," *AFIPS Conference Proceedings,* Vol. 46, 1977. National Computer Conference.
10. Wulf, W. et al., *The Design of an Optimizing Compiler,* American Elsevier, Programming Language Series, New York, 1975.

Evaluation of the software bases of the candidate architectures for the military computer family

by JAMES WAGNER and EDWARD LIEBLEIN
U. S. Army Electronics Command
Fort Monmouth, New Jersey

and

JORGE RODRIGUEZ
Softech Inc.
Waltham, Massachusetts

and

HAROLD STONE
University of Massachusetts
Amherst, Massachusetts

ABSTRACT

One of the Army-Navy Committees' primary motivations for selecting a proven computer architecture for the basis of a family of software compatible military computers is the potential utility of the existing software base. Therefore, the evaluation of the software bases of the candidate architectures constituted one very important factor in the final selection. This paper describes that software base evaluation process and results thereof.

INTRODUCTION

One of the primary reasons for adopting the architecture of an existing successful computer family as the architecture for a future Military Computer Family (MCF) is the potential utility of the existing software base. This concept was supported almost unanimously by members of the Computer Family Architecture (CFA) Selection Committee and it was further agreed that an assessment of the software bases of the three finalists should be made and should constitute one important factor in the final selection of an architecture for the MCF.

It was apparent that the amount of available software would be a major factor in the determination of overall comparative life cycle costs. Consequently, in lieu of an isolated ranking of the software bases of the finalist CFAs, it was agreed that the assessment should constitute an input into the life cycle cost models described by Burr et al.[1]

Therefore, it was necessary (1) to define what was meant by the term "software base," (2) to determine what could and should be measured, and (3) to develop and utilize a methodology for timely assessment of the value of the three software bases that facilitated a relative quantitative ranking suitable for input into the life cycle cost model.

This paper describes the evaluation approach taken as well as the results of the quantitative comparison of the software bases of the three architecture finalists, IBM 360, DEC PDP-11 and Interdata 8/32.

TECHNICAL APPROACH

In the commercial world there are many applications software packages that are transportable across applications (e.g., payroll systems and inventory systems). The military world has not been able to achieve such transportability, primarily due to the disparity among applications and the complex, time-critical, and sensor oriented functions involved.

After an attempt to find commonality among existing military applications software failed, it was agreed to exclude applications software from the domain of the software base. Only support software, which includes software used for producing, modifying, analyzing, and testing a computer-based system, would be evaluated. It will be seen that the evaluation methodology involves subjective quantification of applicability/importance of items in the software base. To aid this process, the software base was structured with respect to the development environment, software development activities, and software tool interdependencies.

The military software development environment

In the past, far too many small to medium scale military computer systems were developed using the computer that was to go into the final operational system as its own software development environment. The consequences of this approach were often disastrous since these development environments were virtually devoid of the very broad spectrum of powerful support software tools that now exist on larger computers.

Many of the activities of software development are independent of the target computer architecture (e.g., planning, requirements determination, system design, program library management, documentation, and configuration management). Many of these activities are now being aided by software tools. The possibility of improving the development process through the use of such tools has been recognized and there is a trend toward even greater use of computer to support these activities.[2-4] The culmination of this trend seems likely to be a totally integrated software development system—a relatively large, integrated data base system containing a complete description of the development project including requirement specifications, component representations (design, program code, test data, documentation, etc.), information about the project activities, and relations among the requirements, component representations, and activities. Software tools will be associated with the data base to perform generation, analysis, transformation, and reporting of the project descriptive data. A key design goal for the support system is to automate maintenance of the completeness, currency, and integrity of the project descriptive data. This will have a significant positive effect on the visibility of project progress, product reliability, and maintenance costs. It is clear, however, that such support tools are expensive to develop and require computer systems with extensive memory and I/O capabilities for effective operation. These requirements make it highly unlikely that such tools will ever be developed for operation on the smaller military tactical computers or their commercial counterparts. The potential benefits of using such an integrated support software system and the high cost of developing it constitute strong arguments for centralizing software development support on one type of host computer even though the software will be developed for operation on one of a variety of militarized configurations, not necessarily even of the same inherent architecture.

Other activities such as testing, are dependent on the target computer architecture. One of the objectives of the testing of individual program components and groups of related programs is to provide a basis for confidence that the system and final software tests can be completed without extensive changes. Therefore, it is beneficial for the environment of program testing to closely resemble the operational environment also, although this requirement is not as demanding as the system test requirement. If the host computer architecture differs from that of the target computer, then the amount of final software testing that is possible on the host computer will be limited to that which can be done through simulation (or emulation) of the target computer.

Additional target dependent activities are compilation, assembly, and link-editing. Their dependencies result primarily from the fact that most existing program translators have been built to operate on and produce code for the same computer architecture. (This dependency does not result from the intrinsic requirements of the translation process.)

Software development activities

In this section a model of the tactical software development process will be presented that provides a structure for the support software tools. This structure is utilized in the quantitative assessment of the three finalist architectures.

The software development process is partitioned into the following major activities: (1) Analyze Requirements, (2) Design Software, (3) Build System Tests, (4) Build and Unit-Test Software, (5) Integrate and System Test, and (6) Maintain System. The following paragraphs describe these activities:

Analyze Requirements (Activity 1)

"Analyze Requirements" performs a decomposition of the user needs into the functions of the required system. Following decomposition and the development of a functional model, functions are allocated to hardware, software firmware, and people. The results are then used to search a descriptive catalog of existing systems to locate suitable candidates for reuse or modification. The systems (if any) resulting from this search and any new functions that must be developed may be simulated to determine their gross performance characteristics. This activity is controlled by the analysts' knowledge of the current state of the art and the available budget for the proposed new system. If a decision is taken to proceed with development, a software functional design specification and a project schedule are produced which are used to control the "Design Software" activity.

Design Software (Activity 2)

During this activity, the software functional design specification is used to produce the implementation specification. A library of "proven algorithms" is available to assist in design. The "Design Software" activity may respond to the "Analyze Requirements" activity with "can't design" or "can't meet schedule". The output of this activity is the implementation specification which is used to control the software unit build and integration activities (Activities 4 and 5).

Build System Tests (Activity 3)

"Build System Tests" is the activity that results in the design and construction of system acceptance tests. Note that it is controlled by the same set of functional specifications that control the "Design Software" activity, and that it is unconstrained by and, therefore, may proceed in parallel with "Design Software" and "Build and Unit Test Software." A library of previously constructed tests that are presumably tied to sub-systems is available for reuse as directed by Activity 1. The output of this activity is the set of system test scenarios, drivers and monitors that will control Activity 5.

Build and Unit-Test Software (Activity 4)

During this activity, the implementation specifications produced by Activity 2 are used to produce unit-tested software modules. A library of previously constructed modules is available for reuse (with or without modification).

Integrate and System Test (Activity 5)

During this activity, the modules produced by Activity 4 and the interface and subsystem specifications produced by Activity 3 are used to bind the system components into their final form. The system is then exercised to validate the system using the test scenarios and monitors provided by Activity 3. The final output of this activity is the completed system released to the maintenance, distribution and configuration control activity (Activity 6). Integration and/or test failures are reflected back to the design, test, production or software production activities.

Maintain System (Activity 6)

The final development activity, "Maintain System" is primarily a clearing house and control center for the reception, evaluation, and control of engineering change requests. These requests are routed to the appropriate activity for implementation, design, or analysis. The maintenance function distributes configuration controlled systems to the users, and releases the results of the development effort into the available technology data base.

Structuring of the software base

The software base will be structured, in part, by partitioning the software tool types according to the specific development activities they support. However, this approach may ignore that part of the software base that supports the operation of such tools and does not clearly indicate those tools that support more than one activity. To make such software visible, the software base will be structured further through a layered approach that will provide insight into the relationship among software base components.

There are at least five distinct virtual layers associated with an operational computer system and three layers of software that support the software development process. Layer 0, the innermost layer, represents the bare computer hardware including items such as processors, channels, main storage, mass storage, bulk I/O, archival storage, hardware monitors, terminals, sensors, and communications interface devices. Layers 1 thru 3 "reside" on the hardware and collectively provide the virtual machine capability that is necessary to support layer 4, the applications software. In the following paragraphs, layers 1 thru 3 are described along with short descriptions of the tools that reside in these layers and the relationship of such tools to the various development activities.

Layer 3: Functional Support Tools

Layer 3 contains those tools that provide direct support to the software development activities. These are the tools with which the applications software developer has the greatest interaction. Layer 3 tools will be related to the specific development activities they support.

Layer 3 Tool Types That Support Activity 1 (Analyze Requirements)

The types of tools that are directly applicable to requirement analysis are listed below:

(1) General Purpose System Simulators—Allow a user to construct a computer model of a real or proposed system and to perform simulation experiments to determine the behavior of the model under various operational conditions.

(2) System Description Languages & Analyzers—Assist system analysts in describing the functional characteristics of a system and in validating the consistency and completeness of a functional decomposition.

Layer 3 Tool Types That Support Activity 2 (Design Software)

The types of tools that are directly applicable to software design are listed below:

(1) Computer System Simulators—Similar in nature to the general purpose simulator except that their basic building blocks represent real computer system components whose modeled behavior approximates the throughputs, capacities, and access times achievable on the modeled equipments.

(2) Data Base Design Aids—Assist data base designers in grouping data elements into logical record classes and in determining the relationships among logical record classes implicit in either the nature of the data or the usage of the data.

(3) Data Dictionary Systems—Assist data base designers in managing the data definition activities.

Layer 3 Tool Types That Support Activity 3 (Build System Tests)

The types of tools required to support system test construction are listed below:

(1) Test Data Generators—Create data files for testing and validating programs.
(2) Test Data Auditors—Compare data files against specification and produce reports of discrepancies and/or compliance.
(3) Test Case Design Advisors—Analyze programs written in a high level language and present the results of that analysis in a form suitable to assist test case designers in the selection of test data.
(4) Test Instruments and Analyzers—Instrument modules under test so as to collect data characterizing the behavior of the module.

Layer 3 Tool Types That Support Activity 4 (Build and Unit-Test Software)

The types of tools that are required to support the program development and unit-test activity are listed below:

(1) Assemblers—Allow programs to be coded in a symbolic language in which statements generally correspond to a single machine instruction. Specific tools include Basic Assemblers and Macro Assemblers.
(2) Compilers—Translate programs written in a high level language into either relocatable object code acceptable to a Linker or assembly language acceptable to an Assembler.
(3) Linkers—Combine the text produced by separate invocations of Compilers and Assemblers ("object modules") into executable code strings ("load modules" or "core images") that can be loaded into the computer's main storage and executed without further preprocessing. Specific tools are Basic Linkers, Simple Overlay Linkers, and Extended Overlay Linkers.
(4) Debugging Aids—Assist the programmer in locating the sources of program errors that have been discovered during unit testing, usually by giving him some control over the execution of the module under test that is external to the normal program code. Specific tools are Interactive Symbolic Debuggers, Non-Interactive Symbolic Debuggers, Interactive Absolute Debuggers, and Non-Interactive Absolute Debuggers.
(5) Module Libraries & Change Control Systems—Provide computer controlled maintenance of groups of related source modules (programs), object modules (the output of Assemblers and Compilers), and load modules (the output of Linkers). Specific tools are Basic Libraries, Integrated Libraries, and Automatic Software Production & Test Systems.
(6) Performance Monitors—Assist the programmer in quantifying the resource consumption characteristics of a program and in isolating performance-critical areas. Specific tools are Language Dependent Monitors and Language Independent Monitors.
(7) Standards Enforcers—Allow source programs to be examined automatically and checked for conformance to installation-defined standards of format, content, and usage.
(8) Preprocessors and Reformatters—Assist programmers in producing well-structured and readable programs by allowing the introduction of structured programming constructs into source programs for languages that do not have them, and by automatically controlling indentation, the placement of comments, etc., to produce readable listings.

Layer 3 Tool Types That Support Activity 5 (Integrate and System Test) and Activity 6 (Maintain System)

There are no unique layer 3 tools that exist to support these activities. The tools that were listed for activities 1 through 4 are generally applicable to activities 5 and 6 at layer 3. Most of the tools used in practice that are specifically oriented to activity 5 are special-purpose, e.g., test environment tools (emulators, hot benches, system integration lab support, virtual machines), test drivers and special performance monitors.

Layer 2: General Support Services

The primary function of layer 2 tools is to provide a framework of common services that will allow the output of third layer functions to be stored, retrieved and intercommunicated. Second layer functions should be usable for common purposes across different third layer functions, and should serve to hide (where possible) differences between first layer and third layer functions. Layer 2 tool types provide general support to all of the software development activities. These tool types are summarized as follows:

(1) Data Base Management Systems—Allow the use of a computer system to define the contents of and the logical relationships between collections of data items that represent some useful abstraction of a real-world phenomenon (tactical command and control system, the modules and documentation of a system of computer programs) without being concerned with the physical mechanics of storing, locating, and retrieving items or groups of items.
(2) PERT/CPM Systems—Assist managers in planning and controlling project activities.
(3) Project Estimation Systems—Assist in the development of work breakdown structures and related performance standards for use in estimating project resource requirements.
(4) Documentation Aids—Assist in the preparation and

maintenance of documentation about the modules of a system. Specific tools are Text Processing Systems, Flowchart Construction Languages and Automatic Flowcharters.

(5) Data Manipulation Utilities—Allow the system user to alter the format and content of data files independent of the logical significance of the data fields involved. Specific tools are Sort/Merge Programs and Editors (Interactive Source Language Editors, Interactive Object Module Editors, Batch Source Language Editors, and Batch Object Module Editors.

(6) Information Retrieval Systems—General purpose application programs operating either on-line (interactively) or in batch that interpret user requests to locate and display information that is stored either within a structured data base or within separate files. Specific tools are Query Language Systems and Report Writers.

Layer 1: Operating System Services

Layer 1 implements the operating system services that present a "virtual machine" interface to the services/tools at layers 2 and 3 and manage the real system hardware. The layer 1 tool types are generally applicable across all of the software development activities. Layer 1 tool types/capabilities are listed below:

(1) Basic Operating Systems (BOS)—Run single user processes from initiation to termination. May or may not overlap I/O with execution. Provide basic I/O support that allows user to refer to files symbolically and to read and write them without knowing the hardware details of the I/O Interface. Provide basic batch supervisor services that control normal and abnormal job termination, job to job transition, and operator communication. Provide a minimum base for program development by supporting at least one language translator and/or linker/loader.

(2) Multiprogramming Operating Systems (MOS)—Provide all of the services of the Basic Operating System. Supports the concurrent execution of two or more user jobs by allowing the execution of any job to be suspended while another is executed without any special programming considerations in the user job. Prevents concurrently executing user jobs from accidentally or intentionally destroying each other or the supervisor.

(3) Multiprocessor Operating Systems (MPOS)—Allow the computing load to be spread across more than one processor based on automatic (programmed) load-leveling algorithms or operator control, but does not require special case programming in the user job. Multiprocessor Operating Systems include the shared storage, loosely coupled, and networked types.

(4) Virtual Machine Monitor (VMM)—The operating system presents an interface to the user program that makes it appear that the program is executing on a real computing system.

(5) Time-Sharing Operating System (TSOS)—This is a variant of the multiprogramming operating system in which system resources are allocated to user jobs in such a way that all jobs appear to progress at the same rate. In addition, users are allowed to "interact" with and receive output from their jobs via terminals. Such systems are optimized for response rather than throughput.

(6) Real-Time Operating Systems (RTOS)—Allow user jobs to be executed within specified short time limits.

EVALUATION OF THE SOFTWARE BASES

One of the fundamental motivations of the entire MCF program is "software capture," i.e., the ability to take advantage of the previous investment in support software associated with the winning architecture. In order to compare the three finalist architectures, a procedure was required which could quantify their support software investment. The items deemed to be identifiable, obtainable and translatable into financial data are applicability, availability from the architecture manufacturer, and availability from sources other than the architecture manufacturer.

Applicability

Applicability involved the determination of the functional relevance of a given software base component (tool type) to the development of military tactical software systems. Applicability was not intended to be a binary criterion but was to be a measure of the potential importance of the component, ranging from "not applicable" to "essential." Factors that were to be considered in determining the importance of a tool type, in addition to essentiality, included spectrum coverage, economic impact, software size, and the number of different instances of the same tool type for a given CFA. Committee members were sent a list of support software tools delineated in the previous section, were allotted 5,000 points, and were asked to distribute these points over the tools thereby indicating the relative applicability/importance of each tool to their activities. The guidance given to the members was that if a tool was essential for military computer software development then it should deserve more weight than one that is only nice to have. Consideration was also to be given to spectrum coverage, i.e., the utility of the component across development activities as well as across development disciplines. Consideration was to be given to economic impact, the potential cost savings that may be realized through use of the tool type.

The results of this applicability balloting were compiled and the tool list was ordered in terms of points received. A predetermined threshold of 1000 points was then applied against the list. In other words, any tool which received

TABLE I—Composite of Software Availability for the IBM, DEC and Interdata Architectures

	IBM	DEC	INTERDATA
1. LAYER 3			
1.1 Requirements Analysis			
1.1.1 General Purpose Systems Simulator	A	N	N
1.2 Software Design			
1.2.1 Computer System Simulator	O	N	N
1.2.2 Data Base Design Aid	A	A	N
1.3 Construct System Tests			
1.3.1 Test Data Generator	O	N	N
1.3.2 Test Data Auditor	O	N	N
1.3.3 Test Case Design Advisors			
1.3.3.1 FORTRAN	N	N	N
1.3.3.2 COBOL	N	N	N
1.3.3.3 CMS-2	N	N	N
1.3.3.4 JOVIAL	N	N	N
1.3.3.5 TACPOL	N	N	N
1.3.3 Test Instruments & Analyzers			
1.3.4.1 FORTRAN	A	O	O
1.3.4.2 COBOL	A	N	N
1.3.4.3 CMS-2	N	N	N
1.3.4.4 JOVIAL	N	N	N
1.3.4.5 TACPOL	N	N	N
1.4 Build & Unit Test			
1.4.1 Assemblers			
1.4.1.1 Basic Assembler	A	A	A
1.4.1.2 Macro Assembler	A	A	A
1.4.2 Compilers			
1.4.2.1 FORTRAN	A	A	A
1.4.2.2 COBOL	A	A	A
1.4.2.3 CMS-2	N	N	N
1.4.2.4 JOVIAL	A	N	N
1.4.2.5 TACPOL	N	N	N

A = Available from architecture manufacturer.
O = Available from a source other than architecture manufacturer.
N = Not available from either source.

TABLE I—(Continued)

	IBM	DEC	INTERDATA
1.4.3 Linkers			
1.4.3.1 Basic Linker	A	A	A
1.4.3.2 Simple Overlay Linker	A	A	A
1.4.3.3 Extended Overlay Linker	A	A	N
1.4.4 Debugging Aids			
1.4.4.1 Interactive Symbolic Debugger			
1.4.4.1.1 Assembly	O	N	N
1.4.4.1.2 FORTRAN	A	N	N
1.4.4.1.3 COBOL	A	O	N
1.4.4.1.4 CMS-2	N	N	N
1.4.4.1.5 JOVIAL	N	N	N
1.4.4.1.6 TACPOL	N	N	N
1.4.4.2 Non-interactive Symbolic Debugger			
1.4.4.2.1 Assembler	N	N	N
1.4.4.2.2 FORTRAN	A	A	N
1.4.4.2.3 COBOL	A	A	N
1.4.4.2.4 CMS-2	N	N	N
1.4.4.2.5 JOVIAL	N	N	N
1.4.4.2.6 TACPOL	N	N	N
1.4.5 Module Libraries & Change Control Systems			
1.4.5.1 Integrated Library	O	N	N
1.4.5.2 Automatic Software Production & Test	N	N	N
1.4.6 Performance Monitors			
1.4.6.1 Language Dependent Monitors			
1.4.6.1.1 Assembly	O	N	N
1.4.6.1.2 FORTRAN	A	N	N
1.4.6.1.3 COBOL	A	N	N
1.4.6.1.4 CMS-2	N	N	N
1.4.6.1.5 JOVIAL	N	N	N
1.4.6.1.6 TACPOL	N	N	N
1.4.7 Standards Enforcers			
1.4.7.1 FORTRAN	N	N	N
1.4.7.2 COBOL	N	N	N
1.4.7.3 CMS-2	N	N	N
1.4.7.4 JOVIAL	N	N	N
1.4.7.5 TACPOL	N	N	N

less than that threshold would no longer be considered. The justification for this was that DoD could not affort to build tools which a representative spectrum of system developers determined were not very applicable.

The applicability results eliminated approximately half of the tools delineated in the previous section leaving 28 tools to be utilized in the availability phase of the evaluation. These tools are delineated in Table I.

TABLE I—(Continued)

	IBM	DEC	INTERDATA
1.4.8 Pre-processors/Reformatter			
1.4.8.1 Reformatter			
1.4.8.1.1 FORTRAN	0	0	0
1.4.8.1.2 COBOL	0	N	N
1.4.8.1.3 CMS-2	N	N	N
1.4.8.1.4 JOVIAL	N	N	N
1.4.8.1.5 TACPOL	N	N	N
2. LAYER 2			
2.1 Data Base Management System	A	A	N
2.2 Documentation Aids			
2.2.1 Text Processing System	A	A	N
2.3 Data Manipulation Utilities			
2.3.1 Editors			
2.3.1.1 Interactive Source Language Editors	A	A	A
2.3.1.2 Batch Source Language Editors	A	A	A
3. LAYER 1			
3.1 RTOS + TSOS	A	A	A
3.2 TSOS + VMM	A	N	N
3.3 TSOS + MPOS + VMM	A	N	N

Availability from architecture manufacturers

This criterion involved determining whether the support software tools delineated in Table I were available from the finalist architecture manufacturers. A tool was considered available if it was presently being marketed and maintained. This definition permitted the criterion to be applied uniformly and equitably to all the finalist manufacturers. It was felt that, if tools under development were also considered, it would have been impossible to determine whether a tool was one month, one year, five years away, etc., from being marketed by the company.

The actual determination of availability of support software tools was to be conducted in two phases. Phase I consisted of providing each of the manufacturers with a list of the support software tools (Table I) as well as a description of the minimum essential characteristics of each tool. Each manufacturer was requested to answer in the affirmative for all tools which they actively marketed and maintained. Phase II consisted of a visit to each of the manufacturers by government personnel and an independent auditor to obtain supporting documentation and to audit the manufacturer responses. Table I depicts the results of this process. All tools which were available from a particular architecture manufacturer are marked by an "A." All tools which were not available are marked by an "N" or an "O."

Availability from other vendors

It was felt that there existed a great deal of support software available from sources other than the architecture manufacturers which met the minimum essential characteristics. It was the committee's feeling that such software should be included. Again, firm criteria were needed—the tool had to meet certain minimum essential characteristics, had to be hosted on and targeted for one of the finalist architectures, and had to be marketed and maintained. The International Computer Programs Inc. (ICP) "INTERFACE Reference Series"[5] was chosen as the source document because of its position as a de facto standard for software marketing. However, only tools which are not marketed by the manufacturer were searched for. Upon finding such a tool in the ICP document, the vendor was contacted to obtain further supporting data. Table I also depicts the results of this process. Tools available from other than the architecture manufacturer are annotated by an "O."

Consolidation of results

The main purpose of the software evaluation was to determine the relative current software dollar investment as well as the current software dollar deficiency of the three architectures. In order to do this, a development cost for each tool was needed. It was known that the architecture manufacturers and software vendors considered such information to be proprietary. Therefore, the following approach was taken: First each manufacturer was requested to provide the source code size (disregarding comments) for his available tools as well as the language which the tool was written in, and the object code size in instructions. Second, a productivity figure was needed. F. Brooks', "The Mythical Man-month"[6] was considered to be the best source since it compiled productivity figures from IBM's OS/360 development as well as Bell Lab's ESS software development. Brooks cites 600 lines of code per man-year for operating system development and 2,000 lines of code per man-year for other software development. It was felt that the state-of-the-art in operating systems had improved significantly since his data was obtained (nearly ten years ago) and thus a figure of 1,000 and 2,000 lines of code per man-year for operating system and other support software, respectively, was decided upon. Third, a fully loaded price per man-year of $70,000 was assumed. Table II depicts the resultant relative software bases and deficiences of the three architecture finalists.

TABLE II—Software Base and Deficiency Comparison

	BASE	DEFICIENCY
IBM	32,269K	9,595K
DEC	22,220K	19,130K
INTERDATA	15,360K	25,970K

A second desired produce of the software evaluation was a schedule for each architecture which depicted the development sequence for eliminating software deficiencies. To accomplish this, a list of deficiencies was generated from Table I for each architecture and sorted in applicability order. Next each list was slightly reordered in terms of a reasonable PERT sequence. In other words if tools X, Y, and Z were rated by the committee members in terms of applicability in the order Z, Y, X, but, the development of Y before Z would incur a saving in overall development costs, then the final development sequence would be ordered as Y, Z, X.

Assuming reasonable development periods for each tool, assuming a figure of 2 million dollars a year as an estimate of the support software R&D dollars available when the MCF program is implemented, and utilizing the deficiency lists in development order, a development schedule was generated for each architecture which provides, at a glance, the relative future deficiencies of each architecture. Schedules were also constructed based upon annual support software R&D expenditures of 1 million and 3 million dollars. Table III depicts the period of time to completely eliminate software deficiencies for each architecture based upon the various support software R&D dollar estimates.

TABLE III—Years to Correct Deficiencies

	DEVELOPMENT DOLLARS		
	1M	2M	3M
IBM	10.5	5.5	4.5
DEC	20	11	8.5
INTERDATA	26	15	10

REFERENCES

1. Burr, W., A. Coleman, and W. Smith, "Overview of the Military Computer Family Architecture Selection," National Computer Conference, *AFIPS Conference Proceedings,* Vol. 46, 1977.
2. "BMDATC Software Development System—Research Description," US Army BMD Advanced Technology Center Report, July 1975.
3. Bratman, H., and T. Court, "The Software Factory," *IEEE Computer,* May 1975.
4. Hamilton, N., and S. Zeldin, "Integrated Software Development System/Higher Order Software Conceptual Description," Technical Report for Contract ECOM 76-0329-F dated Nov. 1976, Charles Stark Draper Laboratory and Higher Order Software Inc.
5. "INTERFACE Reference Series," International Computer Programs, Inc., Carmel, Indiana, 1976.
6. Brooks, F. P., *The Mythical Man-month: Essays on Software Engineering,* Addison-Wesley, Reading, Mass., 1976.

Life cycle cost models for comparing computer family architectures

by JOHN J. CORNYN, WILLIAM R. SMITH
Naval Research Laboratory
Washington, DC

and

AARON H. COLEMAN
U.S. Army Electronics Command
Ft. Monmouth, New Jersey

and

WILLIAM R. SVIRSKY
Systems Development Corporation
W. Long Branch, New Jersey

ABSTRACT

This paper describes the methodology used to compute life-cycle costs of military computer systems as a function of three competing Military Computer Family Architecture candidates (the IBM S/370, DEC PDP-11, and Interdata 8/32), and it presents separate results of applying this methodology to two different models (called Top-down and Bottom-up) of computer resource requirements in the military. The architecture comparisons are made by projecting, computing and combining estimates of hardware and software costs with architecture-dependent factors to obtain life-cycle (development plus maintenance) costs and cost ratios. The results indicate that the three architectures have roughly comparable life cycle costs for the two models. However, neglecting the uncertainties of the input data and assumptions, the bottom-up results indicate that in most circumstances the DEC PDP-11 is superior to both the IBM S/370 and Interdata 8/32 architectures. The top-down model results, on the other hand, indicate that the S/370 is superior for high (greater than one) software-to-hardware cost ratios, the Interdata 8/32 is slightly better for low (less than one-fourth) ratios, and the PDP-11 is best in between.

INTRODUCTION

Background

The motivation for this work was to help the Army and Navy determine the most cost effective of three candidate computer architectures, with the intent that this architecture would form the basis of a software-compatible Military Computer Family (MCF) over at least the next decade.[1-3] The three candidate architectures were the IBM S/370, the Digital Equipment Corporation PDP-11, and the Interdata 8/32. It was felt that inclusion of cost considerations would be a vital element in the acceptance of the Selection Committee's recommendations by DoD higher management.[4]

Overview

The life-cycle cost evaluations described are based on (1) a methodology for combining the results of architectural attribute evaluations (i.e., processor and storage efficiency, support-software availability) with computer resource requirements (i.e., bits of storage, processor speed, program sizes) to compute dollar costs of these resources as a function of architecture, and (2) models defining these computer resource requirements considered to be representative of military tactical and strategic systems in the next 10 to 15 years.

Architecture Attributes

The CFA Selection Committee generated the architectural attribute evaluation results that are the basic inputs to the cost computations. The first of these are the data derived from test program experiments. These data indicate the relative efficiencies of the architectures in utilizing storage and processor hardware resources. The "S" measure is a count of the number of storage bytes required to contain programs, given an architecture. Differences in S can be directly related to differences in the amount of storage and therefore cost requirements. The "M" measure is a count of the number of bytes transferred between the CPU and main memory (including cache) during execution of the test programs, and the "R" measure is a count of the number of bytes transferred among the registers of the CPU. M and R are clearly indicators of the hardware bandwidth requirements of an architecture to do a job. Everything else being equal, memory cost will be greater if more storage is required (larger S) or if the memory has to be faster (larger M, R). Similarly, the CPU cost will be larger if the processor has to be faster (larger M, R). The relationship between memory CPU speed and cost is taken as Speed (in MIPS)=k×cost^g, where k and g are empirically derived constants.

The other architecture attribute that is used in the cost computations is the availability of software tools to aid in developing software for MCF systems. This was established by the Selection Committee by defining a menu of support software (i.e., compilers, editors, etc.) required for military applications and then evaluating the relative percentage of this menu available for each architecture. Using data from actual system developments, a curve was generated relating this relative availability of software tools to the cost (per line of code) of developing software for an architecture. This data was ultimately used in the computation of life cycle software costs. Detailed description of the cost analyses described herein can be found in Reference 5.

The Models

Two models of military computer life cycle resource requirements are used in conjunction with the basic methodology described above. These are called the Top Down (TD) and Bottom Up (BU) models. It is frequently useful when building models for forecasting economic data to apply two different approaches in order to cross check the results.

The TD model predicts computer resource requirements by extrapolating trends in overall expenditures and requirements in DoD for various aspects of computer hardware and software. This model predicts, year-by-year, up to 1990 what these expenditures will be for each architecture and then provides relative architecture costs based on the cumulative costs.

The BU model predicts computer resource requirements by using the hardware and software design characteristics of fifteen actual military data processing systems in development or to be developed. The costs of developing all these systems are computed, given each candidate architecture, for the years 1976 and 1985.

The reader will notice that the notations in the Bottom-Up and Top-Down sections of this paper do not always correspond. The two models and analyses were generated and authored independently,[6,7] except for some initial exchange of ideas and discussion of input data. We have not attempted to normalize the descriptions of the two models.

BOTTOM-UP MODEL

Basic approach

Overview

The computer resource life-cycle cost was estimated for each of 15 Army embedded-computer systems (shown in Table II), which are currently in various phases (conceptual through deployment) of their life cycles. Estimates were made for 1976 and for 1985 production procurements. The lowest cost CFA for each system and for all systems was selected for 1976 and 1985 procurements.

Total Life Cycle Cost

The computer resource life cycle cost for system i and architecture j is defined as:

$$C_{ij}=Hw_{ij}+ASw_{ij} \tag{1}$$

where

HW_{ij}=hardware life cycle cost
ASw_{ij}=applications software life cycle cost

Hardware Life Cycle Cost (LCC)

The computer hardware life-cycle cost for a given system using a specific CFA is defined as

$$HW_{ij}=n_iL_h(P_{ij}+MM_{ij}+SM_{ij}) \tag{2}$$

where

i=index of the system
j=index of the architecture
n_i=number of units to be produced for system i
L_h=hardware life-cycle cost factor, i.e., ratio of total hardware life-cycle cost to hardware acquisition cost. This factor is assumed to be 2 for a 10-year life cycle
P_{ij}=processor acquisition cost
MM_{ij}=main memory acquisition cost
SM_{ij}=secondary memory acquisition cost

The processor acquisition cost for system i using architecture j is defined as

$$P_{ij}=K(a_{ij}Mr_i)^{0.4} \tag{3}$$

where

K=constant relating to processor cost
a_{ij}=processor speed ratio
Mr_i=operating speed in millions of instructions per second (MIPS)

Equation (3) follows from a commonly cited relationship between performance and cost, namely performance=constant×cost^g. For the purpose of the BU model g was assumed to be 2.5. To obtain a value for K, in Eq. (3), we used the fact that recent cost/speed data for several military processors seems to indicate that speeds of 0.5 MIPS and processor costs of $48,000 are representative values; consequently

$$K=48\times 10^3/(.5)^{.4}=6.3\times 10^4.$$

This value of K is used in subsequent calculations for 1976 processor cost estimates and is reduced by a factor of 10 for 1985 processor cost estimates based on an assessment of hardware cost reduction over the next decade.

The main memory acquisition cost for system i using architecture j is defined as:

$$MM_{ij}=c_b(b_{ij}PM_i+DM_i) \quad (4)$$

where:

MM_{ij}=main memory acquisition cost (in dollars)

b_{ij}=static storage ratio

PM_i=main memory (in bits) required for program storage in system i; M_i is derived from system proponents; P is estimated fraction of M_i dedicated to program storage vs data storage. (See Table I for values.)

DM_i=main memory (in bits) required for data storage in system i; M_i is derived from system proponents; D is estimated fraction of M_i dedicated to data storage vs. program storage

c_b=cost per bit of main memory derived from the study of Air Force ADP requirements through the 1980's[8] and Turn's data in *Computers in the 1980's*.[9] Examination of the price per bit of recent militarized memory systems indicates an average cost of 4 cents per bit; i.e., $5000 per 16K byte memory module. This value is used in 1976 cost estimates; 0.4 cents is assumed in 1985 cost estimates.

The secondary memory acquisition cost for system i using architecture j is defined as

$$SM_{ij}=C_a(b_{ij}P'Ma_i+D'Ma_i) \quad (5)$$

where

SM_{ij}=secondary memory acquisition cost (in dollars)

b_{ij}=static storage ratio

$P'Ma_i$=secondary memory (in bits) required for program storage in system i; Ma_i is derived from system proponents while P' is the estimated fraction of Ma_i used for program storage vs. data storage

$D'Ma_i$=secondary memory (in bits) required for data storage in system i; Ma_i is derived from system proponents while D' is the estimated fraction of secondary memory used for data storage vs. program storage

C_a=cost per bit of secondary memory derived from References 8 and 9. Examination of the price per bit of current militarized disc systems indicates an average cost of 0.2 cents per bit, e.g., a 36M bit disc system at $72,000. This value is used in 1976 cost estimates; a cost reduction of 10:1 in the next 10 years is assumed so that a price of 0.02 cents per bit is used in 1985 cost estimates.

To obtain values of n_i, Mr_i, M_i, Ma_i, S_i, a letter was sent from ECOM to project managers of the fifteen systems requesting values for their particular system. Their responses are tabulated in Table I.

The processor speed ratio (a_{ij}) and static storage ratio (b_{ij}) attempt to capture the ability of the j-th architecture relative to system i. They are derived by measuring the performance of the three architectures on twelve test programs and by estimating the relative importance, or weight, of each of these programs to computations characteristic of each system. The performance of each architecture on the test programs is summarized in what are called the S, M, and R measures, where:

S_{kj} is a measure of the amount of memory (in 8-bit bytes) needed to represent test program k on architecture j.

M_{kj} is a measure of the processor/memory transfers required to execute test program k when using architecture j.

R_{kj} is a measure (in 8-bit bytes) of the number of internal register-to-register transfers required by the processor to execute test program k on architecture j.

See Reference 10 for details of the test program experiment.

The relevance of the k-th test program to the i-th system is given by the factors W_{ik}, which were obtained by first dividing the twelve programs into two categories: programs that relate principally to I/O, and those that are associated with traditional processor/memory functions. Within these categories, varying degrees of functional overlap occur among the test programs. Initially a gross value was estimated for each of the two categories; subsequently, this value was distributed across the programs of the category. As an example of how the weights were distributed, one data management system resembles contemporary commercial data processing in its use of COBOL and data bases. Thus, its weighting was heavily biased toward I/O and processing routines such as "Linked List Insertion," and lightly biased toward mathematical oriented routines such as "Runge Kutta." Another system, however, is designed primarily for trajectory computations and thus its weighting favored the "Runge Kutta" and was biased against "Linked List Insertion." The sum of all the test programs weights for a given system is unity.

The processor speed ratio (a_{ij}) and the static storage ratio (b_{ij}) were obtained by combining the above quantities in the

TABLE I—System Proponent Data

Sys. #	System Mission	n_i	Mr_j (MIPS)	*PM_i	*DM_i	*PMa_i	*DMa_i	S_i**
1	Medium Search	192	1.33	.414	.101	1.9	7.7	32
2	Medium Command & Control	27	.26	1.638	.412	2.2	8.8	144
3	Small Search	100	1.00	.512	.128	2.8	11.2	20
4	Large Command & Control	178	.20	13.600	3.400	4.0	16.0	375
5	Medium Command & Control	64	.50	1.600	.400	15.8	63.2	47
6	Large Command & Control	30	.40	3.321	.820	8.4	33.6	250
7	Small Command & Control	832	.75	.618	.152	.4	1.6	100
8	Large Communications	616	.18	3.200	.800	13.4	52.0	175
9	Small Communications	800	.16	.116	.031	0	0	8
10	Small Communications	9	.53	.408	.102	3.2	12.8	83
11	Small Special Purpose	30	.48	.328	.082	.1	.3	14
12	Large Data Management	16	.02	1.600	.400	573.4	2293.6	324
13	Medium Search	50	.35	3.712	.928	3.2	12.8	28
14	Medium Data Management	8	.80	3.200	.800	1912.0	7648.0	1
15	Small Guidance & Control	3325	.20	.006	.002	0	0	1

* P and D are fractions applied to the proponent's stated memory requirements which reflect an estimate of memory used for programs (P) vs. data (D). Values are expressed in megabits.

** S_i stated in 10^3 instructions.

following manner:

$$a_{ij}=\frac{3\sum_{k=1}^{12} W_{ik}(3M_{kj}+R_{kj})}{\sum_{m=1}^{3}\sum_{k=1}^{12} W_{ik}(3M_{km}+R_{km})} \qquad (6)$$

$$b_{ij}=\frac{3\sum_{k=1}^{12} W_{ik}S_{kj}}{\sum_{m=1}^{3}\sum_{k=1}^{12} W_{ik}S_{km}} \qquad (7)$$

Applications Software Life Cycle Cost

The applications software life cycle cost for system i using architecture j is defined as:

$$\bar{A}Sw_{ij}=Cs_jS_iL_s \qquad (8)$$

where

Cs_j=cost (in dollars) per instruction of applications software for architecture j

S_i=applications software size (in instructions), derived from the system proponents data (Table I).

L_s=applications software life cycle cost factor, i.e., ratio of applications software life-cycle cost to initial acquisition cost

The software life-cycle cost factor was taken basically from Fisher's report[11,p.64] which places modifications and retrofits to software at four to five times the cost of the initial product. Thus by taking the midpoint and adding the initial cost as one, we have a value of L_s=5.5. The parameter, Cs_j, the cost per instruction of application software for architecture j, is based on the experience of System Development Corporation with five large scale (24K instructions to 500K instructions) software efforts. The data was compiled by sending questionnaires to the program managers. Program managers responded with: the cost of software production, the number of instructions produced, and for thirteen software tools they estimated what would be the percent increase in project cost if the tool were not available and how much less the project cost would be if the ideal tool were available. From generalizations of this data, it was possible to construct the curves shown in Figure 1. These curves show the variation of cost per instruction as a function of the Tool Availability Index (TAI) for three conditions: (a) worst case, (b) best case and (c) derived median. It should be recognized that the results are largely judgmental and that examples can also be found which yield costs per instruction above and below the worst and best case curves of Figure 1. The Tool Availability Index (TAI) is defined as the ratio of available software tools to the ideal set of software tools. The "ideal set" is defined in a separate report by the Software Evaluation Methodology Subcommittee.[12]

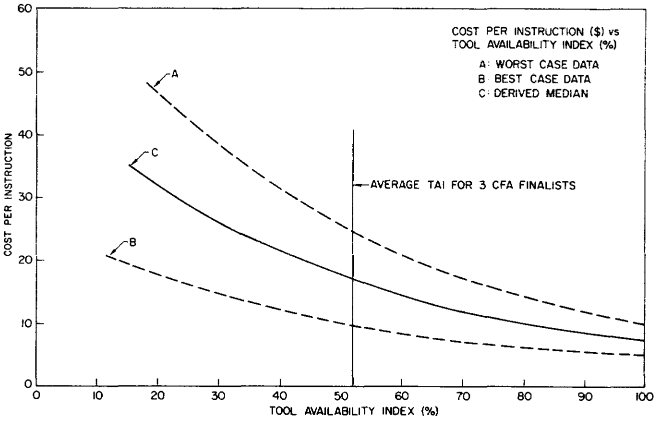


Figure 1—Availability index (%)

By knowing the TAI for a given architecture for any point in time, one can obtain from Figure 1 an estimated cost per instruction. As the percentage of available software tools increases, the cost per instruction of application software can be seen to diminish. The 1976 TAI for each CFA finalist was derived from the report of the Software Evaluation Committee[12] as 34%. 50% and 73% for the Interdata 8/32, DEC PDP-11 and IBM S/370 architectures, respectively.* The average of these values is 52% and is indicated graphically in Figure 1. The cost per instruction for the average TAI is approximately \$25, \$10 and \$17 for conditions A, B and C, respectively. The cost per instruction for the 3 CFA finalists in 1976 was estimated at \$24, \$18, and \$12 for Interdata 8/32, DEC PDP-11 and IBM S/370, based upon TAI values of 34%, 50% and 73%, respectively. The corresponding values for 1985 were computed by assuming an annual expenditure of \$2M to augment the support software base of the selected CFA. The resulting TAI values in 1985 are 83%, 100% and 100% corresponding to cost per instruction of \$10, \$7.50 and \$7.50 for Interdata 8/32, DEC PDP-11 and IBM S/370, respectively. This data was then employed to compute applications software life cycle cost for each of the 15 systems under consideration, for each CFA and for 1976 and 1985.

Results

Total Life Cycle Costs

Table II illustrates the total life cycle costs for each system and for each of the three architectures for 1985. Circled values indicate the least-cost architecture for a particular system and the least total cost for implementing all fifteen systems. The number of times an architecture is selected is also totaled and shown at the bottom of the chart as the Number of System Preferences. Table II indicates that:

The system preferences for DEC, IBM and Interdata

* A subsequent refinement of the data in the SEC report changes these percentages to 37%, 54% and 77%. The changes proved to have no significant impact on the data described herein.

TABLE II.—Total Life Cycle Cost vs CFA
1985 Procurement

1976 AVERAGE APPL'NSW COST=$17/INSTRUCTION											
SYS #	SYSTEM MISSION	n_i	INTERDATA 8/32			DEC PDP-11			IBM $/370		
			HDW	ASW	TOTAL	HDW	ASW	TOTAL	HDW	ASW	TOTAL
1	Medium Search	192	3.9	1.8	5.7	4.1	1.3	5.4	4.7	1.3	6.0
2	Medium Comm'd and Cntrl	27	0.7	7.9	8.6	0.7	5.9	6.6	0.9	5.9	6.8
3	Small Search	100	2.2	1.1	3.3	2.1	0.8	2.9	2.6	0.8	3.4
4	Large Comm'd and Cntrl	178	23.0	20.6	43.6	23.0	15.5	38.5	34.5	15.5	50.0
5	Medium Comm'd and Cntrl	64	3.4	2.6	6.0	3.4	1.9	5.3	4.1	1.9	6.0
6	Large Comm'd and Cntrl	30	1.6	13.8	15.4	1.6	10.3	11.9	2.1	10.3	12.4
7	Small Comm'd and Cntrl	832	13.4	5.5	18.9	13.4	4.1	17.5	18.3	4.1	22.4
8	Large Communications	616	36.3	9.6	45.9	35.4	7.3	42.7	47.3	7.2	54.5
9	Small Communications	800	5.5	0.4	5.9	5.3	0.3	5.6	6.4	0.3	6.7
10	Small Communications	9	0.2	4.6	4.8	0.2	3.4	3.6	0.2	3.4	3.6
11	Small Special Purpose	30	0.5	0.8	1.3	12.5	0.6	1.1	0.6	0.6	1.2
12	Large Data Management	16	18.0	17.8	35.8	18.1	13.4	31.5	19.9	13.4	33.3
13	Medium Search	50	2.4	1.5	3.9	2.4	1.2	3.6	3.0	1.2	4.2
14	Medium Data Management	8	29.9	1.9	31.8	29.6	1.4	31.0	33.3	1.4	34.7
15	Small Guidance & Cntrl	3325	203	0.1	20.4	19.9	0.1	20.0	25.6	0.1	25.7
	Total Cost		161	90	251	160	68	228	204	68	272
	# System Preferences							14.5			0.5

would be 14.5, 0.5 and none, respectively. The average total life cycle cost for all systems/architectures is $250M in 1985. The 8/32 cost is approximately equal to the average cost while the PDP-11 and S/370 costs are 8.9% below and 8.7% above, respectively, the average cost.

Sensitivity Analysis

Some of the parametric values leading to the 1976 and 1985 results were perturbed in order to determine the sensitivity of the life cycle cost model. The results are described in the following subparagraphs.

When the average cost per instruction in 1976 was doubled to $34, or $9 in excess of the worst-case curve in Figure 1, the resulting hardware: software ratios were 6:1 and 1.2:1 in 1976 and 1985 respectively. The resulting life cycle costs for 1976 and 1985 indicated that, in terms of the comparative costs of the three architectures, the model is relatively insensitive to doubling the cost per applications software instruction. Comparing results, IBM picked up one system implementation at both DEC and Interdata expense in 1976 so that DEC, IBM and Interdata architectures would be preferred in 10.5, 4.0 and 0.5 systems, respectively. The DEC architecture remains the lowest in total life-cycle cost for all systems. For 1985 there were no changes, DEC architecture being preferred in 14.5 systems.

The data shown in Table II reflect an investment in the software bases of the three architectures of $2M annually.

The data for 1985 were re-examined to determine the impact of reducing the investment in the software base to $1M annually for two different situations: $17 per instruction of application software and then $34 per instruction. At $17 per instruction the selection of IBM would increase to 5 systems while DEC would drop to 10. This is also predictable since at a $1M per year investment rate in the software base IBM retains for a longer time its cost advantage in producing application software. At this lower investment rate, while IBM has achieved the ideal support software set, DEC has only 83% while Interdata has 61%.

Doubling the cost per application software instruction merely extends a software favorable situation, in that the number of IBM selections equals the number of DEC selections at seven.

The data was re-examined to determine the impact on the results when the S, M, and R measures were used without the discriminating weights applied as described earlier. The results showed that for 1985, the model is relatively insensitive to the weighting factors applied to the S, M, and R values. At $17 per application software instruction a few systems shift from DEC to Interdata; at $34 per instruction a few systems shift from DEC to IBM. DEC remains predominant.

Conclusions

The results obtained with the bottom-up life-cycle cost model are summarized in Table III.

Hence from the bottom-up model results we conclude that the DEC PDP-11 architecture would provide the lowest life-cycle cost for most of the 15 Army embedded-computer

TABLE III—Summary: Bottom Up Life Cycle Cost Analysis

A. AVERAGE TOTAL LIFE CYCLE COSTS (MILLIONS) FOR 15 SYSTEMS

Type Cost	1976	1985
Hardware	$1750	$175
Software	$ 162	$ 75
TOTAL	$1912	$250
HDW/SW Ratios*	12:1	2.4:1

B. 1976 ARCHITECTURE COMPARISON

	# System	Relative Total Cost**		
Architecture	Preferences	HDW	SW	Total
8/32	1	0.92	1.33	0.96
PDP-11	11	0.91	1.00	0.92
S/370	3	1.16	0.67	1.12

C. 1985 ARCHITECTURE COMPARISON

	# System	Relative Total Cost**		
Architecture	Preferences	HDW	SW	Total
8/32	-	0.92	1.20	1.00
PDP-11	14.5	0.91	0.91	0.91
S/370	0.5	1.16	0.91	1.09

* Hardware/software ratios are calculated from cost data

** 1.00 denotes average cost

systems considered and for the 15 systems as a whole in 1976 and 1985 under the following conditions:

(a) Applications software cost per instruction of $17 and $34 in 1976. Lower costs are assumed in 1985 as the support software base is augmented.
(b) Support software investment rate of $1M, $2M and $3M per year to 1985.
(c) Architecture test measures of effectiveness (S, M and R) are weighted for each system application.
(d) Hardware cost reduction of 10:1 from 1976 to 1985.
(e) Hardware life cycle cost is twice acquisition cost, software life cycle cost is 5.5 times acquisition cost.

The results shown in Table III are not significantly changed if the average applications software cost per instruction in 1976 is doubled to $34 (thereby decreasing the hardware:software ratio by a factor of 2) or if the annual support software investment for the selected CFA is increased to $3M or decreased to $1M.

THE TOP-DOWN MODEL

The Basic Model

The principal outputs of this model are values, $R_{jk}*$, $j=1, \ldots, N_y$, $k=1, \ldots, N_a$, which are the discounted costs of an architecture k, relative to a reference architecture, totaled over a period of j years. Here, N_y denotes the maximum time period in years and N_a the number of architectures under examination. An element $R_{jk}*$ is called a discounted cumulative cost ratio. If the model yields $R_{m1}*<R_{m2}*$ then, neglecting irreducibles, architecture 1 is more desirable than architecture 2 for the period j=1 through j=m, since it has a lower cumulative cost for that period.

The study described herein examines cumulative costs over periods of one to thirteen years ($N_y=13$) beginning in 1978. Cumulative costs were calculated up to 1990, j=13. Since this study compares only three architectures, we have $N_a=3$. The index k=1 represents the IBM S/370 computer family architecture (the reference architecture), k=2 the Digital Equipment Corporation PDP-11 architecture, and k=3 the Interdata 8/32 architecture.

The model obtains the yearly architecture-dependent cost (C_{jk}) by summing the architecture-dependent hardware costs (H_{jk}) and software costs (S_{jk}),

$$C_{jk}=S_{jk}+H_{jk}. \tag{9}$$

The nondiscounted cumulative cost through year m for architecture k, D_{mk}, is simply the sum of all costs from year 1 through year m,

$$D_{mk}=\sum_{j=1}^{m} C_{jk} \tag{10}$$

The discounted cumulative cost, $D_{mk}*$, on the other hand, takes into account the time value of money. Inflation aside, a dollar today is worth more than a dollar tomorrow. The model multiplies cash flows occurring in a year j by a discount factor, d_j, and sums the products,

$$D_{mk}*=\sum_{j=1}^{m} C_{jk}d_j. \tag{11}$$

Each discount factor is an average over the year j of the single-payment present-worth factor. These discount factors are the same as those recommended in References 13, 14 and used in Reference 8, volume V.

Since our immediate interest is only in the relative merits of the three architectures, their cumulative cost ratios provide an adequate and useful measure. In addition, the cost ratios are more useful than absolute costs because they serve to cancel out common and possibly unknown multiplicative factors.

Taking architecture one (the IBM S/370) as the reference architecture, we define the nondiscounted cumulative cost ratio as

$$R_{jk}=D_{jk}/D_{j1}, \tag{12}$$

and, analogously, the discounted cumulative cost ratio as

$$R_{jk}*=D_{jk}*/D_{j1}*. \tag{13}$$

Consequently, we have $R_{j1}*=R_{j1}=1$ for $j=1, \ldots, N_y$.

Architecture-Dependent Hardware Costs

The model obtains the architecture-dependent hardware costs (H_{jk}) by summing the architecture-dependent processor (P_{jk}'), main-memory (M_{jk}'), and secondary-memory (E_{jk}') expenditures,

$$H_{jk}=P_{jk}'+M_{jk}'+E_{jk}'. \tag{14}$$

The following paragraphs describe the method of arriving at values for these expenditures.

Total yearly hardware expenditures, B_j, $j=1, \ldots, N_y$, for CFA-related military computer systems are key inputs to the model. They include computer, main-memory, secondary-memory, I/O and peripheral devices, and related expenditures.

Military systems typically require years between initiation of development and full deployment. This phasing-in period is estimated to be approximately equal to average system development cycle time, which experience indicates runs between 3 and 10 years.[15-18,p.15] In order to approximate the expenditure levels during this phasing-in period, we assume that the expenditures B_j, $j=1, \ldots, N_y$, begin at some predetermined level, B_1, increase linearly with time over an initial development period (B_j, $j=1, \ldots, N_d$), and then remain constant for the remaining period (B_j, $j=N_d+1, \ldots, N_y$). For the purpose of the study, the development period, N_d, was estimated to be seven years. The model also includes effects of choosing development periods of five and ten years, assuming the same initial and final hardware expenditure levels. The simplifying assumption of constant base hardware expenditures after an initial development period appears to be reasonable because: (1) total ADP expenditures in DoD, when measured in uninflated dollars, have been nearly constant in recent years; (2) the principle of level funding has tended to guide DoD budget allocations; (3) decreasing hardware costs on a per-unit basis have tended to offset increasing (inflated) hardware requirements; and (4) the exact dollar assumption used is less important than its equal-handed application to each of the candidate architectures.

Although they are few and far between, there are some estimates and projections of DoD computer system budgets available today.[8,11,19] For our purpose, perhaps the most useful was D. A. Fisher's study of ADP costs in DoD.[11] In this report, Fisher estimated that in FY 1973, DoD spent 6.2 to 8.3 billion dollars on ADP. He found that approximately one third of this amount originated in each service, and that about 16 percent of the total went to computer hardware, 45 percent went to software, and 38 percent to other ADP costs, such as support and supplies, keypunching, and computer operation. Using these figures, we deduce that 1.0 to 1.3 billion dollars went to computer hardware and roughly 2.8 to 3.7 billion dollars went to software. During a private conversation with the authors in May, 1976,[20] he mentioned that his recent studies showed that if one were to divide DoD software costs by application, the major portion (approximately 56 percent) goes for embedded computer systems, i.e., the types of systems the CFA project is designed to influence. Approximately 19 percent goes for administrative data processing applications wherein COBOL is the principal language, and 5 percent goes to scientific applications using most commonly FORTRAN. The other 20 percent goes for other types of applications and indirect costs. Assuming these percentages also applied in FY 1973, this would mean that of the approximately 2.8 to 3.7 billion that went to software, about 1.6 to 2.1 billion went for software for embedded computer systems. If we make the assumption that the software-to-hardware cost ratio for embedded computer systems is approximately the same as that for overall DoD ADP systems, we obtain 0.6 to 0.7 billion dollars for hardware for embedded computer systems. Dividing this by three to obtain an estimate of cost for each service, we obtain 0.2 to 0.23 billion. Of course, not all embedded computer systems will be satisfied by CFA. Making a rough assumption that one fourth would be, we obtain a yearly annual hardware expenditure rate for a single service of about 50 million dollars. For the purpose of our study, we took this figure as our nominal hardware expenditure level. The exact expenditure level assumed is immaterial since it is the comparative (not absolute) costs that are being computed. In summary, for most of the cases reported in this study, we assumed the yearly base hardware expenditures will remain constant at fifty-million dollars after linearly increasing over a seven-year development period.

The model assumes that the nominal yearly processor expenditures (K_j) are a constant (u) times the yearly base hardware expenditures (B_j),

$$K._{\alpha}=uB_j. \tag{15}$$

Nominal processor expenditures (K_j) comprise the nominal CPU (P_j) and nominal main memory (M_j) expenditures but exclude expenditures for I/O busses, devices, and other peripheral gear whose costs are insensitive to the architecture of the processor,

$$K_j=P_j+M_j. \tag{16}$$

We assumed in most cases that the constant u was 0.5; that is, we took the total yearly CPU and main memory costs to be one-half the overall CFA base hardware expenditures. This assumption compares well with a recent survey of DP budgets.[21] In order to measure the sensitivity of the model to u, we also used values of 0.4 and 0.6 as explained in the next section.

The model assumes the nominal yearly main memory cost (M_j) is a fraction (α) of the nominal yearly processor expenditures (K_j),

$$M_j=\alpha K_j. \tag{17}$$

used on data on military computer systems found in References 15 and 22, it appears that α usually lies between 0.5 and 0.8. The value used in most of our calculations was 0.65.

Actual main-memory cost (M_{jk}'), which is architecture dependent, is computed from nominal main-memory cost as follows:

$$M_{jk}'=p_k s_k a M_j. \tag{18}$$

The two coefficients, p_k and s_k, are included in this equation because the cost of memory depends upon the efficiency with which an architecture stores programs as well as the rate at which it uses memory in executing them. The parameter s_k, the normalized s-measure, is indicative of the memory space required to represent a program when using architecture k. This space includes all the storage required to represent and execute the program exclusive of input/output used by the program (since the same data arrays are used by all candidate architectures). The quantity p_k in equation (18) is a cost-to-performance coefficient for architecture k which the model obtains by combining the normalized m and r-measures as follows:

$$p_k=(Vr_k+Wm_k)^{1/g} \tag{19}$$

The m-measure, m_k, is a measure of the traffic between main memory and the central processor that is required to execute a program when using architecture k. The r-measure, r_k, is a measure of the data traffic internal to the central processor. The exponent g follows from a general relationship, probably first examined by Grosch,[23] between performance and cost, i.e., performance=K×cost^g. Rein Turn[9] gives a value of g between 2 and 3; the SADPR-85 Study Group,[8] using more recent data, argues for a lower value of about 1.5. Although this latter value was used in most of our calculations, we also investigated the effects of using values of 1 and 2. The parameters V and W are weighting factors for the r- and m-measures. Their sum is assumed to be one. Because the values obtained for the r- and m-measures are almost equal for each of the candidate architectures, making r_k insensitive to the values of V and W, we took both V and W to be 0.5.

The constant a appears in equation (18) because parts of the main memory are insensitive to computer architecture—in particular, the portion occupied by data arrays. The architecture sensitive fraction, a, is called the main-memory static-storage ratio. Most of the calculations assume a equals 0.8. The sensitivity studies, however, also examine the effects of using values of 0.7 and 0.9.

The nominal yearly and architecture-independent central processor expenditures (P_j) follow from combining equations (16) and (17),

$$P_j=(1-\alpha)K_j. \tag{20}$$

The actual architecture-dependent central-processor expenditures (P_{jk}') are given by:

$$P_{jk}'=p_kP_j \tag{21}$$

The model assumes the nominal secondary-memory expenditures (E_j) are independent of nominal processor costs (K_j) and are a fraction, v, of the yearly base hardware expenditures (B_j), i.e.,

$$E_j=vB_j. \tag{22}$$

From these it obtains the secondary-memory expenditures that are architecture-sensitive by

$$E_{jk}'=bs_kE_j. \tag{23}$$

The constant b, the secondary-memory static-storage ratio, is analogous to the main-memory static-storage ratio (a), and the parameter, s_k, is the normalized s-measure discussed in this section.

Systems described in References 15 and 22 led us to a value of v of 0.1 and b of 0.2. The sensitivity studies discussed in the next section examine the effects of choosing v equal 0.0 and 0.2 and b equal 0.1 and 0.3, and show a relative insensitivity of the model to uncertainties in the values of these parameters.

Architecture-Dependent Software Costs

For architecture k, the total yearly software expenditure (S_{jk}) is the sum of the yearly support-software expenditure (Q_j) plus the architecture-dependent application-software expenditure (A_jF_{jk}),

$$S_{jk}=Q_j+A_jF_{jk}. \tag{24}$$

We define support software to be software tools such as compilers, loaders, linkers, simulators, assemblers, submonitors, operating systems, and debug packages and we assume that the amount of money allocated for the development of these tools is independent of the architecture chosen. Hence, the model assumes that support software will be developed with an expenditure of Q_j dollars for each year j where j=1, . . . , 13. For the cases considered in this paper, we let value Q_j be constant at x million dollars, for j=2 through 13 and Q_1=0. The sensitivity studies examined the effects of allowing x to range from zero to eight million dollars in two million dollar increments. Although x is architecture independent, it plays a dominant role in determining the lowest cumulative cost alternative.

The model assumes that base (or nominal) applications-software expenditure in year j, denoted A_j, is a constant (ρ) times the yearly base hardware expenditure, B_j,

$$A_j=\rho B_j. \tag{25}$$

For lack of a better name, we call ρ the software-to-hardware ratio. Working with the reports of Fisher[11] and others,[8,9,19,21,24] we estimate ρ to be about 2.5 to 3 for general-purpose DoD computer systems, and possibly $^1/_2$ to 2 for embedded systems,[16] which usually have multiple deployments of the same software and hardware and do not have software bundled into the system price.

The constant ρ is one of the most crucial parameters in the model. Because of its importance and because of the difficulty of estimating its value, we determined the cumulative-cost ratios for 1985 and 1990 using values of ρ of $^1/_8$, $^1/_4$, $^1/_2$, 1, 2, and 4. Figure 2 shows the 1990 results for support-software expenditures (Q_j) of two million dollars and clearly illustrates the importance of ρ in determining the most cost-effective computer family architecture.

At this point, a word of caution is in order. The error analysis, to be described later, shows that the expected uncertainties of the values of the input parameters can result in substantial uncertainties in the cumulative cost ratios. They imply that the reader should interpret the PDP-11 and Interdata 8/32 curves of Figure 2 as ribbons having widths on the order of twenty to thirty percent of the illustrated values; the choice of the most desirable architec-

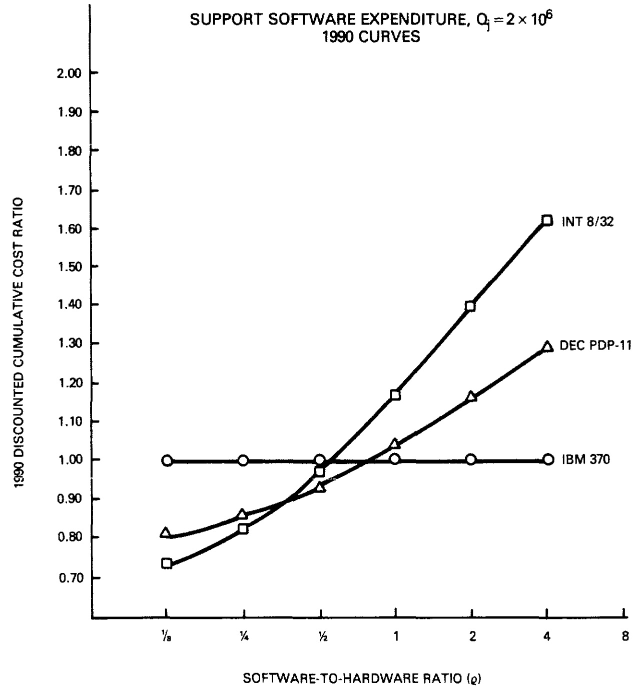


Figure 2

ture for values of ρ between 1/4 and 2 (the values expected in the military computing environment) is by no means clear.

The architecture-dependent applications-software costs, mentioned earlier, are A_jF_{jk}. Here F_{jk} is the amount of base (nominal) applications-software expenditure that should be used in determining the actual applications-software cost for architecture k. It is dependent upon the available support software for architecture k in year j.

In order to derive an expression for F_{jk} we began with the same curves shown in Figure 1 of the Bottom Up model. These curves give the cost per machine language instruction as a function of available support software. One-hundred percent support software means that an "ideal" set of software tools for military computer systems is available. W. Svirsky, T. Giles, and A. Irwin of System Development Corporation, West Long Beach, New Jersey, generated these curves on the basis of the results of a questionnaire sent to SDC project managers of five large scale command and control software efforts.[7,25] They noted, however, that the curves are largely judgmental and examples can be found that yield costs per instruction above and below the worst and best case data. The absolute cost per instruction does not affect the Top Down model, however, because the model depends only upon the relative cost-per-instruction vs support software, as opposed to the absolute cost per instruction given by the curve.

From a graph of the SDC median curve, it appeared to us that the interpolation function

$$y(s')=y_m+(y_M-y_m)\exp(-s'/c) \tag{26}$$

might provide a reasonably good fit. Here y is the dollars per machine language instruction, s' is the support software availability in percent, and y_m, y_M, and c are constants. Insisting that this function pass through the three points $(s', y)=(20, 32.5)$, $(50, 17.75)$, and $(80, 10)$, in accordance with the SDC supplied curve, we obtained $y_m=1.4196$, $y_M=49.151$ and $c=46.6171$. Next we assumed that

$$F_{jk}=k'y(s'), \tag{27}$$

where k' is a constant whose value is determined by the additional assumption that when the average of the available support software of the three alternative architectures in year 1, call it $\bar{s}'$, is substituted in equation (27) for s' we should obtain $F_{jk}=1$. Here we assume $\bar{s}'$ can be expressed as

$$\bar{s}'=\frac{100}{3}\left(\frac{U_{11}}{T_1}+\frac{U_{12}}{T_2}+\frac{U_{13}}{T_3}\right), \tag{28}$$

where U_{jk} denotes the available support software (measured in dollars) in year j for architecture k, and T_k the dollar value of a 100 percent support-software base (or ideal support-software base) for architecture k.

Several members of the CFA selection committee surveyed the industry and estimated values for the currently existing support-software base for each candidate architecture and also estimated T_k. The values they came up with were:*

	k	Currently Available Support-Software Base ($)	T_k ($)
IBM 370	1	31.049M	44.604M
DEC PDP-11	2	20.790M	43.893M
INT 8/32	3	14.100M	44.040M

Making the assumption that the currently available support-software base values given above could be used in the model for U_{1k}, $k=1, \ldots, 3$, we obtained from equation (28) that $\bar{s}'=49.7$, and

$$k'=1/y(\bar{s}') \\ =0.05601 \tag{29}$$

Expressing F_{jk} as

$$F_{jk}=f_m+(f_M-f_m)\exp(-hU_{jk}/T_k), \tag{30}$$

we have

$$f_m=k'y_m=0.0795 \tag{31}$$

$$f_M=k'y_M=2.7528 \tag{32}$$

and

$$h=100/c=2.1451. \tag{33}$$

The dollar value of the available support software in year j for architecture k, U_{jk}, is given by

$$U_{jk}=U_{1k}+\sum_{m=2}^{j} Q_m, \tag{34}$$

where Q_m is dollar expenditure for support software in year m.

The values of Q_m have a significant impact on the results of the model. Demonstrating this impact, Figure 3 shows discounted total cumulative cost ratios for 1990. The cases shown assume that $Q_m=x$ for $m=2, \ldots, 13$ and that x ranges from 0 to 8 million dollars in 2 million dollar increments, and that ρ varies by doubling from 1/8 to 4. Figure 2 is, of course, a subset of these results. These figures show that by increasing support-software expenditures we can effectively counter high values of ρ that may be characteristic of certain computing environments. They also show that as support-software expenditures increase, the differences between the three candidate architectures decrease. For low values of ρ the Interdata 8/32 appears most desirable, for high values of ρ the IBM S/370 appears to be the best choice; for intermediate values of ρ (between 1/4 and 1) the DEC PDP-11 may be best. As mentioned earlier, because of the uncertainties of the input parameters, these results should be interpreted with discretion. Table IV is an example of the yearly software and hardware costs data for a typical case (number 18) shown in Figure 3.

* M denotes millions.

TABLE IV

Year Case 18 (Reference)

	1	2	3	4	5	6	7	8	9	10	11	12	13
	78	79	80	81	82	83	84	85	86	87	88	89	90
Base Appl. Sftwe Exp.	7.14	14.3	21.4	28.6	35.7	42.8	50.0	50.0	50.0	50.0	50.0	50.0	50.0
Base Hdwe Exp. ($Mil)	7.14	14.3	21.4	28.6	35.7	42.8	50.0	50.0	50.0	50.0	50.0	50.0	50.0
Total Yearly Software													
(1) IBM ($Mil)	4.86	10.9	14.3	17.1	19.4	21.3	22.8	21.3	19.9	18.6	17.5	16.4	15.4
(2) DEC	7.48	15.7	20.8	24.9	28.2	30.8	32.9	30.4	28.1	26.0	24.2	22.5	21.0
(3) INT	10.2	20.6	27.4	33.0	37.4	40.8	43.5	40.0	36.8	34.0	31.4	29.0	26.9
Total Yearly Hardware													
(1) IBM ($Mil)	4.26	8.59	12.8	17.2	21.4	25.7	30.0	30.0	30.0	30.0	30.0	30.0	30.0
(2) DEC	3.09	6.22	9.32	12.4	15.5	18.6	21.8	21.8	21.8	21.8	21.8	21.8	21.8
(3) INT	2.58	5.20	7.77	10.4	13.0	15.5	18.2	18.2	18.2	18.2	18.2	18.2	18.2
Total Discounted Cumulative Cost													
(1) IBM ($Mil)	8.70	25.6	47.0	71.6	98.3	126	154	179	202	221	239	254	268
(2) DEC	10.1	29.1	52.8	79.6	108	137	166	192	214	234	250	265	278
(3) INT	12.2	34.5	62.2	93.3	126	160	192	221	245	266	284	300	314
Total Discounted Cum. Ratios													
(1) IBM	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
(2) DEC	1.16	1.14	1.12	1.11	1.10	1.09	1.08	1.07	1.06	1.06	1.05	1.04	1.04
(3) INT	1.40	1.35	1.32	1.30	1.28	1.26	1.25	1.23	1.21	1.20	1.19	1.18	1.17
Discount Factor (10%)	0.954	0.867	0.788	0.717	0.652	0.592	0.538	0.489	0.445	0.405	0.368	0.334	0.304

$\rho = 1$ $Q_j = 2\text{x}10^6$ Dev. Cycle = 7 $\alpha = 0.65$ $g = 1.5$ $a = 0.8$ $b = 0.2$

$s_1, s_2, s_3 = 1.208, 1.000, 0.828$ $m_1, m_2, m_3 = 1.266, 0.928, 0.850$ $r_1, r_2, r_3 = 1.292, 0.938, 0.825$

Results and sensitivity studies

Figures 2 and 3 summarize the results for 1990 of varying the software-to-hardware ratio (ρ=⅛, ¼, ½, 1, 2, 4) and the support-software expenditures (Q_j=0, 1×10^6, 2×10^6, 4×10^6, 8×10^6). These variations constitute the first thirty cases of the sensitivity studies.

All of the sensitivity studies assume the annual discount rate is ten percent (the value recommended by References 13, 14 and used in Reference 8).

Further assessing the model's sensitivity, we selectively perturbed individual input parameters about case number 18's (ρ=1 and $Q_j=2\times10^6$) input data set to obtain twenty-nine additional input data sets. We chose the case 18 data set as a reference because, at this time, it appears to represent the most reasonable set of data based on our current understanding of the requirements of military tactical computer systems, the candidate architectures, and probable life of the MCF.

Figure 4 summarizes the results of the sensitivity studies (for cases 31 through 59) for 1990. The dotted lines in these figures are indicative of the discounted cumulative cost ratios for the reference data set (case 18), and the arrows show the amount and direction of movement from these values as the result of parameter perturbation. From these figures, we see uncertainties in the percentage of the ideal support-software base available in year 1 (U_{1k}/T_k), and uncertainties in u, the ratio of yearly processor expenditure to base hardware expenditure (K_j/B_j), can have significant impacts on the results.

Error analysis

Making the assumption that the errors in the measurements of the model's parameters are normally distributed, the variance of the mean of the calculated discounted cumulative cost ratio R_{jk}* due to variances of the means of the model parameters X_m, m=1, . . . , N_p, can be approximated by

$$\sigma_{jk}{}^2 = \sum_{m=1}^{N_p} \left(\frac{\partial R_{jk}{}^*}{\partial X_m}\right)^2 \sigma_m{}^2, \tag{35}$$

where N_p is the number of parameters under consideration, and σ_m is the standard deviation of the mean of the parameter X_m.[26,pp.97-98]

The partial derivatives were derived from the results of the sensitivity studies in the preceding section. The standard deviations (σ_m) were obtained for each parameter by the method of "educated guess." It should be noted that the standard deviations for ρ and Q_j were assumed to be zero. This is not because they are actually zero (in fact, they are quite large) but because we desired to obtain an estimate of the standard deviation as a function of ρ and Q_j for Figure 2. Here, Q_j and ρ are independent variables.

The results of these calculations indicate that the uncertainties in the model's major results are quite large. Table V summarizes the findings for a software-to-hardware ratio of one ($\rho = 1$) and support-software expenditures of two million dollars per year.

TABLE V

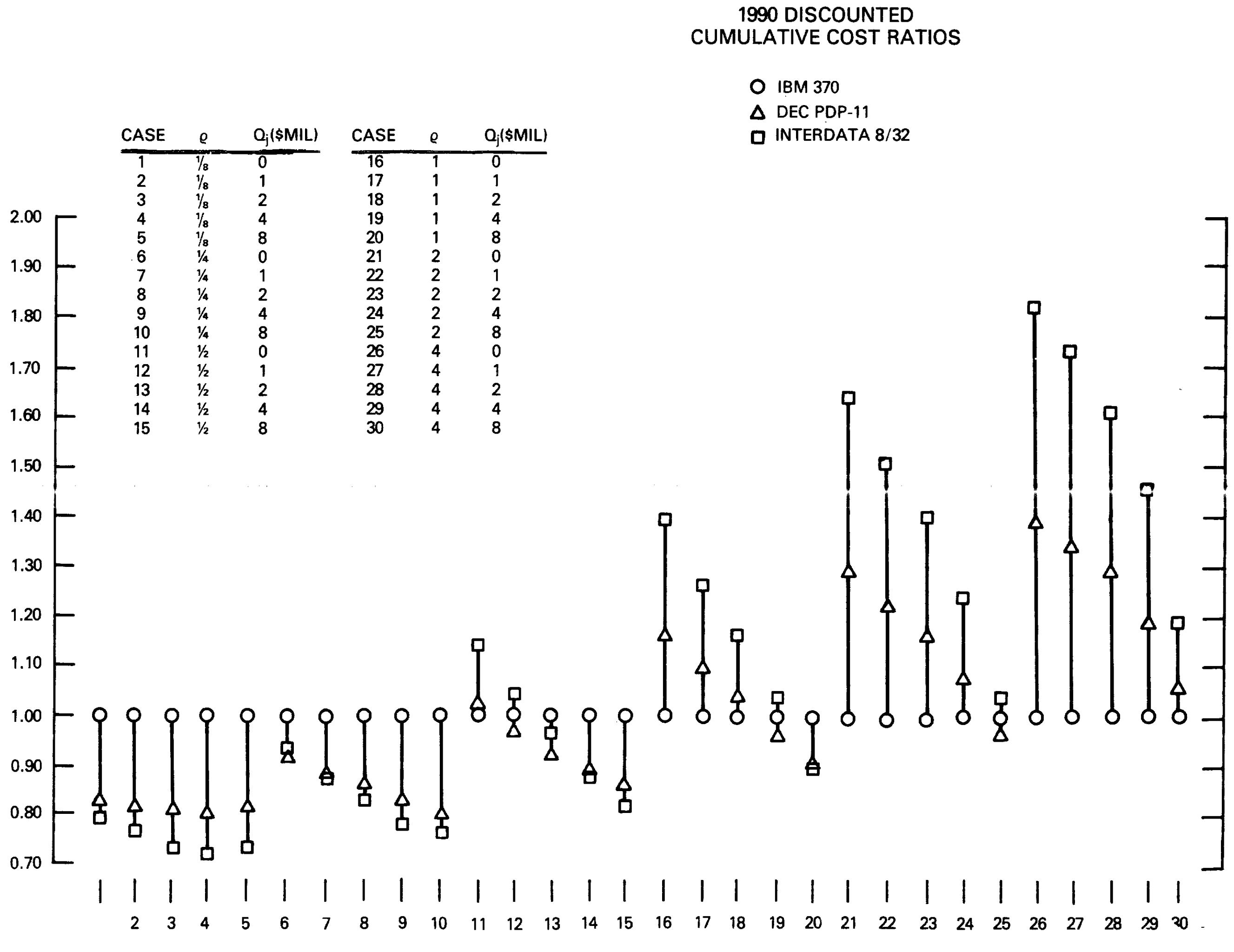
	K	R_{8k}*	σ_{8k}
IBM	1	1.00	0.000
DEC	2	1.06	0.205
INT	3	1.22	0.255

These results imply that because of uncertainties in our input data we cannot clearly resolve the question of which architecture is the most cost effective for this software-to-hardware ratio.

INTERPRETING THE RESULTS OF THE MODELS

The two models serve as checks against one another. The bottom-up results indicate that in most circumstances the DEC PDP-11 is superior to both the IBM S/370 and Interdata 8/32 architectures. The top-down model results, on the other hand, indicate that the S/370 is superior for high (greater than one) software-to-hardware cost ratios, while the Interdata 8/32 is slightly better for low (less than one-fourth) ratios, and the PDP-11 is best in between. These apparently conflicting results were found to be due to uncertainties in the input data, to different input requirements, to contrasting basic model assumptions, and to different methods of combining the same input data.

For example, the bottom-up model weights the raw S, M, and R data, provided by CMU for the individual test programs, according to the estimated relevance of each



CASE	ρ	Q_j($MIL)	CASE	ρ	Q_j($MIL)
1	⅛	0	16	1	0
2	⅛	1	17	1	1
3	⅛	2	18	1	2
4	⅛	4	19	1	4
5	⅛	8	20	1	8
6	¼	0	21	2	0
7	¼	1	22	2	1
8	¼	2	23	2	2
9	¼	4	24	2	4
10	¼	8	25	2	8
11	½	0	26	4	0
12	½	1	27	4	1
13	½	2	28	4	2
14	½	4	29	4	4
15	½	8	30	4	8

Figure 3

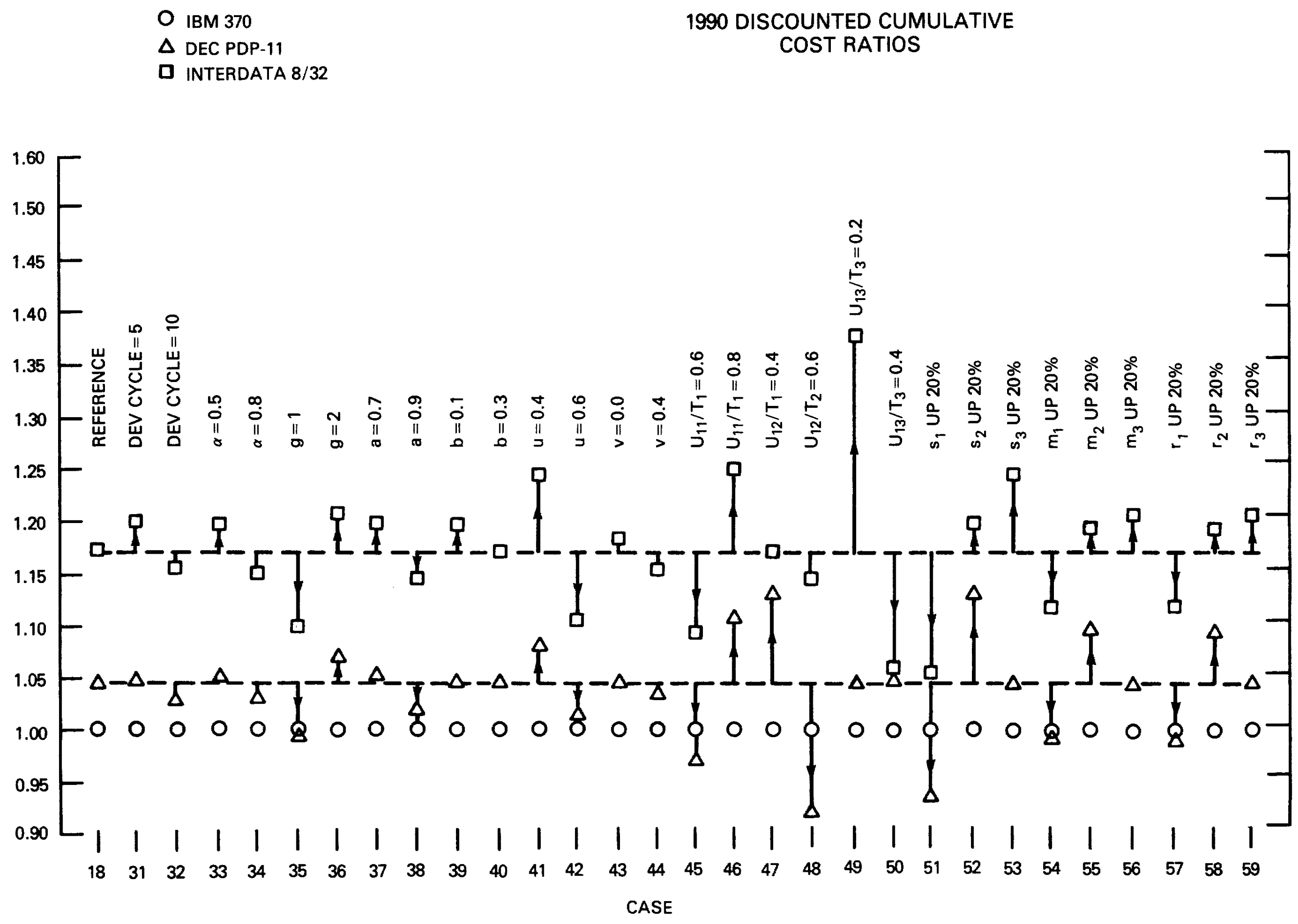


Figure 4

program in each system application. It then combines these into the composite processor speed and static storage ratios, a_{ij}, and b_{ij}. The top-down model, on the other hand, uses composite S, M, R ratios, which were derived by CMU from the individual S, M, R measures for each test program and which were weighted by CMU to obtain minimum statistical variance in these ratios rather than to reflect the importance of particular application programs. A result of these two different approaches to using the individual test program S, M, and R data is a difference in the computed architectural efficiency of the Interdata 8/32 as compared to the PDP-11. In the first case they are comparable, in the second the 8/32 is superior. A further result of this difference is that if the unweighted S, M, and R data are used in the bottom-up model, then the 8/32 becomes the superior architecture in the 1976 calculations when the hardware-software cost ratio is high. This agrees with the top-down model results. Conversely, if the S, M, and R data used in the top-down model were weighted as in the bottom-up model, better agreement between the models would result.

As another example, the assumptions leading to the ratio of software costs to hardware costs are clearly among the most important to the ultimate results, while at the same time are among the most difficult to support with actual data.

The uncertainty calculations in the preceding section for the top-down model could be applied to the bottom-up model with similar results expected. Because of the size of these uncertainties, the results of the models must be interpreted with caution. By chance, each of the three architectures evaluated had either superior hardware attributes (the 8/32), or superior software attributes (the S/370), or a good combination of the two (the PDP-11). As a result, the combined hardware-software effectiveness of the three architectures were relatively close. Probably the strongest conclusions to be derived from the life cycle cost evaluations are that, within the uncertainties resulting from propagating errors in the input data throughout each model's calculations, (1) the models agree and (2) all three architectures would be comparable choices based on life-cycle costs.

REFERENCES

1. Smith, W. R., "AADC Computer Family Architecture Questions and Answers," *ACM Computer Architecture News*, Vol. 4, No. 3, Sept. 1975, pp. 15-21.

2. Salisbury, A. B., "MCF: A Military Computer Family for Computer-Based Systems," *Signal,* July, 1976.
3. Coleman, A. H., "Army/Navy Military Computer Family," *Digest of Papers of COMPCON 76,* IEEE. Cat. No. 76 CH1115-5C, Thirteenth IEEE Computer Society International Conference, Washington, D.C., Sept. 7-10, 1976, pp. 230-232.
4. Estell, R. G., R. P. Sabin, and W. R. Smith, "Final CFA Selection Methodology Subcommittee Preliminary Report," April 1976.
5. Cornyn, J. J., W. R. Smith, A. H. Coleman, and W. Svirsky: "Life Cycle Cost Models for Comparing Computer Family Architectures," *AFIPS Conference Proceedings,* Vol. 46, 1977 National Computer Conference.
6. Cornyn, J. J., "Top-Down Life-Cycle Cost-Analysis Model for Selecting a Computer Family Architecture," Naval Research Laboratory, Washington, D.C., August 1976.
7. Svirsky, W., T. Giles, and A. Irwin, "Life Cycle Cost Analysis of Computer Family Architecture (CFA) Finalists Within Army Embedded Computer Systems," System Development Corporation, unpublished manuscript generated for CFA Selection Committee, August 1976.
8. SADPR-85 Study Group, "Support of Air Force Automatic Data Processing Requirements through the 1980's (SADPR-85)" Electronics Systems Division, Hq. ESD (MCS), Hanscom AFB, MA. Six Volumes. ESD-TR-74-192. Cited by Reference 24.
9. R. Turn, *Computers in the 1980's,* Columbia Univ. Press, New York and London, 1974.
10. Fuller, S. H., W. E. Burr, P. Shaman, and D. A. Lamb, "Evaluation of Computer Architectures via Test Programs," *AFIPS Conference Proceedings,* Vol. 46, 1977 National Computer Conference.
11. Fisher, D. A., "Automatic Data Processing Costs in the Defense Department," Institute for Defense Analyses, Arlington, Virginia, IDA Paper P-1046, October 1974.
12. Wagner, J., et al., "Procedure for and Results of the Evaluation of the Software Bases of the Candidate Architectures for the Military Computer Family," 6 August 1976, prepared by the Software Evaluation Subcommittee.
13. Shishko, R., "Choosing the Discount Rate for Defense Decision Making," Rand Corp, Santa Monica, Calif., R-1953-RC, July 1976.
14. AFR-172-2; DODI 7041.3, "Economic Analysis of Proposed Investments," 30 December 1969, Attachment 2, p. 42. Cited by Reference 8, Vol. V, p. X-9.
15. Kossiakoff, A., T. P. Sleight, E. C. Prettyman, J. M. Park, and P. L. Hazan, "DOD Weapons System Software Management Study," Johns Hopkins Univ., Applied Physics Lab, Laurel, Md., June 1975, APL/JHU-SR-75-3, AD-A022 160/6WC. Abstract in Comp., Control, and Info. Theory, May 3, 1975.
16. Chapin, G. C., "What is Different About Military Operational Programs?," *AFIPS Conf. Proc.* Vol. 42, 1973, Nat. Comp. Conf. pp. 787-795.
17. Premo, A. F., Jr., "Computer Software: Estimating Guidelines," *COMPCON 76, Digest of Papers,* IEEE Pub. No. 76CH1115-5C, Sept. 7-10, 1976, pp. 146-151.
18. Boehm, B. W., "Information Processing/Data Automation Implications of Air Force Command and Control Requirements in the 1980's (CCIP-85), Vol. IV, Technology Trends: Software," Space and Missile Systems Organization, AFSC, Los Angeles, Calif., October 1973, AD 919267L.
19. Boehm, B. W., et al., "Information Processing/Data Automation Implications of Air Force Command and Control Requirements in the 1980's (CCIP-85), Vol. I, Highlights," April 1972, SAMSO/XRS 71-1, U.S. Air Force, AD-900031L. Cited by References 9 and 27.
20. Private Communication, meeting of D. Fisher, W. Smith, and J. Cornyn on May 3, 1976.
21. McLaughlin, R. A., "1976, DP Budgets," *Datamation,* February 1976, pp. 52-57.
22. System Development Corp., "Embedded Computer System Data Processing Requirement for U.S. Army Weapon/Data Systems," Draft, 1 March 1976, prepared for CENTACS, U.S. Army Electronics Command, Ft. Monmouth, N.J., Contract DAAB07-76-C-0334.
23. Grosch, H. R. T., "High Speed Arithmetic: The Digital Computer as a Research Tool," *J. Optical Soc. of Amer.,* Vol. 4, No. 4, April 1953, pp. 306-310.
24. Withington, F. G., "Beyond 1984—A Technology Forecast," *Datamation,* January 1975, pp. 54-73.
25. Private Communication with Al Irwin, W. Svirsky, and Tom Giles of System Development Corporation. SDS data is also cited on p. 88 of Reference 28.
26. Young, H. D., *Statistical Treatment of Experimental Data,* McGraw-Hill Book Company, Inc., 1962.
27. Boehm, B. W., "Keynote Address: The High Cost of Software," TRW, Redondo Beach, Calif., in *Proceedings of a Symposium on the High Cost of Software,* Sept. 17-19, 1973, at Naval Postgraduate School, Monterey, Calif., Stanford Research Institute, Meno Park, Calif., SRI Proj. 3272, pp. 27-40.
28. Brooks, F. P., Jr., *The Mythical Man-Month,* Addison-Wesley Publishing Co., Reading, Mass., 1975.
29. Manley, J. H., "Embedded Computers—Software Cost Considerations," *AFIPS Conf. Proc.* Vol. 43, 1974, pp. 343-347.
30. Taback, M. A. and M. C. Ditmore, "Estimation of Computer Requirements and Software Development Costs," General Research Corp., Santa Barbara, California, RM-1873, March 1974, AD-782-220/8WC, p. 20.
31. Wolverton, R., "Cost of Developing Large-Scale Software," *IEEE Trans. on Computers,* Vol. C-23, No. 6, pp. 615-636, June 1974. Cited by Reference 28.

A microprocessor architecture for digital device implementation*

by THOMAS L. BOARDMAN, JR.
University of Colorado
Boulder, Colorado

ABSTRACT

A microprocessor based architecture for the implementation of digital logic devices is presented. The architecture facilitates replacement of portions of the hard-wired logic chains within the device with groups of microinstructions called kernels. Multiple modules of a device may share the processor simplifying interface problems such as buffering, interlocking, and sequencing. This greatly reduces overall package count, power consumption, system complexity, and therefore system debug time and cost while improving system flexibility with the programmable control store. A three dimensional graphics display processor for the Tektronix 4014 terminal, built using Intel 3000-series microprocessor elements, is discussed demonstrating the viability and benefits of this architecture. Such processor elements permit approximately 250,000 twenty microinstruction kernels to be executed in place of hard-wired logic per second.

INTRODUCTION

The control of basic digital logic functions such as sequencing, data flow, and arithmetic unit operation by bits in a programmable control store is at least as old as the EDSAC II (1953). Until recently, however, this microprogramming of logic functions served largely to improve system design flexibility, not to reduce size, power consumption, cost, or improve performance. The development of microprocessors using MSI and LSI technologies has provided these *additional benefits* and therefore significantly affected the digital logic design process.[1]

Initial utilization of these microprocessors,[2-4] centered around the Intel 8080 and similar hundred thousand instruction per second processors, was characterized by replacement of large sections of digital logic with serial execution of stored programs interfaced to the outside world and internal SSI logic. Such utilization made products such as calculators, point of sale terminals, adaptive traffic control systems, etc. economically feasible and will undoubtedly continue to represent a major part of the intelligent device market. They are, however, limited to applications requiring at most several thousand operations per second since each operation must be implemented as a series of machine instructions.

Recent bipolar LSI technology has produced processors such as the Intel 3000 and AM2900 series devices which are capable of executing several million instructions per second. These processors, while capable of handling orders of magnitude faster applications, are multi-package systems and therefore are not necessarily as cost or performance effective. In addition, since their multipurpose nature demands many internal logic levels between input and output, they can never completely duplicate the *performance* characteristics of SSI logic.

This paper describes an architecture for the interconnection of such bipolar microprocessors and SSI logic which can provide major logic function unit replacement for low speed-requirement operations, minor function replacement for higher speed-requirements, and direct logic execution of time-critical operations. Examples of these modes include matrix multiplication every several milliseconds (hundreds of instructions), multiple-bit shift or buffer push/pop every several microseconds (several instructions), and bit serialization for disk transfers (hardwired logic).

SYSTEM ARCHITECTURE

The microprocessor system referenced herein as an example of the architecture being presented was designed to support eight independent logic devices such as disk units, communication lines, CRT's, and multiply-divide units. It was implemented using Intel 3000 series microprocessor elements with a sixteen-bit word size (eight two-bit arithmetic unit slices) and a sixty-bit microinstruction word. Use of Intel components and these specific field sizes should not be considered characteristics or requirements of the basic architecture.

Whether it be along communication lines or wires etched in a single circuit board, signals must be bussed between the processor and the various devices of which it is a part. The seventy bus lines required by this implementation are shown in Figure 1. Thirteen lines are used for power

* This report was prepared as a result of work performed under NASA Contract No. NAS1-14101 while the author was in residence at ICASE, NASA Langley Research Center, Hampton, VA 23665

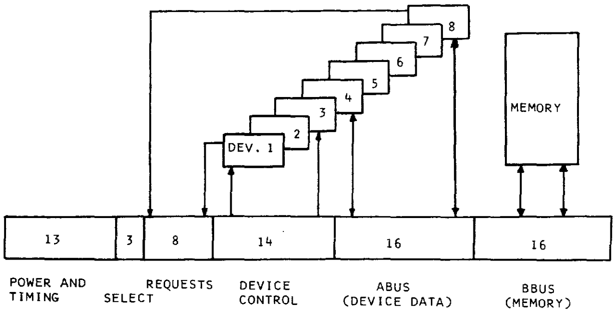


Figure 1—System bus structure

distribution and timing signals which are described below. Three lines specify which one of the eight devices is being selected by a particular instruction. The following eight lines allow the devices to request (interrupt) the processor for execution of specific groups of logic-replacement microinstructions. The interrupt structure, which makes the processor appear as a dedicated slave to each device, is also described below. The following fourteen bus lines are used to control the devices by initiating transfers to and from the processor and operations at the various points in the device logic chain. The next sixteen bits provide a bidirectional data bus for inputs to and outputs from the processor. Finally, the scratchpad read-write memory associated with the processor and thereby available to all devices is addressed and accessed over a second bidirectional bus.

As with most bipolar microprocessors, the arithmetic and processor (next address calculation) functions are performed by separate chips within the Intel 3000 family. Figure 2 shows the specific configuration of these chips. Figure 3 lists the various fields which make up the sixty-bit microinstruction. Many of these are specific to the 3000-series elements and will not be described in detail. Others will be referenced in the more detailed description of the processor implementation which follows.

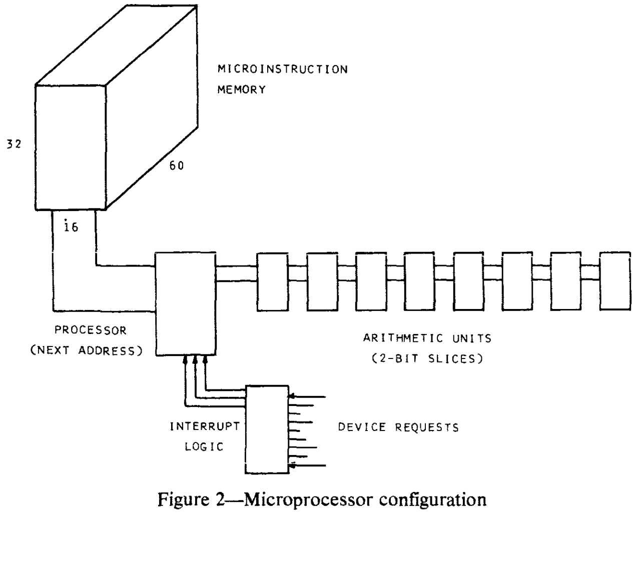
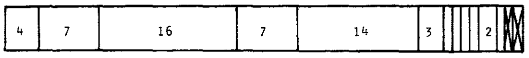


Figure 2—Microprocessor configuration



(4)	CARRY CONTROL	(1)	INTERRUPT DISABLE
(7)	NEXT ADDRESS FUNCTION	(1)	ARITHMETIC UNIT DISABLE
(16)	MASK INPUT TO PROCESSOR	(1)	BRANCH ON CARRY IN
(7)	ARITHMETIC UNIT FUNCTION	(1)	BRANCH ON CARRY OUT
(14)	DEVICE CONTROL	(2)	MEMORY CONTROL
(3)	DEVICE SELECT	(1)	ZERO PROCESSOR INPUTS

Figure 3—Microinstruction fields

PROCESSOR-DEVICE COMMUNICATION

Communication between the microprocessor and each of the devices, which occurs across the seventy-line bus described above, is controlled by the timing signals shown in Figure 4. The system is driven by a single twenty megahertz CLOCK to insure that the devices and processor remain synchronized.

An individual microinstruction begins execution with the leading edge of the CPU CLOCK. This initiates the next address calculation in the processor chip requiring approximately fifty nanoseconds. Once the address is calculated, it is presented to the (60-bit wide) microinstruction memory and the instruction is fetched. This requires sixty nanoseconds for the memory used in this implementation. At the same time, the bidirectional busses are switched to provide input from the devices and scratchpad memory to the arithmetic unit bit-slices. As the instruction is being fetched, an END CYCLE pulse is sent out terminating execution of the previous instruction. Although all devices receive this signal, it means only that they must relinquish the bus and access to the processor. Internal logic chains and interaction with their external devices may continue.

Once fetched, a portion of the microinstruction is LATCHED to permit overlapping of this instruction with the next instruction fetch. The remainder of the instruction is presented to the processor elements directly and to the devices over the bus as indicated in Figure 1. The new

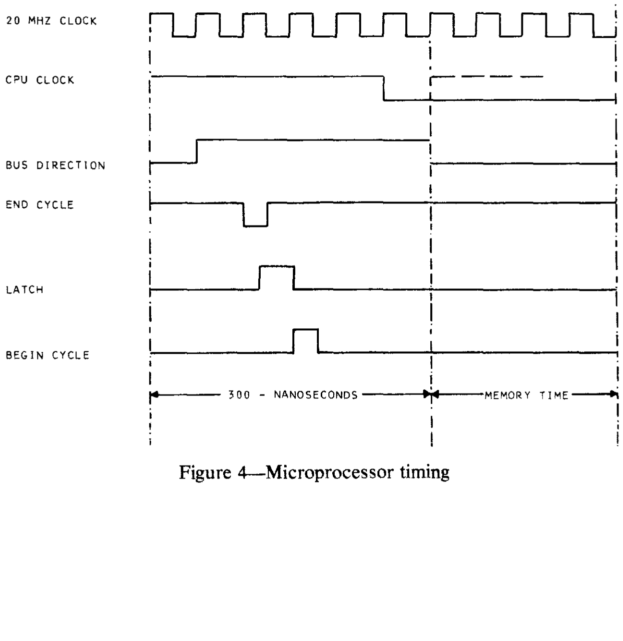


Figure 4—Microprocessor timing

instruction is then initiated with the BEGIN CYCLE pulse causing a specific device indicated in the instruction word to be selected. If requested by the appropriate control bits, the device immediately places data on the bus as input to the arithmetic units. After a settling time, the falling edge of the CPU CLOCK initiates execution of the microinstruction specified in the instruction word. As indicated, the arithmetic unit has access to both device data and memory data for that execution.

After approximately fifty nanoseconds, the results of the arithmetic unit execution are sent to the device selected and to memory as the BUS DIRECTION is reversed. If this is a memory reference instruction, the cycle is extended until the memory signals that it is complete. Otherwise, the CPU CLOCK rises immediately and the microinstruction execution process is repeated beginning with the next address calculation.

Notice that the device can specify input to the arithmetic unit, wait for it to be processed, read the result, and perhaps read a resulting value from memory (since the bidirectional busses again reverse) before the END CYCLE pulse terminates the instruction. This represents a significant characteristic of the architecture since most processors permit only read device, or write device, or access memory on a single cycle. In the class of interfacing problems for which this architecture was intended: read, process, and write operations are very common.

SYSTEM INTERRUPT STRUCTURE

The previous section described the event sequence for the execution of a single instruction. Since the next address opcode is part of each microinstruction (Figure 3), sequential execution of instructions is unnecessary. Groups of microinstructions required to perform a specific function can, however, be thought of as logically sequential kernels. These kernels perform operations for the digital devices to which the processor is connected ranging from single instruction shift operations to complex operations such as a matrix multiplication requiring scores of instructions.

To maximize the usefulness of the microprocessor to the individual devices, execution of these kernels must be possible at any point in a device's logic chain. The hardware interrupt structure provides this capability. In parallel with the execution of each instruction, the interrupt logic shown in Figure 5 monitors requests coming across the bus from all (eight) devices. At the point in the execution cycle immediately after the processor has computed the next address, the highest priority request is compared against the priority (number) of the device currently selected. If the requesting device is of higher priority, the computed next address is optionally stored in scratchpad memory and an instruction dedicated to handling interrupts from the requesting device is fetched.

The fetched interrupt handling instruction causes the contents of a scratchpad memory location (interrupt vector) dedicated to the interrupting device to be read, and branches to an instruction common to all device interrupt handling. This second instruction copies the value read, which is the microprogram address to be executed next for that device, into the processor so that it can be used as the next address executed, as depicted in Figure 5. Processing (kernel execution) for the interrupting device begins on the third instruction after the interrupt was received.

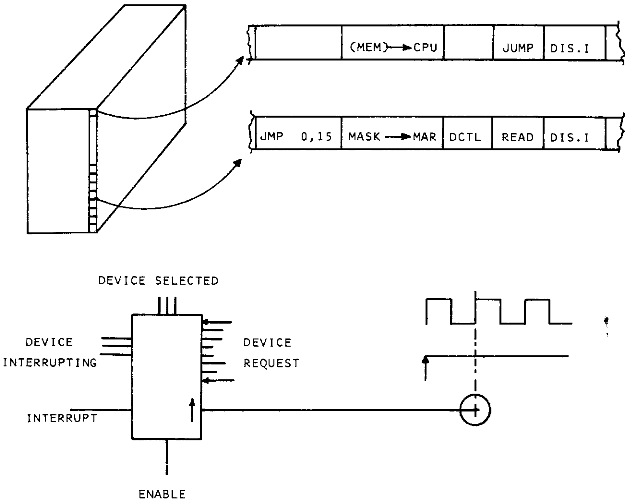


Figure 5—Interrupt structure

Any kernel can include instructions to change the contents of the interrupt vector and therefore point to a new kernel to be executed following the next interrupt from its device. This allows the processor to perform widely varied tasks for a device at different points in its logic chain. Since each microinstruction includes fourteen device control bits (Figure 3), the processor can provide not only logic simulation but also sequencing of the logic chains within the device.

Return from interrupts is handled by the same logic and procedure by which they are initiated. The end of each interrupt processing kernel is a branch to a NOP instruction of lowest priority. Execution of this NOP is immediately interrupted by the previously interrupted kernel, its interrupt vector is read, and that value becomes the next address executed.

It was mentioned above that the next address calculated immediately before an interrupt (return address) is *optionally* stored in the interrupt vector for that device. If the option is not invoked (as specified by a bit in the microinstruction word shown in Figure 3), the address is not stored and the return procedure will reinitiate execution at the point previously set in this vector address. This will typically cause re-execution of some instructions. It allows, however, general purpose arithmetic unit registers to be used without fear of their being altered between execution of instructions within a kernel since those instructions will be re-executed if an interrupt does occur. This represents another significant characteristic of the architecture as it reduces both microprogram size and save-restore time. Conventional register-save schemes are inappropriate due to the relatively small register complement in microprocessors, the high interrupt rate (virtually all kernels are executed in response to device requests), and the relatively small size of most kernels.

An interrupt disable bit is included in the microinstruction word for time-critical sequences of instructions. In addition, the priority scheme protects higher speed devices from being delayed by slower ones. Generally, however, it is expected that hard-wired logic will be used to implement time-critical functions utilizing the ease in switching between logic and microinstruction kernels inherent in this architecture.

SPECIFIC IMPLEMENTATION CHARACTERISTICS

Although specific details of the implementation are not critical to the microprocessor architecture presented herein, they are discussed briefly as one example of its utilization. Intel 3000 series components were used to build a prototype system to serve as a display controller for the Tektronix 4014 graphics terminal. Devices connected to the processor include an interface to the 4014, a floppy disk controller, a serial interface to a host computer, and a multiply-divide unit. The system is capable of receiving segmented images over a serial communication line from a host computer, massaging this display data into a compact form, storing it on the disk unit, and displaying the images with three-dimensional translation, rotation, and scaling in both the store and refresh modes of the 4014. Functions performed by the microinstruction kernels ranged from single instruction read character and store in memory to using the multiply-divide unit to multiply four by four matrices for coordinate transformation.

Idiosyncracies of the Intel 3000 components permitted a bit in the microinstruction word, shown in Figure 3, which disables the arithmetic units so that non-destructive tests may be performed. In addition, deficiencies in the conditional branch characteristics required bits specifying branch on carry-in and carry-out.

ADVANTAGES OF THIS ARCHITECTURE

The computer architecture described in the previous sections is intended to provide an approach to simplifying digital logic unit design. Its major feature is the incorporation of a high speed microprocessor to replace portions of the digital logic chain with sequences of programmable microinstructions. When properly applied, this will reduce the package count, wiring complexity, power consumption, and therefore overall system cost.

In applications where timing characteristics permit, multiple devices can draw on a single microprocessor for logic replacement. In addition to the obvious cost and complexity reduction benefits, this significantly simplifies the intradevice communication problems. Scratchpad memory can serve as a buffer to mask the effects of differing device speeds. Internal processor registers can be used to maintain a single copy of interlock and device communication controls. Perhaps most significant, the serial nature of the microprocessor can reduce device race-condition conflicts and serve to isolate the devices for debugging.

Naturally, the usefulness of including a microprocessor in a digital design will depend on the extent to which it can perform logic functions required by the design. Arithmetic and shift operations, common within these processor elements, easily replace hard-wired logic causing significant reduction of space consuming data path wiring. Temporary storage of data and control information, simplified by the processor's internal registers and scratchpad memory, is another candidate for logic replacement. In addition, the processor's ability to control the device sequencing using device control bits and updating the interrupt vector to point to different kernels (states) can significantly increase design flexibility and reduce re-wiring during the debug process.

A final advantage involves simplification of the hard-wired logic debugging. Since, within this architecture, the processor kernels are interacting with the device at various states in the logic chain, test programs can be written to repetitively active isolated portions of the logic. This allows modular debugging and provides repetitive signals necessary for good oscilloscope traces.

A NOTE ON SPEED

In the preceding sections, specific timing characteristics have been avoided as they do not directly affect the architecture presented. The microprocessor system implemented as an example has a basic instruction time of 300-nanoseconds with an additional 200-nanoseconds required for memory reference instructions. That speed limitation is largely due to characteristics of the Intel 3000 elements and implies a minimum interrupt service time of one microsecond. (This includes one memory reference instruction to read the vector address, one non-memory reference instruction branch to the kernel, and execution of the first instruction in the kernel.) This example could therefore sustain megacycle request bursts and approximately 250,000 processor requests per second assuming ten to twenty instruction kernels. This seems adequate for most logic systems, and has proven so for the graphics display processor application.

The real timing issue, however, is not absolute speed but rather the relative speed of the processor compared to available hard-wired logic. Assuming equivalent technologies, the speed difference will depend on the overhead in gate levels necessary to provide multiple functions within the microprocessors. Evidence suggests[1] that this speed reduction (or processor complexity) factor is fifteen to twenty. The microprocessor architecture described herein is oriented toward interleaving kernel execution and hard-wired logic. The speed factor implies that the processor will be usable for those functions in the logic chain where the design timing requirement is at least twenty times slower than the basic logic time necessary to perform the function.

SUMMARY

A microprocessor-based system architecture has been presented for the design of digital devices. It is centered around the interconnection of the microprocessor and digital devices in such a way that various portions of the digital logic chain can be replaced with sequences of microinstructions. Where multiple devices are augmented with a single processor, the architecture provides a very convenient interface between them. A prototype graphics display controller was built using Intel 3000 series microprocessor elements which has demonstrated the viability of the architecture for realistic digital design problems.

REFERENCES

1. Rattner, J., J. C. Cornet and M. E. Holt, Jr., "Bipolar LSI Computing Elements Usher in New Era of Digital Design," *Electronics*, September 1974, pp. 89–96.
2. Bailey, S. J., "Microprocessor: Candidate for Distributed Computing Control," *Control Engineering,* Vol. 21, No. 3, March 1974, pp. 40–44.
3. Hoff, M. E., Jr., "New LSI Components," *6th IEEE Computer Society International Conference Digest*, December 1972, pp. 141–143.
4. Weissberger, A. J., "Distributed Function Microprocessor Architecture," *Computer Design,* November 1974.

A hybrid computer interface for microprocessors

by JOSEPH P. HEID

General Electric Company
King of Prussia, Pennsylvania

ABSTRACT

I/O Interfaces for Microprocessor Systems, designed for use in analog data environments, should be made as powerful as reasonably possible to compensate for the "no-frills" micro-processor. The Hybrid Computer Interface (HCI) is a multi-function module composed of standard off-the-shelf components that augments the micro-processor's capabilities by performing not only the standard AD & DA conversions but also all four arithmetic, and compare operations. Its central element is a programmable hybrid arithmetic unit (HAU) which performs these operations directly on both analog and digital operands. The HCI derives its program from the micro-processor memory and operates either as a pre-processing peripheral or as a self-sufficient processor independent of the micro-processor.

INTRODUCTION

The performance-cost ratio of digital computers has improved significantly with the latest version—the μ processor. However, though sizable cost reductions in μp hardware have been realized, performance characteristics and user convenience are somewhat compromised. But users mating digital computers to analog oriented data environments have become accustomed to inconveniences not shared by those designing in the all digital domain.

Typically, when interfacing a digital computer to an analog data environment, the designer selects an AD/DA converter system which first funnels the digital equivalent of all analog operands into equations programmed for the digital computer. The digital results produced by the computer are then reverse funneled back to the converter system for distribution as analog display and control signals. Unfortunately, these bi-directional conversion operations are non-functional in that nothing useful is done to the data; however all conversion operations, be they with energy or with data, share the common problems of loss and inefficiency.

It is generally acknowledged that the computer system overpowers the typical application. With the more limited μp the opposite might be true; compromises in μp performance dictate that it be bolstered by more powerful peripherals so that the computational and processing burden can be shared. It is suggested that a Hybrid Computer Interface can share this burden when applying the μp to analog oriented data environments.

DESCRIPTION OF THE HYBRID COMPUTER INTERFACE (HCI)

The HCI is a programmable, multi-function module which operates directly on both analog and digital operands, and which produces analog and digital outputs. The HCI is like a digital computer in that it is programmable, executing instructions sequentially. Also the HCI contains an instruction decoder and a single arithmetic unit. The HCI differs from a digital computer by its use of both digital and analog memory. Since the HCI operates on analog operands directly, the language "separation" between it and the user is less than for a digital computer. It is more understandable than a digital computer because its logic is less complex and because programming is less difficult.

The HCI block diagram shown in Figure 1 consists of five functional components:

1. Instruction Decoder—decodes instruction words and generates timing pulses for their execution.
2. Dual Input Mux—samples the two designated analog input operands and routes them to the HAU.
3. Hybrid Arithmetic Unit (HAU)—performs the designated operations with the selected input operands.
4. Output Distributor—routes the HAU analog output to the designated analog memory output channel.
5. Discrete I/O—tests state of designated input discrete and generates designated discrete output pulse.

Each instruction word consists of four four-bit fields; the first field denotes the instruction to be executed, the next three specify input and output addresses. The second and third fields designate the two input operands selected by the Input Mux; the fourth field designates the Output Distributor channel for storing the HAU analog output, or the memory location for storing the HAU digital output. For

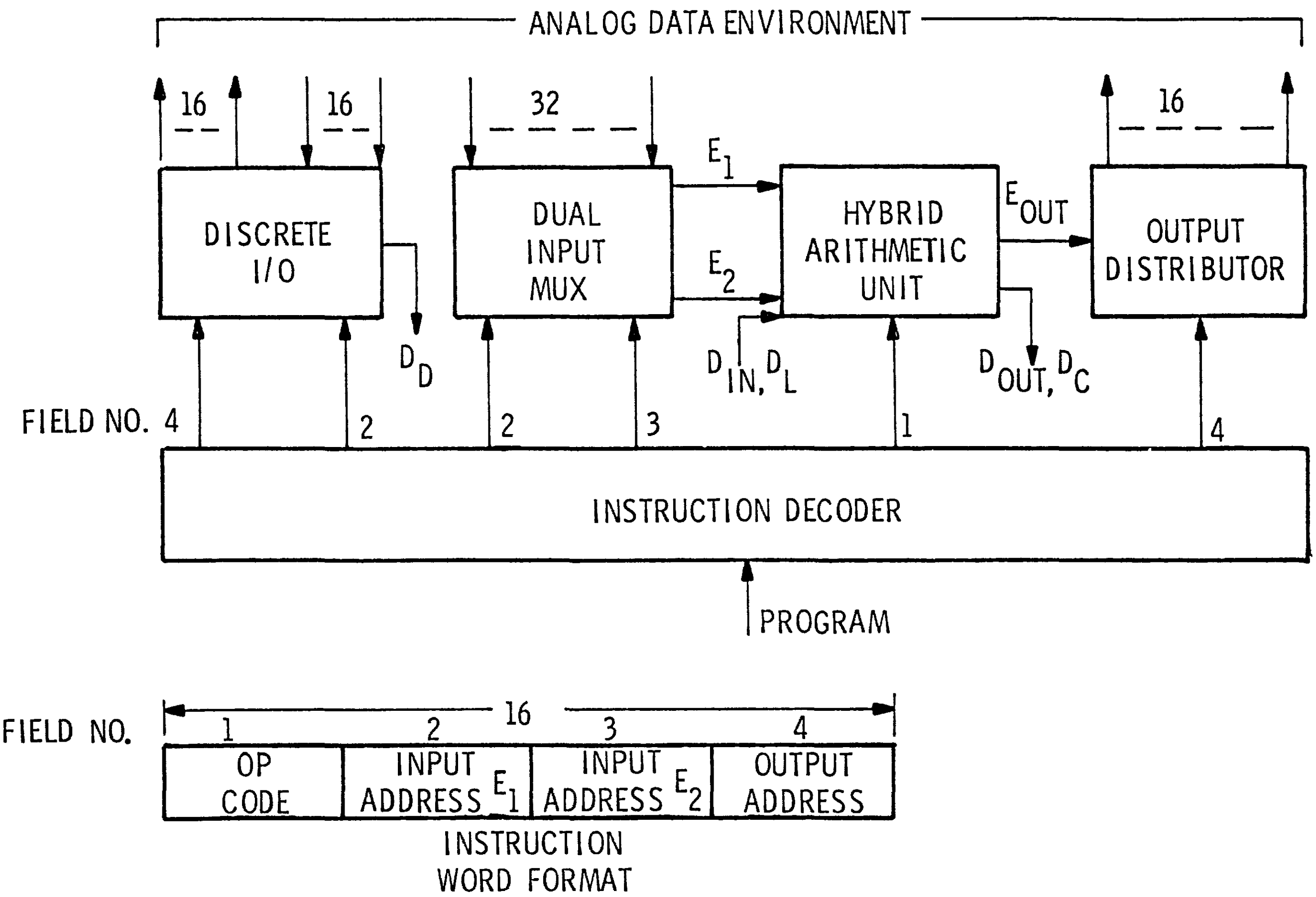


Figure 1—Block diagram hybrid computer interface

the discrete signals the second field designates the discrete input channel to be tested, the fourth field designates the discrete output channel to be pulsed.

In addition to the analog inputs selected by the Input Mux and the analog output routed to the Output Distributor, the HAU also receives input digital words, and produces an output digital word and a discrete flag bit.

Hybrid arithmetic unit

The HAU consists of a common set of interdependent analog and digital components that executes the designated instruction by input control signals from the Instruction Decoder. It does not consist of independent computing modules that rely on additional switching for routing of inputs and outputs.

Transfer function and instruction repertoire

The Hybrid Arithmetic Unit (HAU) performs the following three transfer functions using the two analog and one digital operands as inputs:

1. Analog$-E_{OUT}=E_1+E_2\times D_{IN}$
2. Digital$-D_{OUT}=E_1/E_2$
3. Discrete$-D_C=1$, $E_1>E_2\times D_{IN}$

These three basic functions are expanded into the instruction repertoire shown in Table I, eight instructions producing an analog output, two producing an n-bit digital word, and four producing a single comparison decision bit.

For several of the instructions the two input address fields are not variables. The input operands are 0 or +10 volts as implied by the op code.

HAU components

The HAU components that perform these instructions are shown in Figure 2. They are:

1. R/2R Ladder Network (n stages) sums bipolar analog inputs E_1 and E_2 producing an output null voltage current.

TABLE I—HCE Transfer Functions

OP CODE NO.	MNEMONIC				
		ANALOG TRANSFER FUNCTION (10 μS)			
0	ADA	ADDITION	E_{OUT} =	$E_1 + E_2$	$D_{IN} = +1$
1	ADD		=	$E_1 + D_{IN}$	$E_2 = +10V$
2	SBA	SUBTRACTION	=	$E_1 - E_2$	$D_{IN} = -1$
3	SBD		=	$E_1 - D_{IN}$	$E_2 = +10V$
4	MPY	MULTIPLICATION	=	$E_2 \times D_{IN}$	$E_1 = 0V$
5	DAC	DA CONVERSION	=	$E_2 \times D_{IN}$	$E_2 = +10V$
6	INP	INTEGRATION	=	$E_1 + E_2 \times D_{IN}$	$E_0 (t) = E_1 (t+1)$
7	INM		=	$E_1 - E_2 \times D_{IN}$	$D_{IN} = \Delta T$
		DIGITAL TRANSFER FUNCTION (20 μS)			
8	DVD	DIVISION	D_{OUT} =	E_1/E_2	
9	ADC	AD CONVERSION	=	E_1/E_2	$E_2 = +10V$
		COMPARISON TRANSFER FUNCTION (10 μS)			
A	CPA		D_C=1	$E_1 > E_2$	$D_{IN} = -1$
B	CPD			$E_1 > D_{IN}$	$E_2 = +10V$
C	CPP			$E_1 > E_2 \times D_{IN}$	
D	CPG			$E_1 >$ GND	$E_2 = 0V$
		LIMIT CHECK			
B	CPD			$D_{IN} < E_1 <$ $(D_{IN} + D_L)$	$E_2 = +10V$

2. SPDT switches—connects E_2 or $-E_2$ to each 2R input leg of the ladder, one additional switch routes the ladder output.
3. Digital Mux—selects one of four digital, coefficient values that produces a signed multiplication of the E_2 input.
4. H Register—stores the digital data input operand D_{IN} and the digital output ratio D_{OUT}.

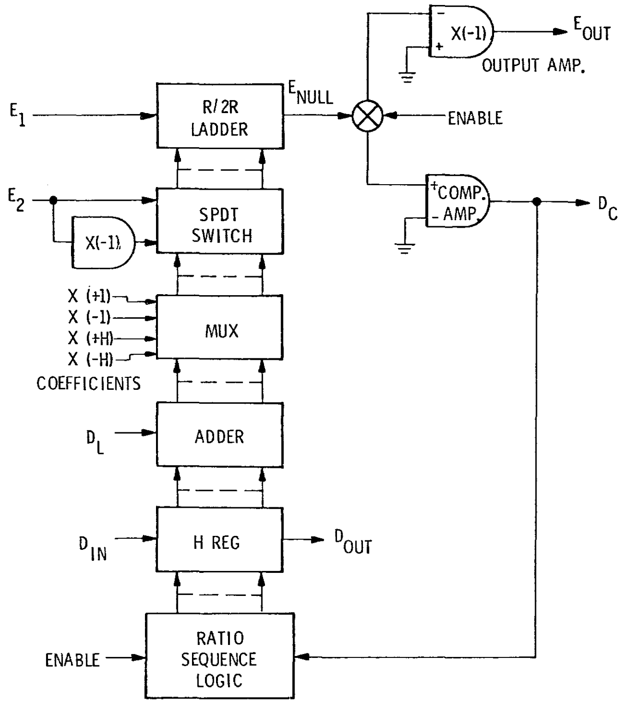


Figure 2—HAU functional components

5. Limit Check Adder—adds limit span value DL to low limit value stored in H register (for Hi-Lo limit checking).
6. Inverter Amplifier—inverts E_2 for input to switches, converts the ladder output current into the analog output voltage.
7. Comparator amplifier—amplifies null voltage at ladder output producing the discrete output for compare instructions.
8. Ratio Sequence Logic—executes the trial and error sequence for the divide and AD conversion instructions.

HAU operations

When the HAU adds two analog operands the +1 coefficient input of the Digital Mux is enabled causing all the ladder input switches to select the E_2 input. For subtraction the −1 coefficient input of the Digital Mux is enabled causing the ladder input switches to select the $-E_2$ input. For both instructions the contents of the H Register do not enter into the operations. However, when analog and digital operands are added or subtracted, the contents of the H Register, or its complement, are gated thru the Digital Mux by the +H, or −H coefficient input to produce a signed multiplication of the E_2 operand. For all analog transfer function instructions the ladder current output is connected to the output inverter amplifier thru the SPDT switch, thereby producing a proportional voltage E_{OUT}.

For digital and comparison transfer functions the ladder output null voltage is connected to the high impedance input of the comparator amplifier. Both of these transfer functions are subtractive in nature; the digital transfer function drives the ladder output voltage to a null in deriving the ratio of the E_1 and E_2 operands, whereas the comparison transfer function creates a null voltage equal to the difference of these operands. The divide instruction also performs the inverse hyperbolic operation of 1/X where the numerator constant 1 is represented by a fixed analog voltage of 0.1 volt for a two decade coverage of 0.1 volt to 10 volts for the divisor input. This range for division extends the standard comparator amplifier and requires additional sensitivity for the full two decade range. (National LM111 comparator)

TABLE II

Analog Volts	Digital	
	Sign	Mag
+9.92	1	177
+5.00	1	100
+0.08	1	001
+0.00	1	000
−0.00	0	177
−0.08	0	176
−5.00	0	077
−9.92	0	000

For limit check instructions two compare operations are executed; the first operation checks that the E_1 operand is above the low limit stored in the H Register. The second operation checks that the E_1 operand is below the sum of the low limit and the Hi-Lo Limit span value DL. The positive sense of Dc is tested for the low limit check, the negative sense of Dc is tested for the high limit check.

All HAU analog inputs and outputs have a +/−10 volt range. The analog digital scaling for signed eight bit digital words is illustrated in Table II.

The analog and digital results produced by all HAU operations are arithmetically consistent over all four quadrants inherently, without need for additional flag instructions or hardware. However, for the digital transfer function divide instruction the sign of the E_2 divisor/operand is first tested in order to establish the Dc or negative Dc sense of the comparator amplifier for the trial and error sequence to follow. Table III lists the quotients produced for the four quadrants. When E_2 is positive the negative Dc sense causes each trial stage of the H Register to be reset;

TABLE III

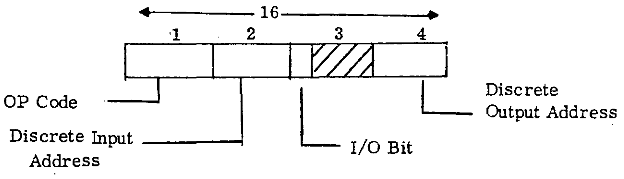
	D_{OUT}			
		+10V		−10V
E_2=E_1	S	Mag	S	Mag
+9.92	1	177	0	000
+5.00	1	100	0	077
+0.00	1	000	0	177
−5.00	0	077	1	100
−9.92	0	000	1	177

when negative the positive DC sense causes it to be reset. By testing the ladder null output prior to initiating the trial and error sequence when the H Register is reset, the effect is identical to multiplying the E_2 by +1 thereby permitting a polarity check on the E_2 operand. This test always assumes that E_2 is larger than E_1, as it must be; if not a null is never attained, producing a full scale digital output.

The timed sequence of operations performed by the HAU for the three transfer function instructions are illustrated in Figure 3.

Discrete I/O

The mux for discrete inputs selects the designated signal for input to the μp where its state is interpreted appropriately. The output distributor for discrete signals performs the reverse function; the designated discrete output channel is activated to perform its user assigned function. The common format for these two discrete instructions as shown below, is consistent with that for the three HAU transfer functions.



THE HCI AS A μp PERIPHERAL

By using the HCI as a peripheral to a μP, two complementary arithmetic units are brought to bear on control/computational problems. The problem is divided to allow the HCI to perform preprocessing computations and editing. The μP provides overall direction and coordination and performs the program storage function. Figure 4 illustrates the I/O ports required to establish instruction and data communications between the HCI and μp. Two output ports of eight bits each are assigned to the 16 bit instruction word. Two ports, one output and one input, handle the transfer of the digital data words and discrete signals.

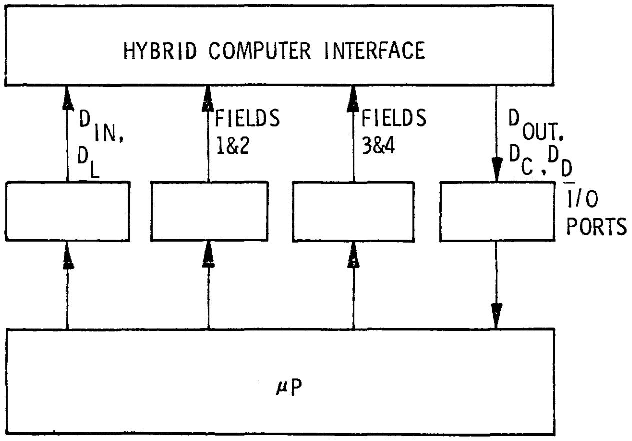
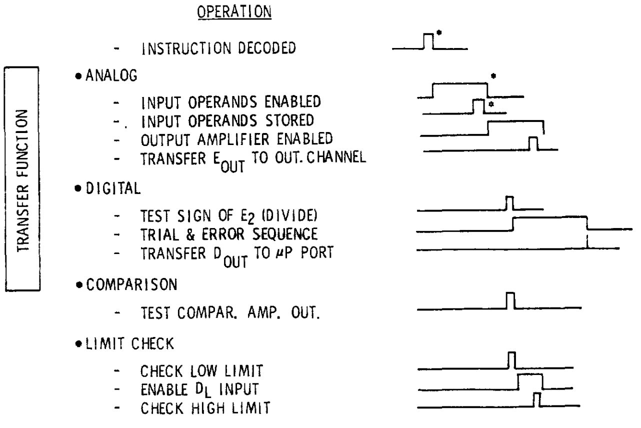


Figure 4—μP/HCI interface

Figure 5 illustrates the sequence of operations executed by the μP for servicing the HCI. Instruction execution times are reduced by 20-30 percent by the addition of a buffer register for the instruction word, allowing the μP to fetch the next instruction concurrent with HCI operations. However, when the HCI signals a branch in the program the instruction in the output port must first be replaced with the designated instruction. The HCI waits for this fetch. The period defining the computational cycle is established by an external clock which interrupts the μP at the beginning of each cycle.

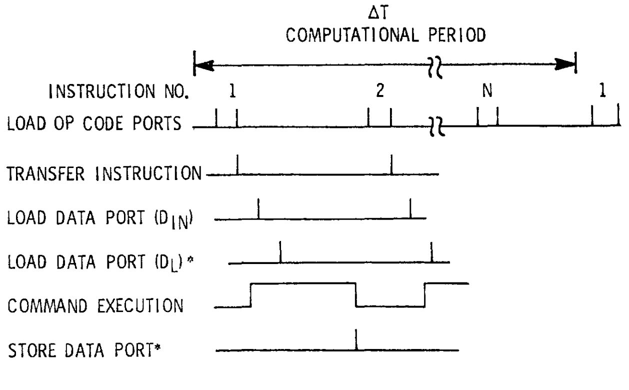
THE HCI AS AN INDEPENDENT COMPUTER

The HCI as described includes four of the five components of a complete computer: it lacks only digital memory. By adding the memory component the HCI operates independently of the μP. This capability might prove very



*COMMON TO ALL FUNCTIONS)

Figure 3—HCI transfer function operations



*EXECUTED ONLY FOR DIGITAL OR COMPARISON FUNCTIONS

Figure 5—μP operations

useful in distributed computer systems where a central computer coordinates several remote, semi-autonomous intelligent terminals. A distributed system application may not warrant the use of a μp at each terminal because of the simplicity of the processing and computations. However, the services of a computer for special support would be available as needed via the communications links.

This memory addition to the HCI is augmented with appropriate logic to permit program branching and looping. Also the digital transfer function instructions that produce digital outputs for storage, and comparison transfer function and discrete test instructions that enable the program sequence to be altered, must be accommodated.

Inasmuch as the output address field, by its four bit length, limits the range of memory locations to be directly addressed, an intermediate memory permitting indirect addressing is employed. The output address field of the instruction fetches the final memory address from this intermediate memory. This intermediate memory contains 16 locations in each of three sections, one each for the digital, comparison and discrete test instruction types.

In a dedicated application this independent memory for the HCI would be mixed ROM and RAM. The ROM memory would store the fixed program while the RAM would hold digital results produced by the HCI and used in subsequent instructions.

PROGRAM ROUTINES

In addition to the basic instruction repertoire executed by the HCI, a sequence of these instructions can be grouped to perform the next higher order computations such as squaring, square root, function generation (non-linear sensor), differentiation, intercept-slope correction, as well as complete programs for 3 mode process control algorithms. The following example demonstrates how such sequences are programmed for the HCI.

Squaring

The flow chart for the four operations constituting the squaring sequence is shown in Figure 6. The first two operations are preliminary in that the input is first sampled and held on an output memory channel which serves as the input for the next operations. The sample and hold operations though not necessary eliminate errors due to sampling skew. The AD operation converts the input variable to its digital equivalent and the final multiply operation produces the product of the analog input and the contents of the H register.

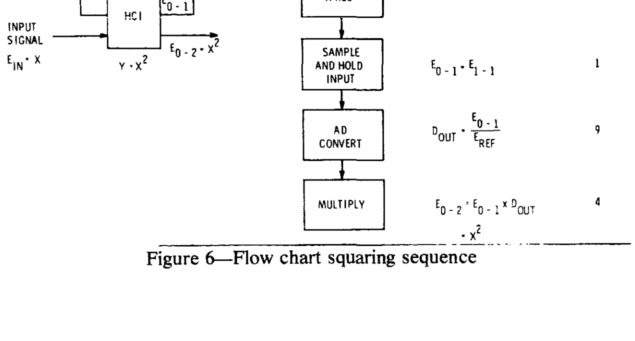


Figure 6—Flow chart squaring sequence

Square root

The square root sequence employs an iterative technique based on the equation:

$$Y_{i+1}=\frac{1}{2}\left(Y_i+\frac{X}{Y_i}\right)$$

$$=Y_i-\frac{Y_i-\frac{X}{Y_i}}{2}$$

where the ith value had been produced previously during the i-1th computational cycle. The square root flow chart, shown in Figure 7, consists of ten operations. The first four operations tests that the input variable is positive and assigns the initial trial value as a function of the input value, in order to hasten the convergence. The next seven operations form the iterative loop which calculates the delta change in the next trial value. When the delta change is within the + and − tolerance value, the limit check operation causes an exit and produces the calculated square root value at an output channel. A total of four analog output channels and six analog input channels are used during the square root sequence. However, eight of these channels serve in a scratchpad capacity in that they can be used for other such sequences. In order to eliminate the necessity of using addressable input/output channels for scratchpad purposes an additional set of committed internal input/output channels are provided. The instruction address fields activate these channels instead of the user input/output channels, by command from the μP.

Differentiation

The differentiation sequence shown in Figure 8 consists of seven operations. The first operation derives the ΔE numerator, the next two convert the digital ΔT value stored in the μP memory into its analog equivalent. The division operation produces the derivative in digital form, the following D-A operation presents the derivative in analog form. The final operation stores the current operand value for use in the next computation cycle. The ΔT value is proportional to the computational period.

Hardware diagnostic

The performance of the HCI is checked by diagnostic programs since it can analyze the results that it produces. These programs test that all input and output channels as

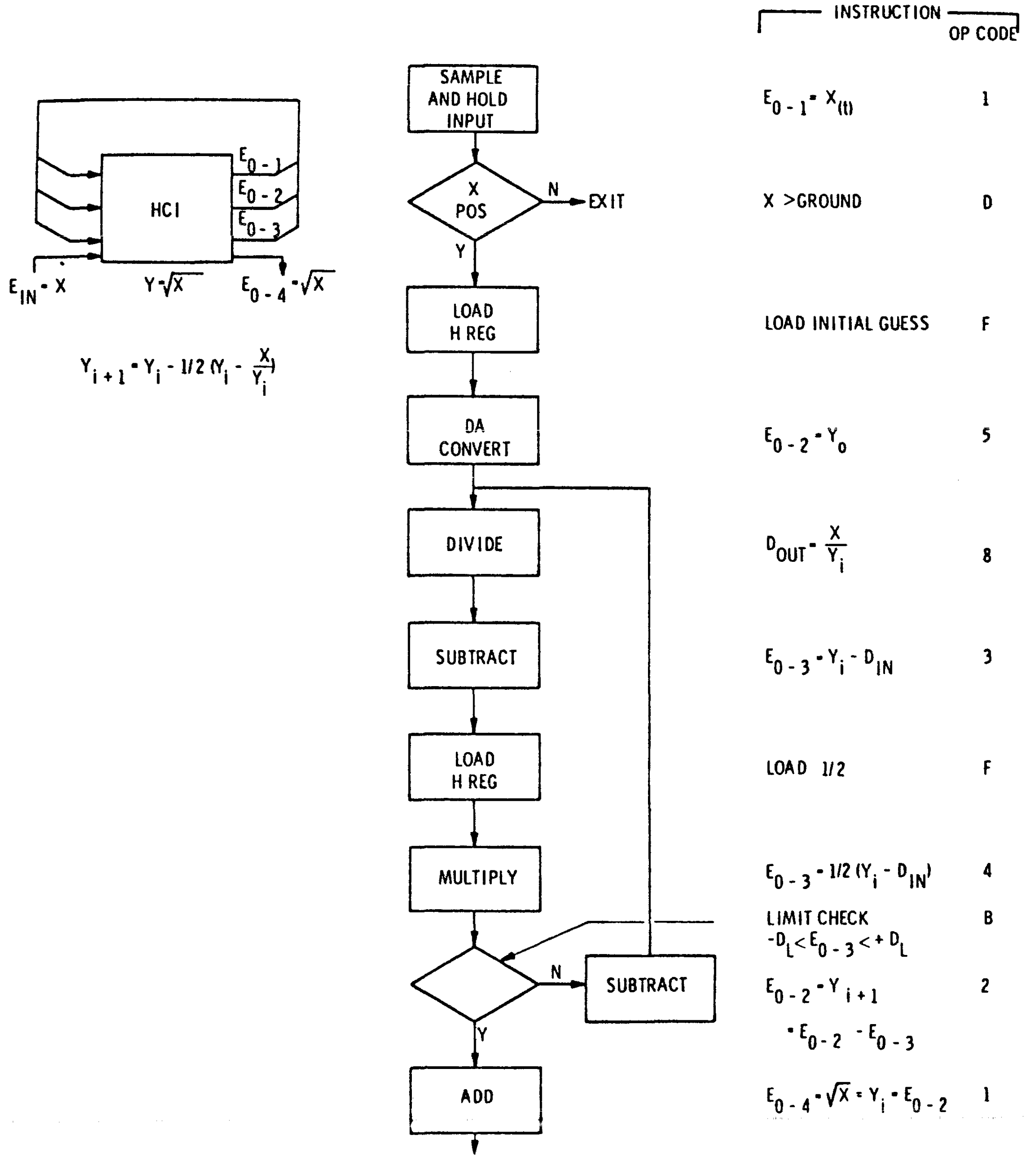


Figure 7—Flow chart square root sequence

well as the components of the HAU are functioning correctly. Typical of these diagnostic programs is the tracking program shown in Figure 9. This program produces an output which tracks an input triangular waveform by means of compare instructions which increment or decrement the contents of the DL register to maintain a null. The subtract instruction indicates the tracking error between the input and output signals. The execution time for this program is 30 microseconds allowing an eight bit HCI to track a ramp of three volts per millisecond within one bit.

SUPPORT SOFTWARE

No software has been generated to reduce the HCI programming problem. However because of the small number of instruction types the programming problem is not severe. Most HCI programs are relatively short because of the three-address instruction format and because the HCI operations are inherently more functional. Therefore, a viable solution is to machine code the program for direct entry into the μP memory.

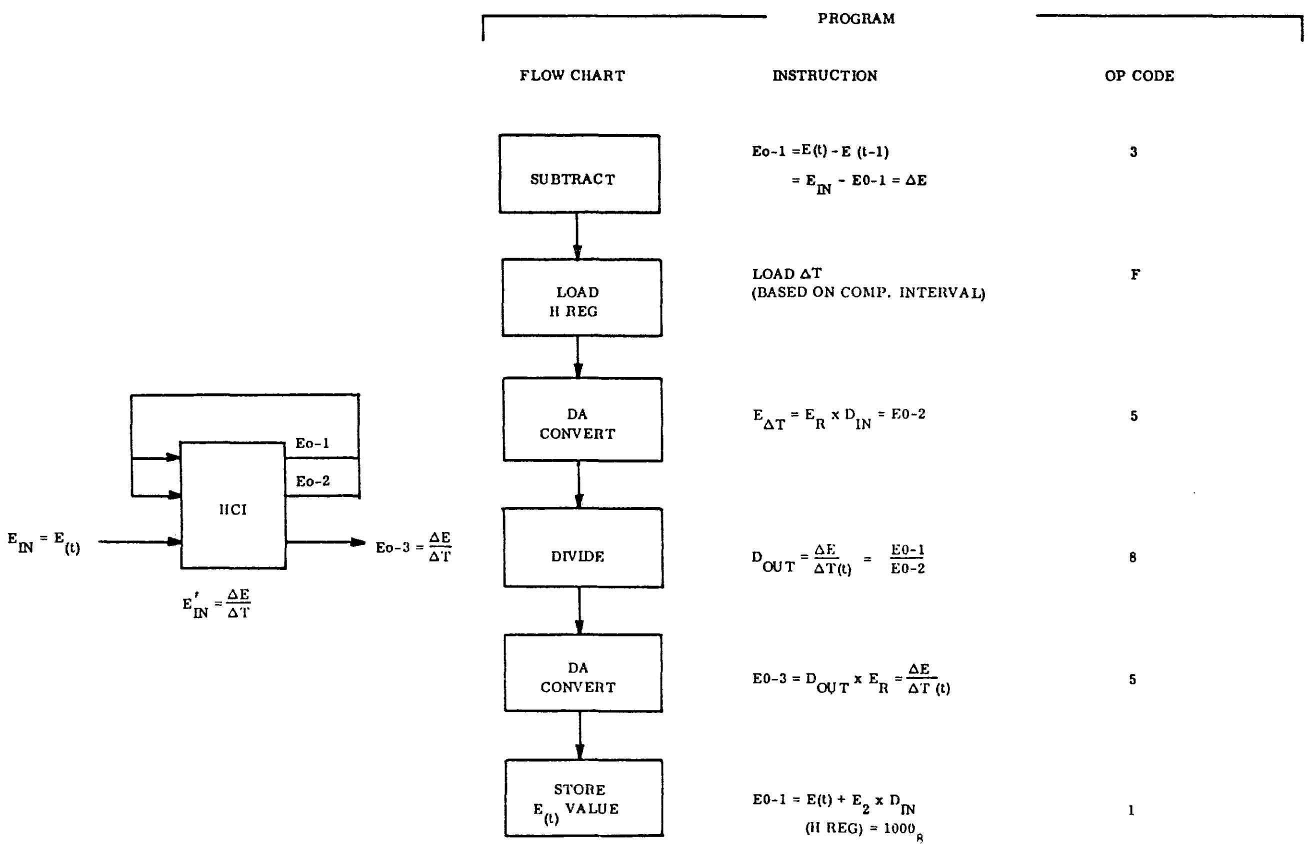

PROGRAM
FLOW CHART
INSTRUCTION
OP CODE
SUBTRACT
$Eo-1 = E(t) - E(t-1)$
$= E_{IN} - E0-1 = \Delta E$
3
LOAD H REG
LOAD ΔT (BASED ON COMP. INTERVAL)
F
DA CONVERT
$E_{\Delta T} = E_R \times D_{IN} = E0-2$
5
DIVIDE
$D_{OUT} = \frac{\Delta E}{\Delta T(t)} = \frac{E0-1}{E0-2}$
8
DA CONVERT
$E0-3 = D_{OUT} \times E_R = \frac{\Delta E}{\Delta T}(t)$
5
STORE $E_{(t)}$ VALUE
$E0-1 = E(t) + E_2 \times D_{IN}$
$(H REG) = 1000_8$
1
HCI
Eo-1
Eo-2
$E_{IN} = E_{(t)}$
$Eo-3 = \frac{\Delta E}{\Delta T}$
$E'_{IN} = \frac{\Delta E}{\Delta T}$

Figure 8—Flow chart differentiation sequence

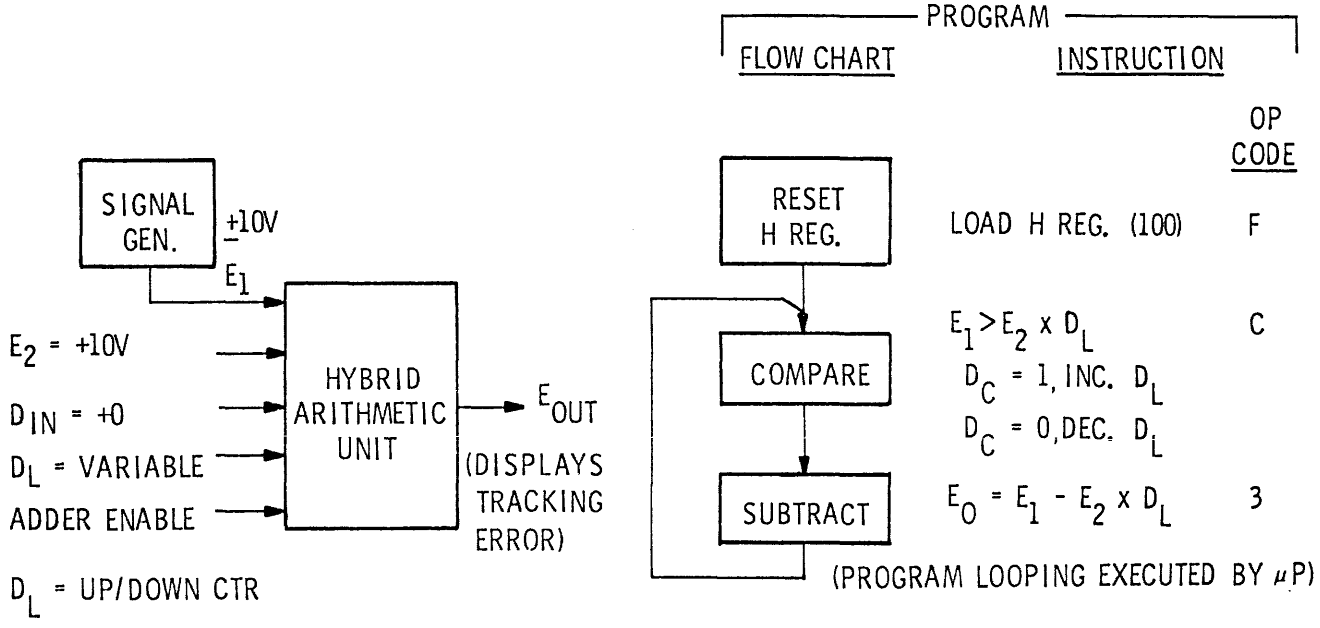


Figure 9—Tracking diagnostic program

The main μP program fetches these HCI instructions from its memory and initiates each computational cycle by an interrupt from an independent timing source. These and other such housekeeping type tasks are handled by routines programmed for the μP.

HARDWARE COMPONENTS

The component complement for the HCI consists of off-the-shelf hardware. The Dual Input Mux and Output Distributor is composed of analog devices such as gates, and sample and hold amplifiers. The HAU consists of a 10K R/2R ladder, inverting mode op amplifiers, SPDT analog switches (RON<40Ω), a comparator amplifier, and variety of 7400 TTL digital ICs. The instruction Decoder and the Discrete I/O section consists entirely of 7400 TTL digital ICs.

For an eight bit HAU the number of ICs of all types total approximately 90. The cost of these ICs totals approximately $700. Over half of this expense is for the sample and hold amplifiers. Using a less expensive sample and hold amplifier reduces the IC cost to $500 (non production quantities). The standard voltages of +5, +15, and −15 volts are required. Total power dissipation is less than 2 watts.

PM/II—Multiprocessor oriented byte-sliced LSI processor modules

by MARIO TOKORO
Keio University
Yokohama, Japan

and

TAISUKE WATANABE, KATSURA KAWAKAMI, JUN SUGANO and KATSUHIKO NODA
Matsushita Research Institute Tokyo, Inc.
Kawasaki, Japan

ABSTRACT

This paper is concerned with the design and implementation of LSI processor modules named PM/II. The basic concepts involved in designing PM/II modules are (1) to provide maximum flexibilities with the smallest kinds of modules, (2) to construct a wide variety of computers from micro to midi and/or maxi where the total number of components are at a minimum, and (3) to realize PM/II in "the process free design" which avails itself of the idea that basic architecture is invariant even when implemented in any semiconductor process. PM/II are now actually being produced in CMOS.

At present the PM/II modules provide an Executor (EX), a Sequence Controller (SC), and an I/O Controller (IOC). They realize three dimensional extensibilities, i.e., (1) for functions through functional decomposition, (2) for varying bit-width of operation by byte-slicing, and (3) for multiprocessor configurations by using a sophisticated inter-processor communication function. A multiprocessor system composed of PM/II's is already running, proving the validity of the design concept.

INTRODUCTION

Recently rapid progress in semiconductor technology has put high density and low power-delay LSI products to practical use. In the case of density, it has even been said that for semiconductor processes there exist no problems even when the demand for large amounts of gates per chip are taken into account by computer architects. While various kinds of microprocessors have been consumed in great quantity, to a large extent the LSI-zation of mini and office computers has prevailed. LSI hardware has the potential to change the architecture of middle and/or large scale computers in the near future.

Several approaches to the development of LSIs for middle and/or large scale computers are immediately apparent. One approach is to make a chip which contains the total functions of an individual CPU. Yet this has the disadvantages of lack of flexibility; (1) when design errors occur, (2) when design changes in technological development occur, (3) in system configurations and applications, (4) in testing chips, and (5) in the trade off between pins, yield and power dissipation. So far only 8 to 16 bit simple microprocessors have been realized due to these disadvantages.

If the principle of mass-production of a few LSIs is hereafter held to, it seems profitable for manufacturers to develop as few kinds of chips as possible from which a variety of computers—ranging from micro to midi and/or maxi, can be composed. The straightforward approach to this problem seems to be the use of the concept of processor modules, which are sometimes called computing modules or computer modules. This approach attempts to develop minimum kinds of LSI processor modules with maximum capabilities in each, so that any digital system can be composed of the smallest number of them. Some research on processor modules has previously been accomplished,[1-4] with the one realization being bit-sliced microprocessors such as MMI6701, Intel3000, AMD2900, Macrologic, SBP0400, and M10800. The performance and the flexibility of bit-sliced microprocessors in applications has been proved. However, a number of MSI modules and/or SSI modules are indispensable when composing an applications system. This is caused by the following: (1) bit-sliced microprocessors have been developed to extend the capability of ALU's which still require the design and manufacture of many support modules, (2) they do not get rid of the family series of MSI's, and (3) they have been developed with excessive restrictions on the number of gates per chip and on the number of pins. Yet processor modules should be essentially designed through analyzing various computers and their applications and decomposing into functions for those applications, and then synthesizing the functions into combinations of modules in order not only to gain maximum flexibility in system configuration, but also to be able to compose systems with a minimum number of components.

PM/II is a series of processor modules designed with the above considerations taken into account. The authors have classified the applicability, (i.e., the extensibilities of the capability of processor modules) into the following three criteria: (1) extensibility of functions, (2) extensibility of bit-width of operations, and (3) extensibility of total system performance. The first means the availability for the addition or change of functions such as a multiplier/divider to a system. In order to satisfy this requirement, PM/II is functionally decomposed. For the second requirement, the execution module of PM/II is designed byte-sliced. And for the last one, a powerful function available for multiprocessor configuration is attached, since multiprocessor systems have been becoming important in realizing high performance systems.

In addition, the architects have proposed "the process-free design" against various restrictions from semiconductor technology. That is the methodology of design in which the basic structure is invariant in spite of the change of semiconductor processes, and in which maximum performance can be obtained with a single process. CMOS process has now been taken for the actual development. However, PM/II modules will have the same capabilities in other technologies. Moreover, they will give better performance in other technologies such as Shottky TTL, and/or in improved technologies expected in the future.

The three basic modules in PM/II have been designed and are being processed. They are a Byte-Sliced Executor (EX), a Sequence Controller (SC), and an I/O Controller (IOC). At the present, a multiprocessor system with three processors including breadboards of PM/II is running, satisfying the design purpose and proving the validity of the design concept. The rest of this paper is devoted to the description of the design concept, architecture, application, and evaluation of PM/II.

DESIGN CONCEPT

A CPU can be functionally decomposed into a number of functional blocks such as several data processing blocks, a sequence control block, an I/O interface block, a control memory, and a main memory. Such a method of decomposition, e.g., functional decomposition, decreases the number of connections among blocks, and this leads to a decrease of pin numbers of a chip when producing an LSI module. It is also preferable from the point of semiconductor technology to have the work load, processing speed, and chip size of blocks balanced among the blocks, unless the flexibility of each block is lost.

On designing processor modules, the most principally used blocks have been newly designed, e.g., ALU, sequence control block, and I/O interface block, while control and main memory have not needed to be redesigned.

The ALU may vary its operations and bit width as a function of its performance objectives. Bit-slicing is applied to the module with the powerful instruction set for composing the ALU, so that functional flexibility and semiconductor technological balance is satisfied.

A sequence control block calculates the next address of its control memory. The address space is generally limited to a fairly small size, and dynamic microprogramming techniques are feasibly employed for large microprograms. This results in a sequence control module with a powerful addressing calculation capability, which is not bit-sliced.

Data processing blocks are connected to buses. Since bus configuration and the control method varies according to the application, it is profitable to separate I/O control functions from data processing blocks to form an I/O control module. An I/O control module has been designed for an asynchronous bus. The module is composed of relatively few gates, so it is not difficult to redesign to meet other buses.

Based on the above considerations, designs of these three modules have been carried out from the following view points, which are proposed to construct a figure of merit of processor modules, e.g., (1) processing speed, (2) configurational flexibility, (3) performance/cost of control memory, and (4) availability for multiprocessor systems.

Processing speed

Pipelining

In order to obtain high speed processing capabilities, instruction fetching and the execution of modules are overlapped to form pipelining. This technique is very effective when the cycle time of the control memory and the execution time of modules are balanced. Therefore it agrees with the process free design method.

Conditional jump instructions take extra cycles in pipelining, which may cause inefficiencies in execution time and control memory utilization. To minimize these inefficiencies, a signal named "Wait" has been introduced which suspends the execution until a designated condition is satisfied. This signal is available for quick responses in process synchronization which are frequently encountered by I/O instructions. On the other hand conditional branching routines are essentially left conditional. Microinterruption is provided to recover from a dead lock.

Inside the module EX, a pipeline technique is also adopted for data transmissions into and out of the ALU. Three-address instructions for EX are employed for this reason.

Two data ports

The data transmission delay between LSI chips is several times longer than that inside a chip, and this may limit the rate of data throughput. In order to gain a high rate of data throughput, two independent parallel data ports are provided, which largely affect the bit width of operations in an EX.

Configurational flexibility

Byte sliced

The width of parallel operations in an EX is designed to be eight bits, or a byte, which is generally the smallest unit of data. While this decision has been made to balance with the number of pins and the size of the chip area of an SC, this has satisfied the requirement of extensibility for bit-width of operations and consequently has brought about configurational flexibility.

Two data ports

An EX, provided with its two independent data ports, increases configurational flexibility of the PM/II. For example, one port is assigned for a system bus and the other for the processor bus which connects local memory, multiplier/divider, and other devices. Another scheme is where one port operates asynchronously and the other synchronously.

External modification

SC is designed to have an 8-bit External Address Modifier (XAM) input port, which enables it to modify the next address of the control memory dynamically. By connecting a PLA decoder or such a device to XAM, quick response for parallel branching is easily attained which is significantly effective for real time control and language emulation.

Addresses of general registers in EX can be designated by data. This enables indirect register designation which is essential to emulation.

Performance/cost of control memory

Module instructions

Each module has its own instruction set independent from the others, which is partially encoded. The microinstruction format becomes wide when modules are combined to compose a system, so that the control memory cost becomes quite expensive. However, when constructing a low price processor, one is not required to prepare a full parallel control of resources in the system. In order to meet the requirement of parallel control with reasonable cost, a "Chip Enable" bit is attached to the instruction of each module. If a chip is not selected, a "No Operation" instruction is generated inside. This bit very easily and directly enables the overlap of fields among modules.

Availability for multiprocessor system

Communication synchronization

Usually, a "Test and Set" instruction for a flag is used to synchronize communications between processors. However it is more efficient to provide a register with a flag, which is set simultaneously when a message is transferred to the register if and only if the flag is off. This facility is adopted and implemented in an EX. A P-Register (PR), which is an 8-bit register in an EX, receives information from other processors. The content of the PR is not affected until a flag named P-Register Full (PRF) in the IOC is reset by an instruction.

Bus system

The IOC is designed to interface the EX with an asynchronous bus system. The asynchronous bus system is very general and extensible in multiprocessor construction. In this bus system, processors, memories, and the other devices are uniquely addressed, and the communications between them are carried out by handshaking. A distinctive feature of this bus system is its special signal (RSYNC) which supports the communications between processors.

SPECIFICATIONS

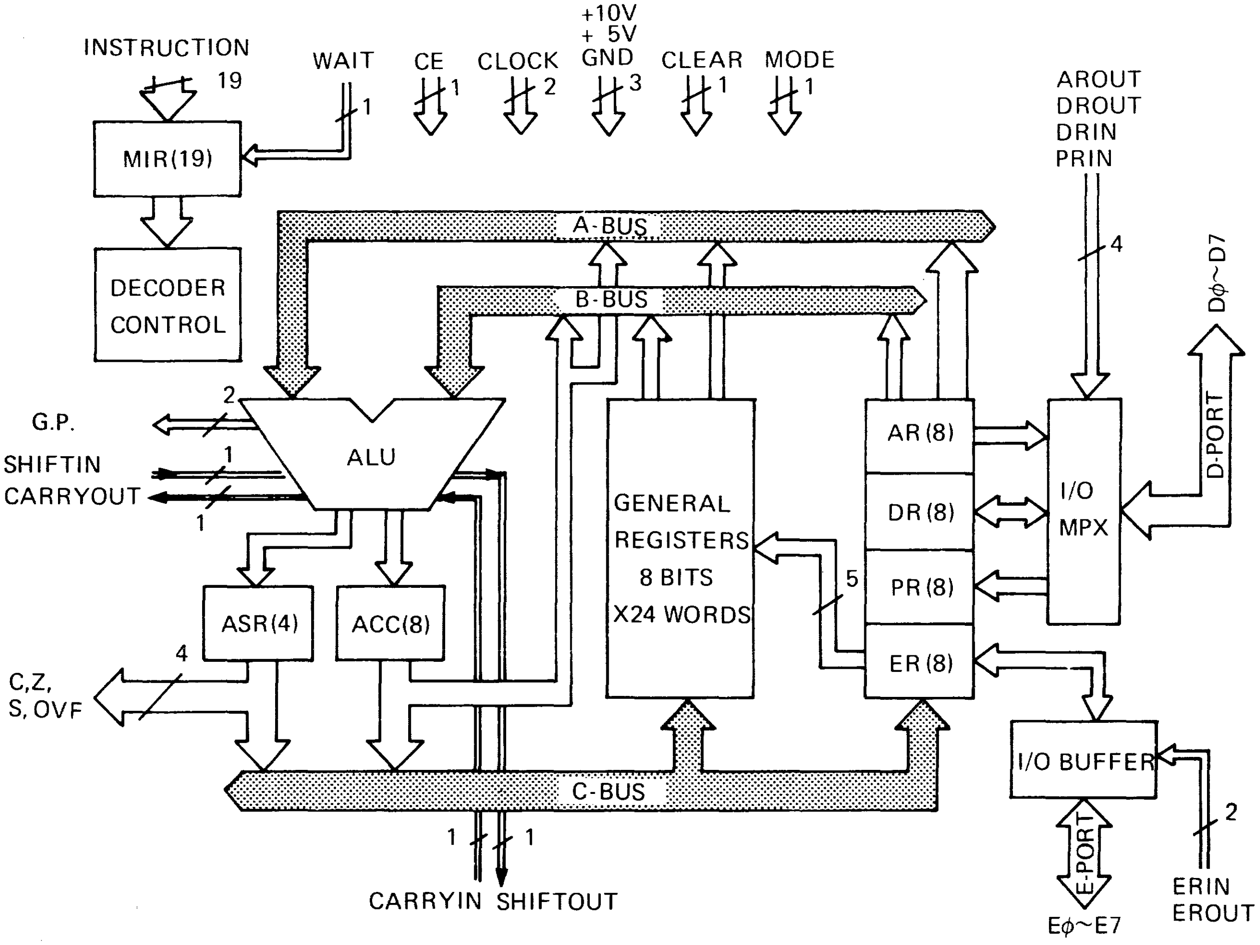
PM/II is being implemented by the CMOS process. The delay of logic elements has been confirmed on test chips, showing for example that the delays of an inverter, a 2-input NAND/NOR, and an I/O pin are at most 5, 10, and 30 nsec, respectively. The specifications of each module are shown in Table I. The maximum number of the pins of the LSI package are 60, and more than 4500 transistors are included in SC and EX chips. Cycle times of 240 and 280 nsec are expected in 16 and 32-bit system operations, respectively.

Executor

The block diagram and instruction set of the EX are shown in Figure 1. Four types of instructions are designed to obtain smooth pipeline processing. The first type of instructions, which are the most typical ones, consists of four fields, ALU Operation (AO), A, B and C. The contents of General Registers or I/O Registers designated by A and B are transferred to the ALU through an A and B-bus. Concurrently the contents of Accumulator (ACC), which holds the results of the previous operation, are stored into the register designated by the C field through the C-bus. Six kinds of operations, ADD, SUB, AND, OR, XOR, and LAJ, are performed in ALU. The LAJ (Load Adjust)

TABLE I – Specifications

	EX	SC	IOC
PACKAGE	60 PINS	60 PINS	24 PINS
ELEMENT	4500 Tr.	4500 Tr	1000 Tr.
SIGNAL LEVEL	TTL COMPATIBLE	TTL COMPATIBLE	TTL COMPATIBLE
BIT WIDTH	8 BITS	12 BITS	
INSTRUCTION WIDTH	19 BITS	17 BITS	3 BITS
INSTRUCTION SET	14	8	3
CYCLE TIME	240ns. min.	240ns. min.	240ns. min.
REGISTER	24 WORDS	16 WORDS	
POWER SUPPLY	10V, 5V, 0V	10V, 5V, 0V	10V, 5V, 0V
CLOCK PHASE	2 PHASES	2 PHASES	2 PHASES
CLOCK PERIOD	120n SEC	120n SEC	120n SEC



AO	A		B	C
E		I		C
SR		NU	B	C
ST		NU		C
4	2	3	5	5

A : SOURCE FIELD
B : SOURCE FIELD
C : DESTINATION FIELD

E : EMIT
I : IMMEDIATE DATA
SR: SHIFT/ROTATE
ST: STORE ACC/STR

AO: ALU OPERATION

(ADD. SUB. AND OR. XOR. LAJ)

Figure 1—Block diagram and instruction formats of EX

instruction is effectively used for the decimal operation, which sets "6" for every four bits of the ACC as the result, if separate carries of each 4-bit addition are generated.

The second type of instruction sets immediate data into a register designated by the C field. The third type is designed for shift and rotate operations, having one operand. The fourth type, which specifies the data transfer from ACC or the ALU Status Register (ASR) to a destination register, has an 8-bit Not Used field, which is efficiently used when combined with the SC instructions.

These instructions are activated when a Chip Select (CS) bit is on. If it is off, the operation is not performed, but carry and shift signals are propagated through the EX.

Two bi-directional transmission ports and four 8-bit I/O Registers are provided. Information stored in AR, DR, and PR are transferred into and out of the chip through D-port according to the port control signals, while ER is connected to the E-port. Using AR and DR for the address and data registers, respectively, D-port permits simple connections to an external bus. PR provides the function of receiving messages from other processors.

Four port control signals AROUT, DROUT, DRIN, and PRIN, are generated by the IOC. These signals are used for bus driver control so that the complicated bus interface circuitry is eliminated.

The low order 5 bits of ER can be used as an address pointer of the general registers. The ASR contains the status of the result of ALU operations, i.e., Carry, Overflow, Zero, and Sign. The ASR is connected not only to the corresponding pins but also to the C-bus, so that the content of the ASR can be stored into general registers. This enables the quick restoration of status which is indispensable to interrupt processing and emulation.

In order to connect several EX's in cascade to make a powerful ALU, four pins CARRY IN, CARRY OUT, SRIN, and SROUT are provided. The status of the operations are represented by the ASR of the most significant EX and the Zero bits of all the ASR's. A mode control input is used to specify the most significant EX which manipulates the carry and shift control. The two signals, Carry Generation (G) and Carry Propagation (P), are prepared for external look ahead carry generations.

Sequence controller

The block diagram and instruction set of the SC module are illustrated in Figure 2. The length of the Control Memory Address Register (CMAR) has been determined to be 12-bits wide, so that the next address calculation time is almost equal to that of the parallel data operation in two or four cascaded EX's. Maximum size of control memory is 4K words which may be enough for ordinary applications, however, extension of size or dynamic microprogramming are easily attained with a few external IC's.

The SC has five types of instructions. The first is a jump absolute/jump relative instruction with a 12-bit displacement field. The large displacement field not only releases programmers from the anxiety of overbound addresses but also achieves high speed processing. The second type of instruction sets the immediate data into the Stack or Counter. The third one is the conditional jump relative. Branching occurs if the contents of the T/F field are equal to the contents of a bit of the Control Status Register (CSR) or XAM as designated by the BP. The fourth one is provided for interruptions, subroutine calls, and other uses. The contents of the register designated by the RS-field are transferred through the SC-bus to the register designated by the RD-field. If RS and RD are the Stack and CMAR, respectively, this instruction effects a so-called "return" from subroutine or interrupt process. Control memory addresses can also be dynamically modified by this instruction, if the XAM and CMAR are pointed to by RS and RD, respectively. Then each bit of the CSR is set or reset by the fifth type of instruction.

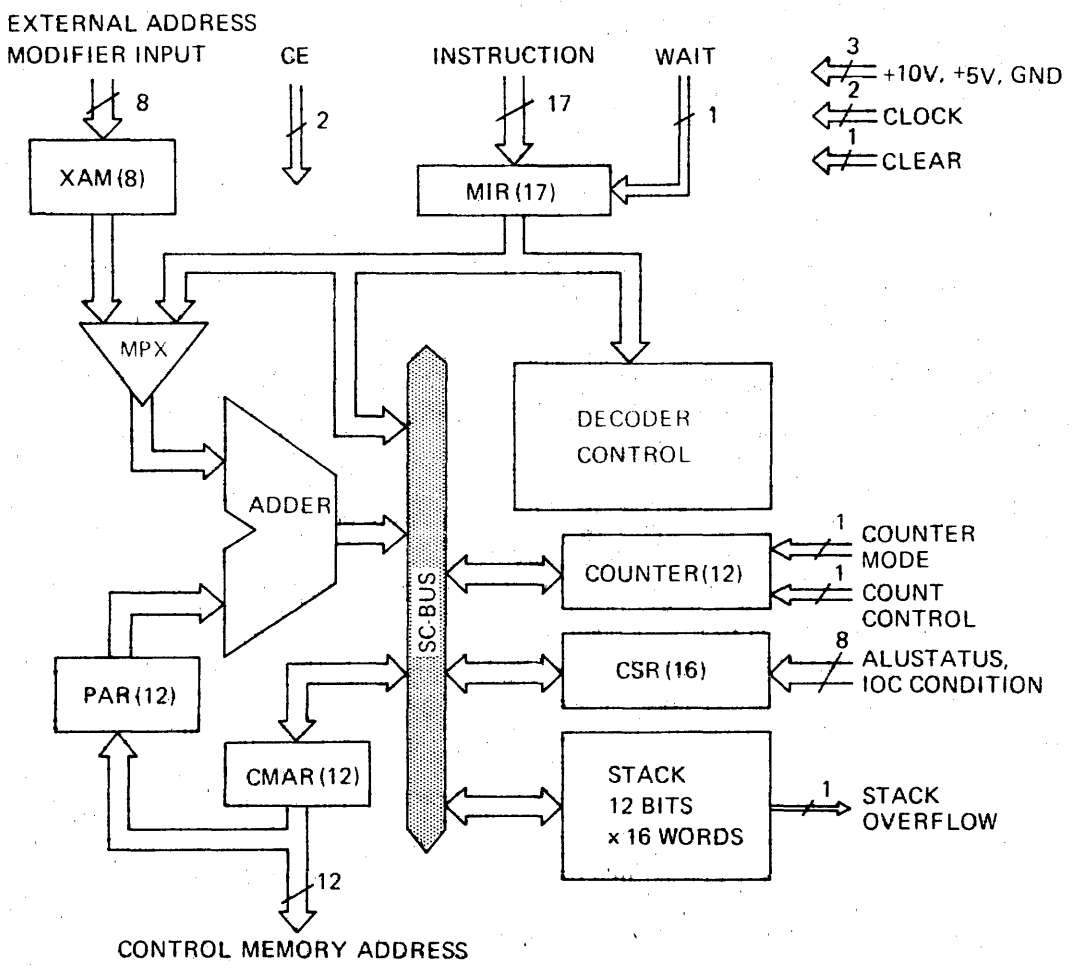
These instructions are activated by the two CE's which are wired-ORed inside. If they both are off, the SC just increments its CMAR by one.

As shown in Figure 3, the instruction access and the calculation of the next address, which is done by adding the contents of the Pipeline Address Register (PAR) and the D-field of the microinstruction register, are performed simultaneously. Therefore, the instruction stored in the next address of a branch instruction is always executed. A sixteen-word Stack is prepared so that sixteen-level subroutine nesting or eight-level interruptions are available. A 12-bit Counter performs in two modes as specified by the Counter Mode. In the first mode it is decremented by the pulse from the external pin. If a bit of control memory output is connected to the pin, it acts as a loop counter. In the second mode, the contents of it are decremented at each machine cycle. Therefore, it plays an interval timer, an underflow of which generates an internal interruption. The CSR is sixteen bit wide and consists of a 4-bit ALU status, a one-bit IOC, 4-bit interrupt flags, a 3-bit interrupt mask, and 4-bit general purpose flags. Three pins are prepared to handle external interrupt requests. Two of them are usually assigned to the signals from the IOC, which represent unsuccessful transmission. The other is left for miscellaneous use. When an interrupt is accepted, the control memory address is automatically changed to the fixed address. The contents of CMAR and PAR must be stored in the Stack at the beginning of the interrupt process.

Every bit of XAM is used as a jump condition, and also the contents of XAM can be added to the contents of PAR, so that the next address of the control memory is modified dynamically.

I/O controller

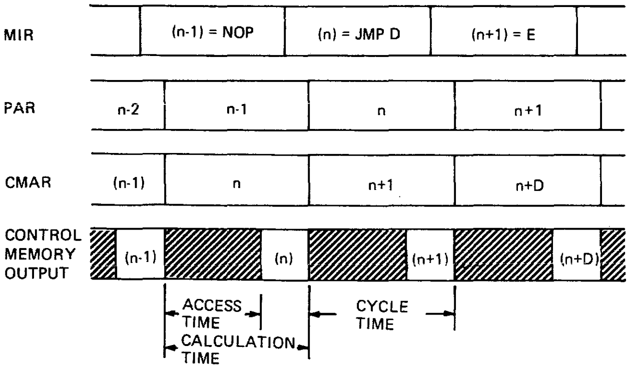
The IOC is designed to interface the EX with the asynchronous bus system. The bus transmission scheme is illustrated in Figure 4. A remarkable feature of this scheme is that as a response to DTSD, three kinds of signals, Data Acknowledge (DTAK), Reject Sync (RSYNC) and Quit (QUIT), are provided for the simple construction of multiprocessor systems. DTAK is returned when there is a



J		D	
E		I	
CJ	TF	BP	D
T	RS	RD	NU
SR		BP	NU
2	2	4	8

J	JUMP	D	DISPLACEMENT	T	TRANSFER
E	EMIT	I	IMMEDIATE DATA	RS	SOURCE REGISTER
CJ	CONDITIONAL JUMP	BP	BIT POSITION OF STR	RS	DESTINATION REGISTER
TF	TRUE/FALSE	SR	SET/RESET BIT		

Figure 2—Block diagram and instruction formats of SC



(n) represents the content of the address n

Figure 3—Timing of pipeline processing

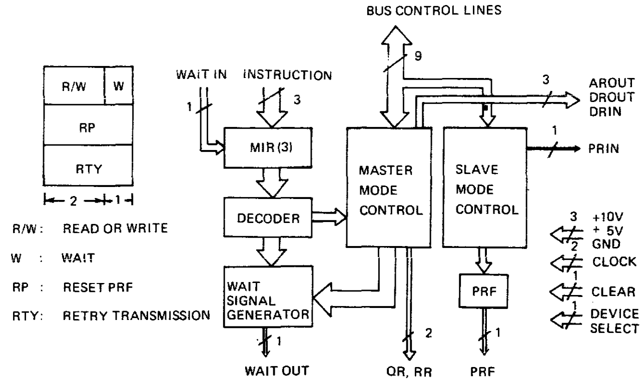


Figure 5—Block diagram and instruction formats of IOC

successful transmission. RSYNC means that the data transmission is rejected by the receiver. QUIT is generated by bus control circuits in the case of unsuccessful transmission caused by hardware trouble.

Figure 5 shows a block diagram and its instruction set. The IOC performs two kinds of operations. One is the master mode transmission, which is caused by Read/Write instructions. In this mode, AROUT and either DROUT or DRIN are controlled by the IOC. If the IOC receives RSYNC or QUIT instead of DTAK, it generates RR and QR, respectively. The other is the slave mode activated by reception of a DTSD when Device Select (DS) is on. DS must be connected to its address decoding circuitry which recognizes when the device is being addressed. In this transmission mode, PRIN controls the reception of the data from the "master" device into PR. Once the data are stored in the PR, the flag is set so that the SC recognizes that the message from an external device has been received. PRIN cannot be activated until the PRF is cleared. RSYNC is returned as the response to DTSD, when PRF is "on."

The first type of instruction to the IOC consists of a 2-bit Read/Write field and a one-bit W-field as shown in Figure 5. When the W-field is on, WAIT is activated during the transmission if it is being processed. WAIT suspends the next instruction until the end of the transmission. The second and third type of instructions provide functions to clear PRF and retry transmission, respectively.

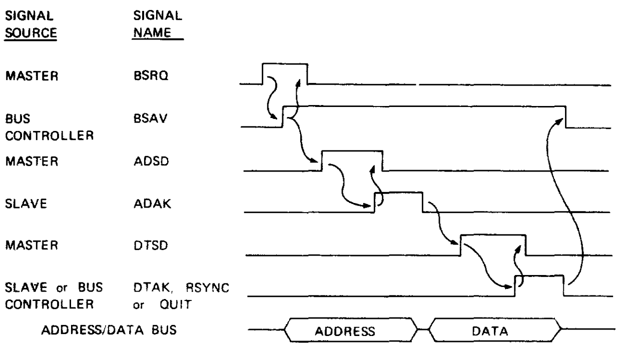


Figure 4—Bus transmission Scheme

SYSTEM COMPOSITION WITH PM/II

Examples of a processor system and a multiprocessor system are described below, which are mainly composed of EX's, an SC, and an IOC, as mentioned above.

Processor composition

Figure 6 exemplifies a standard processor system implemented with PM/II's. The system consists of four PM/II modules, a mapping array, a dynamic microprogram control module, a control memory, and extra circuitry. The number of IC's composing the system are listed in Table II, which shows that PM/II's decrease the need for extra circuitry.

The system contains two data buses. One is an asynchronous data bus which connects main memory, I/O devices, and the other processors. The other is a synchronous data bus which can connect local memory, multiplier/

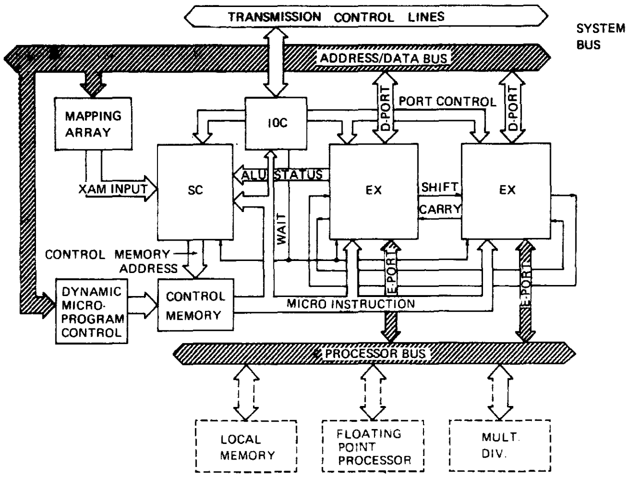


Figure 6—An example of processor construction

TABLE 2 – Components of the system shown in figure 6

CPU	PM/II modules	4 packages
MAPPING ARRAY	MSI	2
	SSI	5
DYNAMIC MICROPROGRAM CONTROL	MSI	8
	SSI	9
CONTROL MEMORY	LSI	27
	SSI	4
EXTRA CIRCUITRY	SSI	18
TOTAL		77

divider, and so forth. This bus obviously can connect some I/O devices directly, therefore PM/II can easily construct intelligent I/O's. The external mapping array is connected to XAM in the SC module which enables a sophisticated emulation for machine languages and higher level languages. These circuits depend on the target machine instruction, which can easily be constructed with a PLA. Two interrupt input pins of the SC are allocated to RR and OR of the IOC. The other interrupt input pin is left for devices which will be connected to the processor bus.

Microinstructions, which are stored in control memory, can be changed by the processor or other processors; therefore dynamic microprogramming is realized in this system. Byte calculations, such as byte-data addition, subtraction, shift, and data-emit, can easily be performed by controlling CE bits.

Systems composed of PM/II can have one of three basic types of microinstructions. Figure 7 shows them for a system with two EX's. These are Horizontal, Joint, and Vertical types. A Horizontal type microinstruction is 41-bits long and has independent fields for SC, EX, and IOC modules. A two-bit CE field activates the EX modules. The CE bit of the SC is set permanently "on," which does not appear in the instruction format as shown. A Joint type microinstruction, 33-bits long, is where a field of EX and SC instructions are partially overlapped. In this type 3 CE bits are required to control two EX's and an SC independently. SC instructions with short formats, such as bit manipulation instructions, can be executed concurrently with EX operations, and EX instructions with short formats with SC operations. A Vertical type microinstruction, 24-bits long, is where EX and SC are fully overlapped. Two CE bits control the operations of EX and SC. When both EX's are not selected, an SC function can be performed.

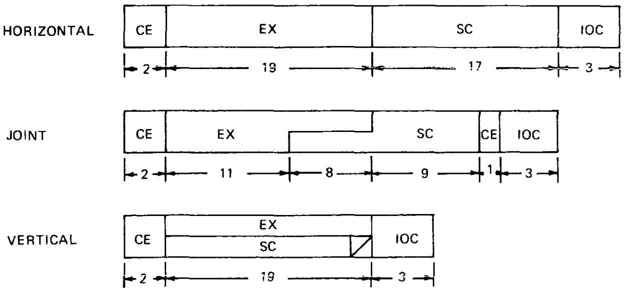


Figure 7—Three types of microinstructions

An emulator for a typical 16-bit minicomputer was implemented on this processor with the PM/II breadboard. The target minicomputer is MACC 7/F manufactured by Matsushita Communication Industry. About 3k bytes of control memory have been required when using vertical microinstructions. It is confirmed that the execution time is less than twice that of the target when executing FORTRAN programs, even without particular external circuits.

Multiprocessor

Figure 8 depicts a memory shared multiprocessor system currently in operation. P1 is built up by the PM/I components which are prototypes of PM/II. P2 consists of the PM/II modules implemented by the breadboard, and P3 is a MACC 7/F with the bus interface for the PM/II. The element processors, connected with a single bus, communicate through the shared memory and the P-Register. The microprograms of the element processor P1 can be loaded dynamically by P3, and P2 loads its own microprograms dynamically, while P3 has a fixed instruction set. Using the multiprocessor system, many fruitful experiments on the flexibility of hardware configuration, process efficiency, and control program description are made. As a result, the following advantages have been derived to display the applicability of PM/II in the multiprocessor environment; (1) hardware can be connected in a unique and simple way, (2) software for each processor can be developed and debugged almost independently, and (3) the conciseness and ease in description particularly of distributed function type and/or resource shared type control programs can be achieved.

A high level language interpreter is now working on the multiprocessor system, where P1, P2, and P3 process garbage collection, language interpretation, and I/O control, respectively.

Through monitoring the system, it has become clear that the bus contention begins to occur when more than three processors emulate machine instructions simultaneously. Based on the results of these experiments, a new multiprocessor system employing PM/II, in which every element

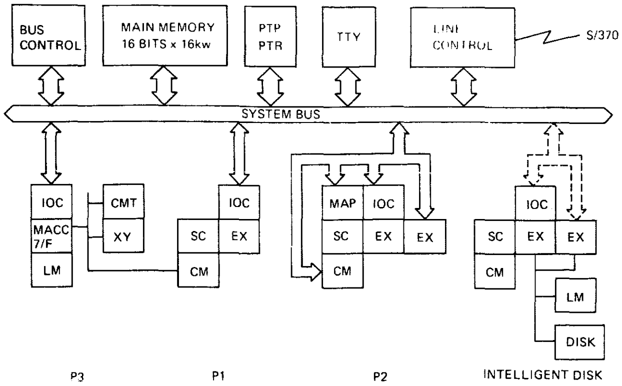


Figure 8—Expermental multiprocessor system

processor has its own local memory, is currently under development.

CONCLUSION

The design concept, specifications, and system composition of PM/II's are described above. Several instruction mixes are estimated, which are illustrated in Figure 9 and compare PM/II with three microprocessors on the market. Intel 3000 and PM/II represent their performance when they compose 16-bit computers, while PFL-16A is a 16-bit microprocessor itself. The performance of PM/II is measured at its vertical instruction set, which shows very satisfactory results.

The breadboards of PM/II are in operation, and have been used for experiments on machine language and high level language emulation, and so forth. A memory shared multiprocessor system with three processors including the PM/II is running, which has made clear the validity of the design concept.

Based on the results of these experiments, a new multiprocessor system employing PM/II is currently under development. Design of the operating system and language processors is nearly finished, and the hardware implementation will soon be started.

Along with the many results which have shown the applicability of PM/II, one disadvantage has appeared, i.e., the slight difficulty in microprogramming. This may be caused by the adoption of pipelining. In order to reduce this difficulty, and to support hardware and microprogram debugging even before hardware implementation, an integrated development support system named IMPULSE has been designed and implemented. IMPULSE consists of a debugging part and a microprogram generating part. The former debugs hardware and microprograms with a description of module connections and microprograms, driving a multi-level logic simulator. The latter is a hierarchical system composed of a high-level microprogramming language compiler, an optimizing assembler in which users can write programs without considering pipelining, and a general purpose micro-assembler. The precise description of IMPULSE will be available in a separate paper.

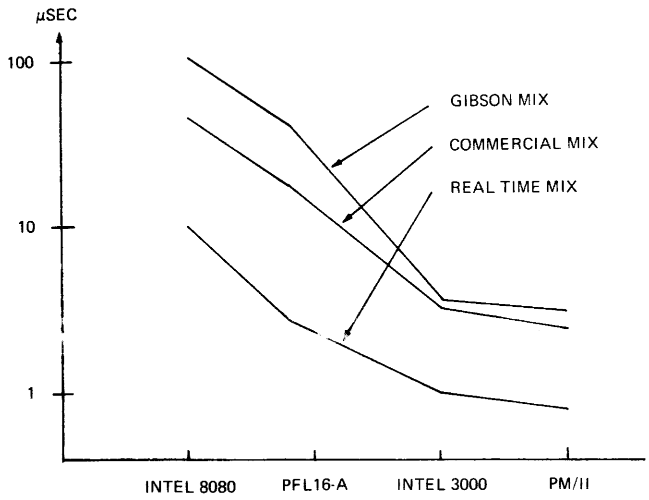


Figure 9—Evaluation of performance

The three modules of PM/II are now being implemented in the CMOS process. This process is selected because of its high noise immunity and low stand-by power dissipation. PM/II will augment the consistency and completeness to develop other required modules in the future.

ACKNOWLEDGMENT

The authors wish to thank Dr. Y. Koike, the President of Matsushita Research Institute Tokyo, Inc.; Dr. T. Yoshida, the Managing Director, who powerfully promoted the research; Dr. M. Ohnuki, the Director and the colleagues of his group for their ardent semiconductor technological examination; Dr. K. Ueda for valuable discussion; Mr. S. Shimazaki for his assistance in the implementation and experimental stages.

The authors are also grateful to Professor H. Aiso of Keio University for his numerous suggestions from an architectural viewpoint.

REFERENCES

1. Clark, W. A., "Macromodular Computer Systems," *SJCC 1967,* pp. 335–401.
2. Bell, C. G., J. E. Eggert, J. Grason, and P. Williams, "The Description and Use of Register Transfer Modules (RTM's)," *IEEE Trans. C,* Vol. C-21, No. 5, 1972, pp. 495–500.
3. Reigel, E. W., U. Faber, and A. Fisher, "The Interpreter—A Microprogrammable Building Block System," *SJCC 1972,* pp. 705–723.
4. Bell, C. G., R. C. Chen, S. H. Fuller, J. Grason, Satish Rage, and D. P. Siewiorek, "The Architecture and Applications of Computer Modules: A set of Components for Digital Systems Design," *Digest of COMPCON 73 Spring,* pp. 177–180.
5. *SUE Computer Handbook,* Lockheed Electronics Company, Los Angeles, 1972.
6. *PFL-16A Technical Summary Manual,* Panafacom Limited, Tokyo, 1976.

An organization for optical linkages between integrated circuits

by G. JACK LIPOVSKI
University of Texas
Austin, Texas

ABSTRACT

Conventional integrated circuit packaging techniques which use pins for input-output have several disadvantages, which are becoming increasingly important as more logic is put on a chip. Recent intensive development of optical links for long distance communications suggests that they could be used between integrated circuits, to alleviate the bottleneck created by the connection technology, so that LSI technology can be further exploited. However, the complex linkages between integrated circuit cannot be economically realized by just replacing each wire by an optical link; rather a "bus organization" should be developed so that, by time multiplexing one optical link that threads just once through each integrated circuit, different time slots can be used to realize any necessary transfer. Additionally, the time slices are controlled by a microprogram, so that "wiring changes" can be realized by program changes. A unified treatment of busses is first developed, then two physical realizations of the time-multiplexed bus are described.

INTRODUCTION

Large scale integration (LSI), and microprocessors in particular, have revolutionized our approach to digital systems. In this paper, we report on what we perceive will be the next step in this revolution. To support this opinion, especially to answer the objections of some of our respected colleagues, and to justify some decisions we make later, we aim to identify some of the implications of the LSI revolution. We do this using the premise-implication style[9-13] because this style clearly exhibits our assumptions and presents our design decisions in an organized way that invites the reader to carefully scrutinize them.

By way of introduction, we state what we believe are the ground-rules of our approach; the current state of the art:

P1: Many (thousands of) gates can be economically put on an IC, but few (tens of) pins can be put on one chip.

P2: A communication link exists which is much faster than the processor it is connected to.

Premise 1 reflects the fact that for optimal cost, about 2,000 gates should be put on an IC chip.* Consider that the single gate 7430 sells for about 10¢, while the (approx.) thousand gate Intel 4040 microcomputer sells for about $5.00—a 20 to 1 cost ratio per gate. However, no more than a few connections (pins) can be made to the chip. We soon argue that the connection (pin) limitation is the bottleneck to development of LSI systems. Premise 2 indicates that a very fast connection mechanism exists. In most technologies and in the majority of applications, which do not require much speed, it is possible to make most of the circuitry small and low power, and therefore slow, but to make a small part of it much faster, and therefore larger in size and more power consumptive. So the bottleneck raised by premise 1 can be resolved by trading pins against speed by time multiplexing a very fast communication link. We will soon argue that optical links, or light pipes, will provide such speed that a few such links could replace all but four of the pins, and provide sufficient communication bandwidth for the larger and larger chips that will be appearing.

I1: Minimum hardware cost is achieved, not by minimizing gates, but by using extra hardware to minimize connections.

Argument

Some ten years ago, minimizing gates generally minimized hardware cost. Even now it might appear that minimizing gates minimizes the number of integrated circuits, but this is true to some extent now only for design with Small Scale Integration (SSI), and for the design of integrated circuits. In fact, it is common now to waste gates in order to minimize the number of IC's and the amount of interconnection. For instance in a memory system, an address decoder is put in each memory chip. Really, only

* We beg the reader's permission to use some approximate figures in support of our arguments without giving references. Many of these figures are available from current catalogs, but others were obtained through private communication with industrial contacts. Although permission to use approximate numbers was granted, permission to identify the source was generally not given.

one decoder is actually needed, but a decoder is put in each chip so that $\log_2 n$ address pins can select one of n bits or words in the memory chip. Gates are often wasted to minimize interconnections in this manner. Minimizing gates, per se, is now generally pointless. We have to learn to minimize the number of chips and the amount of interconnections between chips.

End of argument

I2: Better communication links than conventional IC pins will soon be required.

Argument

We argue this question by observing the rate of growth of logic on the chip, and of pin connections to the chip, and then by observing some of the costs of current systems.

Consider that the maximum number of gates on a chip doubles every year or so. Consider that the 256 bit memory was followed by the 1K, then the 4K, and now the 16K memory in rapid succession. We note that these increases could be handled by adding just a few more address pins, because memories in particular can take advantage of I1. However, many large volume applications of microcomputers require only 1K words. We will reach a saturation point to memory size. Then larger memories, and larger microprocessors, can only take advantage of doubled logic complexity by doubling the width of the computer word, and thus the number of input/output pins. That can only last for a short time, for there is little demand for 32 bit or 64 bit microcomputers. Finally, as more logic can be put on a chip, more of the system components are put on it. If one puts the 8 bit CPU on a chip, some forty links are needed; if one puts the entire computer with I/O logic on a chip, some twenty links are needed for each I/O device. In these last two cases, the number of pins should be significantly increased each year to take advantage of the logic on a chip. However, the maximum number of pins has increased about linearly with time from 14 to 24 to 40, and now 64. Clearly, we are running out of pins.

Logic to minimize pins (I1) should be used. However, some current examples of this, such as sequentially sending the data and address on the same pins in the 8008[19] or data and status information in the 8080[18] require an extra register to hold the address or status. Although this reduces the number of pins on the large CPU chip, it actually adds more pins to the entire system since the register needs input and output pins for each bit it stores. It makes the CPU chip cheaper, but adds to the system cost. We perceive that these techniques using logic to minimize pins will not be adequate.

We now consider some costs of current integrated circuit chip and printed circuit board techniques. Each output connection requires a large buffer amplifier and connection pad on the chip and each input requires a pad and protection circuitry. In fabricating the chip, wires are bonded one at a time to the pads, and each bond has a probability of destroying the chip. Fabrication time and cost, and rejected chips are therefore increased as the number of pins. In putting the chip on a PC board, significant costs are encountered. Although the chip itself is perhaps 0.1″×0.1″, the DIP package is perhaps 2″×1″ to make the connections to the PC board. If it weren't for the pins, a complete microcomputer system would occupy, say, a 2″ square. The capacitance of the DIP package and PC board considerably slows down signals sent through pins. A good quality (high reliability multilayer) PC board alone can cost $500.00, and such a board may support only, say, $100 worth of integrated circuits. Design errors and custom design changes are difficult to correct and implementation multilayer boards. Finally, failures are often caused by bad PC board connections, and failures are increased as pins are increased.

The pin connection technology is already a major source of trouble and cost, and should become even more so as time goes on. If it can be replaced by a better technology, perhaps even if that technology has a few problems of its own, we should seek out and develop this technology.

End of argument

I3: An optical link should be developed to interconnect integrated circuits.

Argument

Optical links are being intensively studied to replace coaxial cable in the communications industry. Very encouraging results were reported in the August 5, 1976 issue of Electronics, entitled, "Optical Communications, its time has come."[14] Development of optical communication systems for telephone, computer and even airplane and automobile electronics were reported. Articles on the light sources like LED's,[15] and lasers indicate that the former can now be modulated at up to 200 mHz at 10 percent efficiency and the latter can be modulated at 1 ghz. Although LED's and lasers are not directly compatible with MOS or bipolar technology, they can be mounted on the substrate as they are in the Hewlett Packard 7300 numeric and hexadecimal indicators. 5 ghz MESFETS and very fast Transfer Electron Logic devices[3] can drive these sources. A report on light fibers[16] indicates that commercially available step-index fibers can transmit 500 Mb/km, which implies that for the short distances between integrated circuit chips, very high bandwidth is feasible. For the short distances between IC's, coupling between fibers will not be necessary, and "sloppy" coupling to and from the fibers should be quite acceptable. 1 GHz silicon Schottky-Barrier photodiode detectors[1,2] have been easily fabricated. There are detectors like those above which are compatible with bipolar logic, and other detectors that are directly compatible with MOS technology. The technology to transmit data at 50 megabits per second on one light pipe required for a microcomputer data and address bus system is certainly available today at a cost that is probably comparable to the cost of reliable multilayer circuit boards, and the potential for economic gigabit per second transmission on light pipes is not that far off. Although light pipes are now being developed for long distance communication, such as between a computer and an intelligent terminal, this technology could provide better communication links than conventional IC pins for IC's themselves (I2). At the very least, it cannot be denied, this possibility is worth studying.

End of argument

I4: An organization for optical links for integrated circuits should be developed.

Argument

In long distance communication, the communication is basically point-to-point. There isn't much logical organization to concern the designer. Coupling tens of terminals by means of light pipes has been reported.[17] However, between IC chips, point-to-point communication would require too many LED transmitters and detectors on each chip, and the star coupling in Reference 17 would not handle a few very important linkages such as priority networks and carry lookahead. We submit that, if a bus is realized by optical links and that bus requires a priority circuit, we should be prepared to realize the priority circuit too in the same optical link. We opt to use time slices of a time multiplexed very fast optical link to realize all the linkages required between integrated circuits. Moreover, we will control the time slices by a microprogram. Wiring changes cited in I2 as a problem for multilayer boards becomes easier because they are now program changes. Primarily, we believe all connections required by a typical microprocessor must be realized. Secondarily, we think that optical linkages should be developed for MSI, and even some SSI which might be necessary to complete a microcomputer or to implement a system without a microcomputer.

End of argument

In the next section, we develop a suitable theory on classifications of busses which, to our knowledge, does not exist in the literature. This classification scheme is used later to show that the carry lookahead linkage is the universal linkage from which all other important linkages can be derived. That is, if we implement one carry lookahead linkage that "threads through" all chips, we can realize any linkage between any of the chips by time slices on that time multiplexed link. In a later section we show two physical realizations of the universal linkage, the carry lookahead linkage.

A CLASSIFICATION OF LINKAGES BETWEEN IC'S

The kinds of linkages between chips are categorized now, setting the stage for development of the universal linkage, and introducing some terminology for the rest of the paper. This terminology generally follows Bell and Newell.[5] A link is a single wire (scalar), or a bundle of wires (vector) that provides communication between ports. Suppose B_1, B_2, . . . B_n are ports that input information to links from modules and C_1, C_2, . . . C_m are ports that output information from the links to the modules. A simple link has just one pair of ports B_1 and C_1 on the same link. It is simplex if information is transmitted in only one direction, say from B_1 to C_1. It is half-duplex if information can be transmitted in either direction, but in only one direction at any given time. It is full-duplex if information can be transmitted in both directions at the same time. It is (time) multiplexed if different information signals are transmitted over the same link on different time slots. Each time slot can be simplex, half-duplex, or duplex on a multiplexed link.

Complex links have more than two ports, and are classified herein as follows. A broadcast link or time slot has one port, say B, broadcasting to the other ports, C_1, C_2, . . . C_m. A special port is identifiable as a broadcaster in this type of link or time slot. For instance, a microprocessor commonly broadcasts an address to memory chips on a broadcast link. A collection link or time slot has one port, say C, collecting signals (possibly by OR'ing them or AND'ing them together in a wire-OR or wire-AND link) from the other ports B_1, B_2, . . . B_n. Usually only one port, B_1, is transmitting information while the others are transmitting 0's (for wire-OR) or 1's (for wire-AND). Alternatively, using tristate logic, only one port B_1 will send out a signal while the other ports appear as open circuits. A special port is identifiable as a collector in this type of link or time slot. For instance, a collection time slot is used on the data link of a microcomputer when one of several memory chips read a word back to a microprocessor. On occasion, a general broadcast-collection link or time slot may have more than one broadcaster and more than one collector. For instance, a general broadcast-collector link is equivalent to the conventional I/O bus such as the unibus.[7]

A propagating link* or time slot OR's or AND's the output of each broadcast port to the higher numbered collection ports, or conversely, OR's or AND's the outputs from lower number broadcast ports as follows. B_1, B_2, . . . B_n are broadcast (input) ports and C_1, C_2, . . . C_n are corresponding collection (output) ports. In an OR propagating link, C_2 is B_1, C_3 is B_1 OR B_2, C_4 is B_1 OR B_2 OR B_3, etc. (C_1 is 0 by default). Propagating links are used, for instance, in priority circuits, where requests R_1, R_2, . . . R_n are put into B_1, B_2, . . . B_n and grants G_i are R_i AND NOT

* One might argue that this is not a link because it performs logic and has an order or direction among ports. However, wire-OR collection links perform logic, and simplex links have a direction. Moreover, propagating links (e.g., priority circuits) form a small but important part of the linkage problem between integrated circuits. They must be handled just as broadcast or collector links. Thus, we classify this as a link.

C_i. The propagating link serves to notify the ith stage that some higher priority stage (1, 2 . . . i−1) has a request, so that the request of this stage will not be granted. A round-robin propagating link, used in round-robin priority circuits, selects some state i as the "beginning" of the propagating circuit, C_{i+1} is B_i, C_{i+2} is B_{i+1} OR B_i. The link wraps around circularly, so that C_1 is B_i OR B_{i+1} OR . . . OR B_n, etc. and C_{i-1} is B_iEL G_{i+1}OR . . . OR B_n OR B_1 OR . . . OR B_{i-2}.

End of argument

I5: Simple simplex, broadcast, collection, general broadcast-collection, progagating, round-robin propagating, and lookahead linkages will be realized between integrated circuits.

Argument

These classes of links have been used in a course in top-down design of digital circuits[8] for some six years to assist the students in deciding what they need to do to link digital modules together. For each class, they are shown chain realizations (e.g., as in a ripple adder), minimum delay realizations (e.g., as in a wire-OR bus or carry lookahead adder) and tree realizations (e.g., as in a tree of OR gates) and the delay, fan-out and cost properties of each are considered. Experience has not shown links that do not fit into these classes.

Moreover, if one of these classes cannot be realized, then extra wiring or extra light pipes will be needed. There is no point, for instance, in converting an I/O bus to light pipes if it needs a priority circuit to effect DMA unless the priority circuit can be economically implemented. And given the very high speed of light pipes (I3), it should be implemented as a time slice of the same "bus" that carries the data that the priority circuit controls (I4). If this is not possible, the extra cost of additional pin connections or light pipe connections to realize a priority circuit may significantly detract from this approach.

End of argument

THE LOOKAHEAD AS A UNIVERSAL LINK

In this section, we show that any of the links in I5 can be realized as special cases of the lookahead link that is available in a carry lookahead generator like the SN74182. Thus, if we can build a lookahead link, we can program time slices on it to realize the other links. We first show that a complete carry lookahead generator can realize any linkage in I5, then we show that contiguous sections of it can be programmed to independently realize these linkages.

I6: Let G_1, G_2, . . . G_n be generates, P_1, P_2, . . . P_n be propagates, and C_1, C_2, . . . C_n be collectors (carries), and C_{in} be the carry input and C_{out} be the carry output of a carry lookahead generator. A lookahead link is directly implementable in this circuit. Moreover, C_{in} can broadcast to all C_i, or C_{out} can collect from all G_i, or G_i and C_i can be inputs B_i and collectors C_i for a propagating or round-robin propagating link, or C_{in} can communicate to C_{out} on a simple simplex bypass link. Finally, in a tree, if G_T is connected to C_T in the root of the tree, then the OR of G_1, G_2, . . . G_n is broadcast to C_1, C_2, . . . C_n if P_1, P_2, . . . P_n are 1.

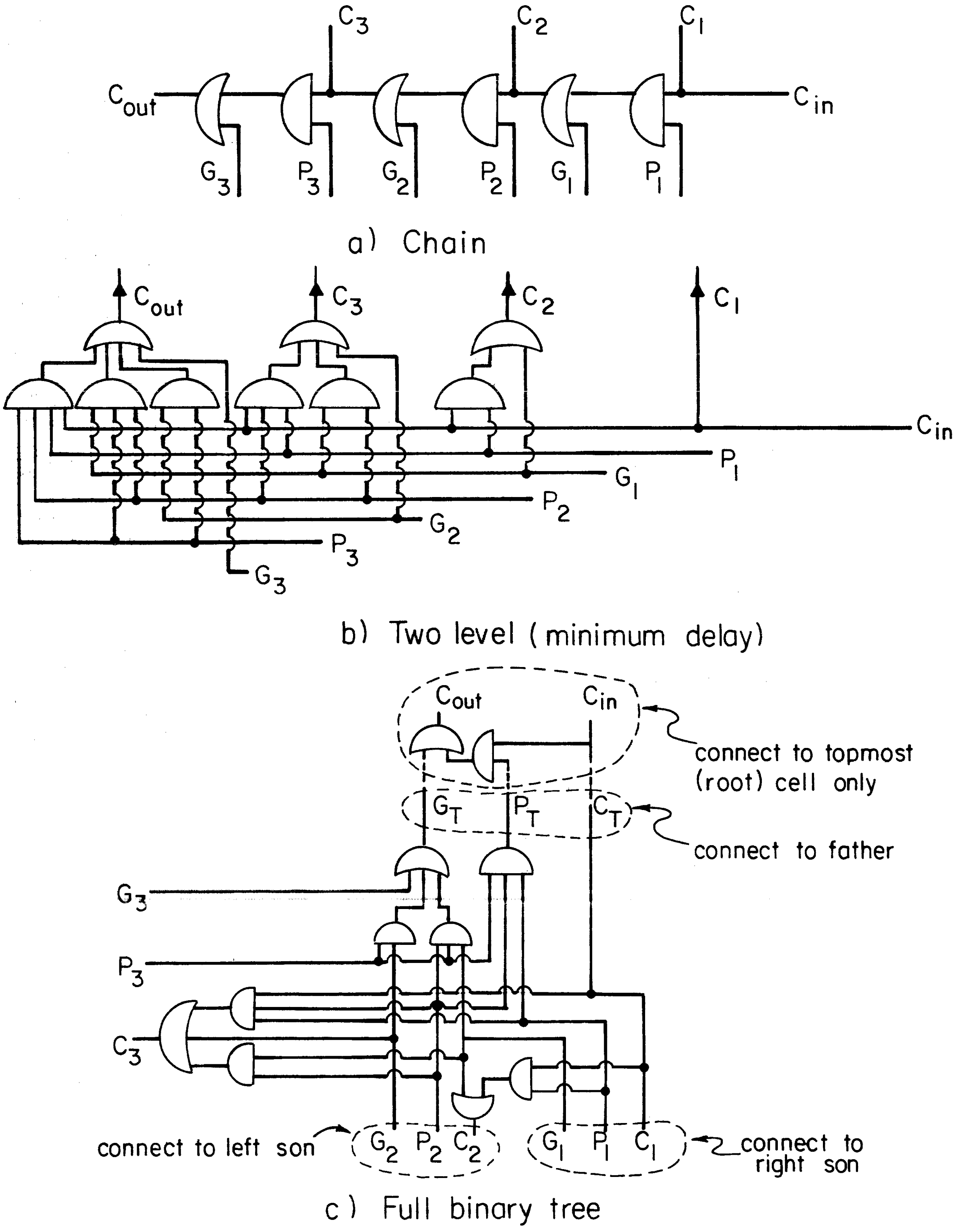
Argument

We show this by example. See Figure 1a, a chain carry lookahead, Figure 1b, a two-level carry lookahead, and Figure 1c, a full binary tree carry lookahead. If gates have zero delay, these circuits are identical. We consider each case described in implication 6 now.

A lookahead link is implemented by putting the generates and propagates from an adder into G_i and P_i, and feeding the collectors C_i into the carry inputs of the adder. A broadcaster is implemented by forcing P_i to '1' and G_i to '0.' Then C_{in} is broadcast to all C_i. An OR collector is implemented by setting P_i to '1', and ignoring all inputs C_i. Then any signal broadcast into G_i will be collected by C_{out}. An OR propagating link can be realized by setting P_i to '1.' Then signals broadcast into G_i are OR'd into C_j, j>i. Similarly, if P_i is '0' and all other P_j are '1', and C_{out} is fed into C_{in} then a round-robin propagating link is effected for inputs G_j and outputs C_j. A simple simplex bypass link from C_{in} to C_{out} can be realized by making P_i '1' and G_i '0', not using C_i. Then C_{in} broadcasts only to C_{out}. Finally, in a tree circuit, the signals from C_{in} do not feed into G_T of the root cell of the tree, while G_T collects all G_i, and C_{in} is broadcast to all C_i. Thus, without any feedback loop that might cause the logic to latchup or have excessive propagation delays, G_T can be fed into C_{in} so that the OR of G_1, G_2, . . . G_n is broadcast to C_1, C_2, . . . C_n. This realizes the general broadcast-collector link or input-output "bus" in the whole circuit.

End of argument

I7: The carry lookahead generator can be time-sliced by decentralized programs in each integrated circuit, so that in every slice the chain of generators G_1, G_2, . . . G_n and their corresponding propagates and collectors can be segmented into contiguous segments, say S_1=(G_1, G_2, . . . G_{i-1}), S_2=(G_i, G_{i+1}, . . . G_{j-1}), S_3=(G_j, G_{j+1}, . . . G_{k-1}), and S_4=(G_k, G_{k+1}, . . . G_n), where each segment effects one of the linkages discussed in I1, except the general broadcast-collector link and round-robin propagating link. The last generator of a broadcast segment (e.g., G_{i-1} if S_1 above is a broadcast segment), or a lower indexed collection or priority segment (e.g., S_1) can provide information to be broadcast in a broadcast segment (e.g., S_3) if all intermediate segments are bypass segments (e.g., S_2).



a) Chain

b) Two level (minimum delay)

c) Full binary tree

Figure 1—The carry lookahead generator

Argument

By means of a cyclic program of m steps, $s_1, s_2, \ldots s_m$, the generates and propagates can be programmed to provide m time slices. See Figure 5. Each step of the program should select a generate which is '0', or some input from the integrated circuit that is to be broadcast (e.g., the adder generate signal), a propagate which is generally '1' or '0', and a destination, if any, of the data coming from the collection input. The program can be decentralized so that the integrated circuit having G_i, P_i, and C_i contains internal to it the orders to select inputs to G_i and P_i, and the destination for C_i. Means to start each program together (initialization signal) and to step each program together (clock) can be provided. The linkages between integrated circuits are thus established by the programs. Such programs can be stored in each IC in a read-only memory, PROM, EAROM, or equivalent.

Generally, the propagate is '0' in the last propagate of a broadcast, priority or lookahead segment (e.g., if segment S_2 discussed above is a broadcast segment, P_{j-1} is '0'), and is the propagate signal from the adder inside a lookahead segment, otherwise it is '1'. The generate is '0' inside a broadcast or bypass segment and is made to be the signal to be broadcast in a collection, priority, or lookahead segment. Thus, these linkages are easy to realize. Now note that if two segments S_i and S_{i+1} are respectively, a collection and broadcast segment, then the effective C_{out} for S_i, the collected data, becomes the effective C_{in} for S_{i+1}, the broadcasted data. Generally, several generator signals in S_i are OR'd together, and broadcast to all collectors in S_{i+1}. Moreover, if S_i is a broadcast segment and consequently the last propagate is 0, then the effective C_{out} is exactly the last generate. By use of bypass linkages, if S_i is a collector (or S_i is a broadcaster), S_{i+1} is a bypass link, and S_{i+2} is a broadcaster, the (last) generators in S_i can broadcast to one (or more) collectors in S_{i+2}. Communication can efficiently be done in one direction, from G_i to C_j, $j>i$, in general. Note that the length of the bypass link should be kept small to efficiently use the lookahead generator in this way. However, simple simplex linkage from any C_i to any C_j, $i<j$, is feasible.

End of argument

From I7 the following communication scheme is suggested. Since the technique in I7 affects communication in only one direction, a pair of lookahead generators will be used, where the second propagates in the reverse direction. The integrated circuits will be arranged in some order in the first lookahead generator and in the reverse order in the second lookahead generator (the optimal arrangement should be done by some algorithm which is not yet known to us). The chips will be connected by a lookahead generator with G_i, P_i and C_i in IC_i, and a second lookahead generator with G_j, P_j, and C_j in $ICn_{+1-j)}$. The program to time share the lookahead generators would be set up in this way. For each required linkage, the range of collectors and generators that will be committed to that linkage will be recorded. (Note that the generators normally go with collectors except that the last generator of a broadcast segment works independently of its collector.) These will be combined by a scheduling algorithm so that as many linkages as possible are effected in each time slice, or conversely, so that all linkages can be effected in the minimum number of time slices. The programs will be derived for each chip, and these will be stored in each chip.

Consider, for example, a conventional 8 bit microcomputer in IC_1 primary memory for the microcomputer in IC's 2 to 4, and input/output chips in IC's 5 and 6. A fetch cycle might take 25 steps on the two lookahead generators. The first step uses the forward lookahead for IC_1 to send a signal to IC's 5 and 6, such as the enable signal for direct memory access while the reverse lookahead collects interrupt requests and establishes priority from IC_5 and IC_6 to be broadcast to IC_1. The next 16 steps broadcast to on the forward lookahead generator the 16 bit address from IC_1 to IC's 2 to 4 where the addresses are assembled and decoded. The next 8 steps collect the data (instruction) in the reverse lookahead generator from IC's 2 to 4 into IC_1. We assume that the lookahead generator is 25 times faster than the microprocessor, in order to collapse the twenty-five bus wires into one.

A tree lookahead generator can be made by interconnecting modules such as these in Figure 1c, in a tree, as in Figure 2. The collector, generate and the propagate links of the father (e.g., IC_1) are connected to those of the son (e.g., IC_2). Leaves of the tree (e.g., IC_4) have no sons. The tree has an additional advantage for lookahead generators. The tree can be pruned into subtrees, such as IC_3, IC_6, and IC_7, where the subtrees are independent lookahead generators that are capable of being segmented in time slices, or used as a whole for a general broadcast-collector linkage.

I8: A tree carry generator can be programmed to form subtrees in various time slices by setting C_T to '0' or to G_T in the root of the subtree, and setting the value of G and P into its father the values '0' and '1', instead of connecting G, P, and C between father and son. The separate subtrees can independently realize the linkages in I2 if C_T is set to '0' or the general broadcast-collector link or round-robin propagating link, if C_T is set to G_T, or be pruned from the remaining tree if a fault exists in it.

Argument

As stated, subtrees can be created to effect separate linkages. In Figure 2, the integrated circuits are ordered as follows for the full tree: IC_7, IC_6, IC_3, IC_5, IC_4, IC_2, IC_1. By creating a subtree below IC_3, two separate linkages with order IC_7, IC_6, IC_3 and IC_5, IC_4, IC_2, IC_1 are effected. Each can be programmed as in I2 if C_T for the root node of the subtree is 0. Additionally, if C_T is connected to G_T and all P_i in the subtree are '1', the whole subtree becomes a general

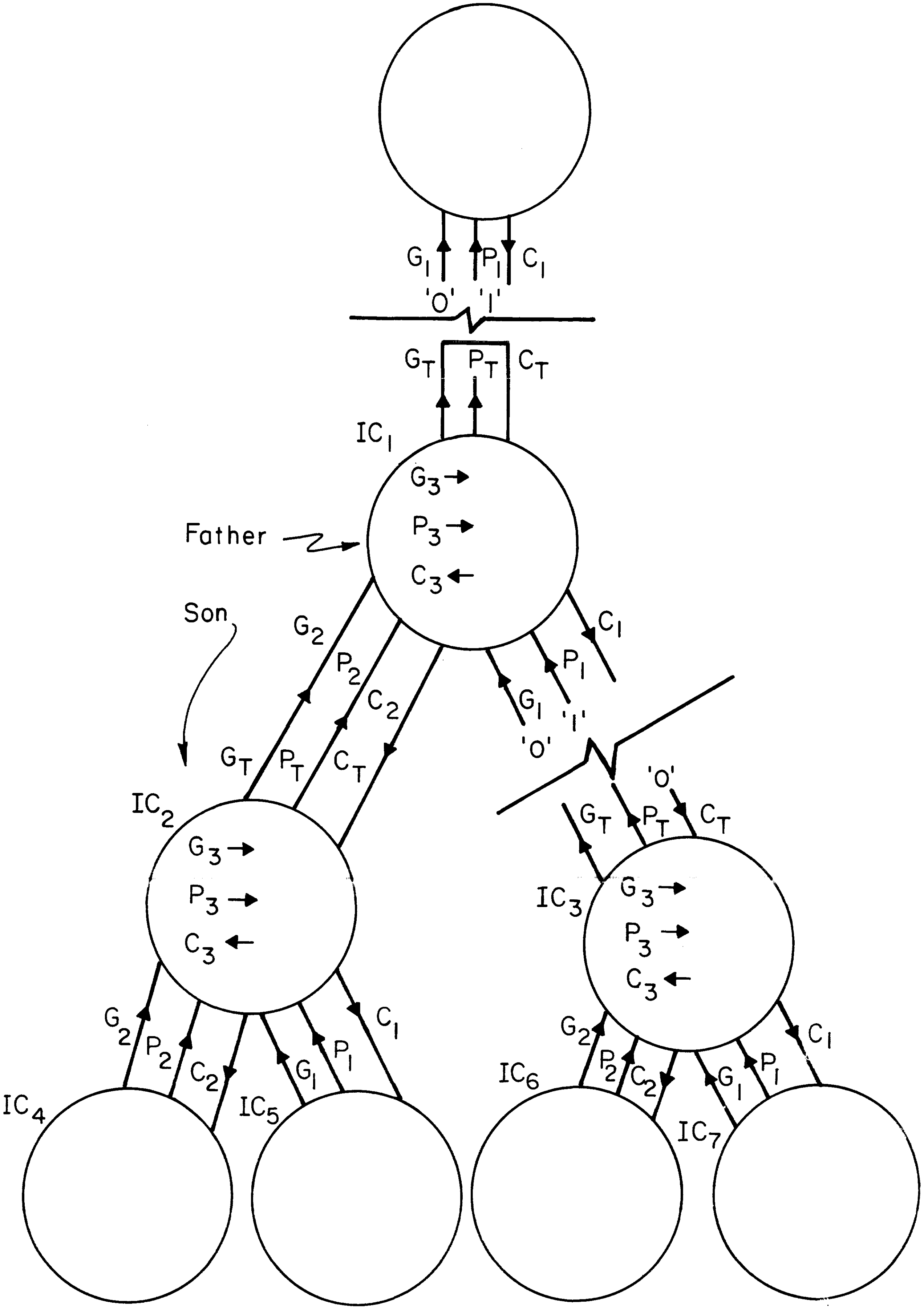


Figure 2—A tree lookahead generator

broadcast-collection linkage, like an I/O bus. Alternatively, the linkage for a round-robin priority circuit can be obtained. See I_4, I_5, I_2, I_1 for instance. Finally, if a fault exists in the subtree, the subtree can be pruned from the remaining tree to permit the latter to function correctly.

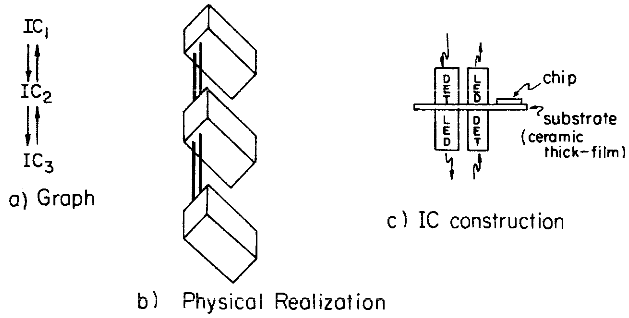
End of argument

Creation of general links and round robin propagating links are very significant, since they provide communication from generators to the right of collectors in this order, whereas the techniques in I7 provide only linkages from generators to the left of collectors. Thus, two separate carry lookahead generators are not necessary. The tree lookahead can provide linkage in both directions.

The techniques in I8 suggest a modular approach to designing links. A subtree can correspond to a submodule with a bus within it. The bus and support linkages (e.g., priority circuits) can be realized in time slices in the lookahead generator subtree. Other linkages discussed in I7 can be implemented within the submodule. From time to time when this submodule needs to communicate with other submodules, the root node of the subtree can be programmed to connect the subtree to the neighboring subtree(s). Then the larger subtree can be treated as a general broadcast-collector bus, or segmented as discussed in I7.

OPTICAL LINKS FOR THE LOOKAHEAD GENERATOR

Having shown that the lookahead generator can be programmed to realize all of the required linkages between integrated circuits, we now consider how it can be fabricated with optical transmitters (LED's) and optical receivers or detectors (photodiodes). Now the ripple or chain lookahead generator (Figure 1a) has some advantages, especially in minimizing the number of optical links between chips. A realization is shown in Figure 3. A forward lookahead generator requires but one link between each chip, and a reverse lookahead generator would require a second link. However, the tree lookahead generator has other advantages, which might make it attractive for larger systems. A possible realization of it is shown in Figure 4.



a) Graph

b) Physical Realization

c) IC construction

Figure 3—A chain lookahead generator

Figure 3a shows the graph representation with forward and reverse chain lookahead generator. Figure 3b shows how these links can be easily implemented by a stack of chips, where Figure 3c shows the placement of transmitters (LED's) and detectors (DET) in a side view of the integrated circuit. Figure 4a shows the tree lookahead generator. The nodes of the tree are arranged in "outline form" or preorder in Figure 4b, and optical links connect the generate, propagate, and collector signals between father and son. Note that four distinct types of integrated circuits are used in this realization: depending on whether the node is a leaf which has no sons, or a non-leaf, which has sons, and whether the node is a left or right son. Type A is a non-leaf right son, type B is a non-leaf left son, type C is a leaf right son, and type D is a leaf left son. The placement of transmitters (LED's) and detectors (DET) for a type A integrated circuit in a side view of the integrated circuit is shown in Figure 4c. This one dimensional scheme can be extended to three dimensions by letting a node of one chain be the head of another chain perpendicular to the first chain. See Figure 4d.

The logic to supply and use the generator, propagate and carry linkages is shown in Figure 5, and the logic for the linkage is shown in Figure 1. The logic in Figure 1 should be as fast as the optical link. The logic in Figure 5 should be fast enough to supply propagate and generate signals and to collect the incoming signals. However, the rest of the logic on the chip can be conventional slow logic that might be used on high density chips.

I9: If the delay through the optical link and lookahead logic is acceptably low, then the chain lookahead generator (Figures 1a and 3) should be used. If this delay is substantial, or if the general broadcast-collector link round-robin propagating link, or the pruning of faulty nodes is useful in submodules as in I6, then the full binary tree lookahead generator (Figures 1b and 4) should be used.

Argument

We first argue that a lookahead generator of some kind should be time multiplexed to realize different linkages. From I5, we aim to implement those linkages that have been identified and classified earlier.

However, we are constrained to use a minimum number of actual simple simplex links between integrated circuits. I7 has shown that such links can be realized by time multiplexing a lookahead generator, which uses simple simplex links between integrated circuits. This provides adequate capacity to provide hundreds of linkages, if fast Schottky logic and detectors are used in the lookahead generator while high density slower logic, such as MOS, are used in the logic supported by these linkages. The actual physical connections can be reduced to four wires (power, ground, clock and initialize as discussed in I1) and two optical links (as shown in Figure 3). The chain lookahead generator thus well satisfies Implication 4.

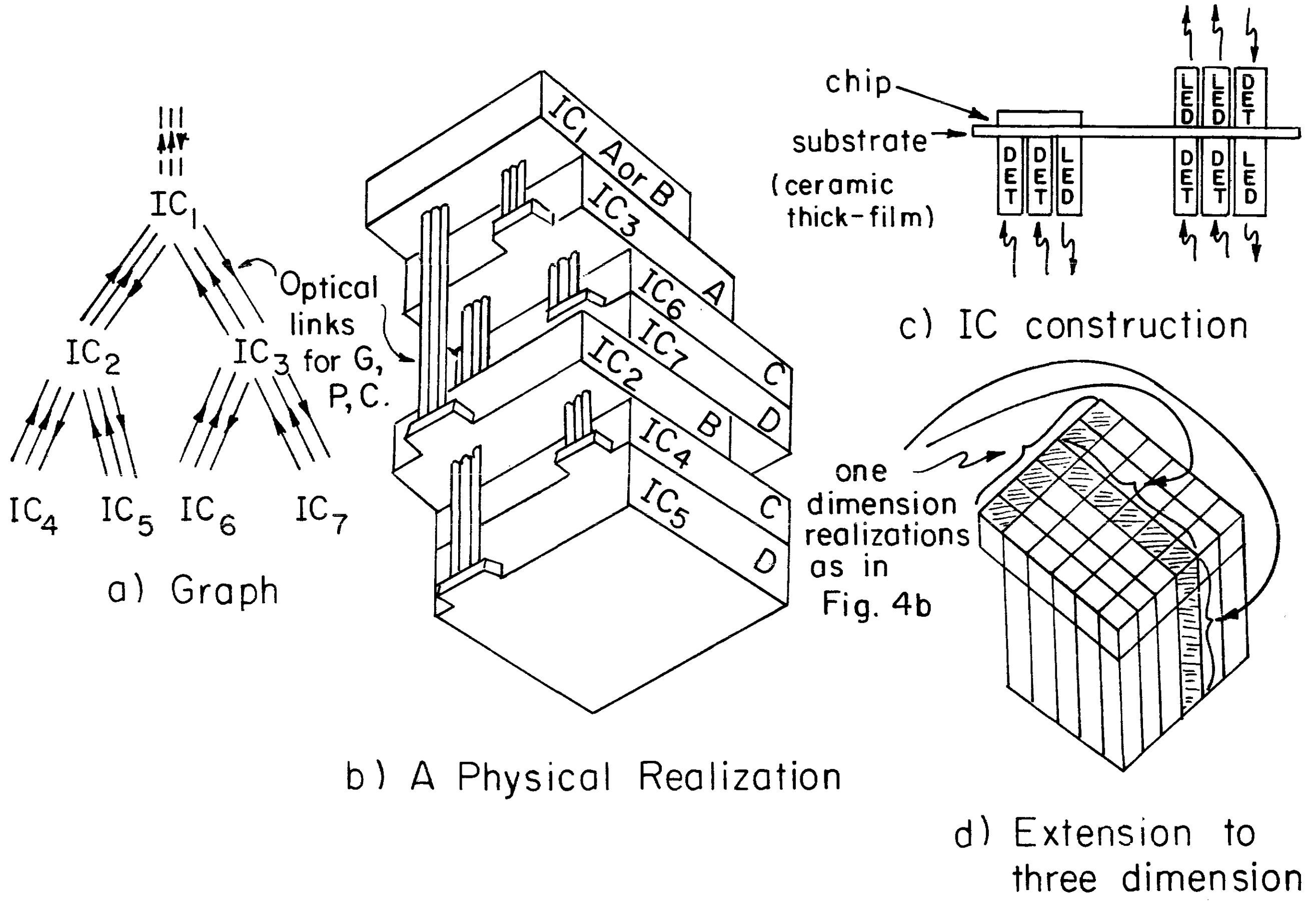


Figure 4—A tree lookahead generator

We now argue that a full binary tree lookahead generator such as described in Figures 1c, 2, and 4 should be used for fast links. The minimum delay lookahead generator (Figure 1b) has too many links between IC's. However, several tree lookahead generators can be used that require slightly more links but have much better delay properties than the ripple lookahead generator. See Figures 1c and 2. The non-leaf nodes can omit G_3, P_3 and C_3. This is conventional in the SN74182. However, having "paid" for the optical links, we should use these "free" connections like G_3, P_3, and C_3. We refer to this tree, where all nodes have G_3, P_3, and C_3 connections, as a full tree. Note that almost half of the nodes in a binary tree are non-leaf nodes. Thus, using these connections cuts the required size of the tree by half. If a nonbinary tree were used, however, this advantage would decrease. The leaf nodes are specialized, as IC types C and D are leaf node versions of A and B in Figure 4b. Since half the nodes (or more for non-binary trees) are leaf nodes, it should pay to save the extra transmitters and detectors that are not required in leaf nodes. Finally, we select a binary tree, because for a tree with fan-out f, 2f integrated circuit types are required to implement the full tree lookahead generator. However, it should be noted that a tree with fan-out f will be $\log_2 f$ times faster than a binary tree. It might be useful.

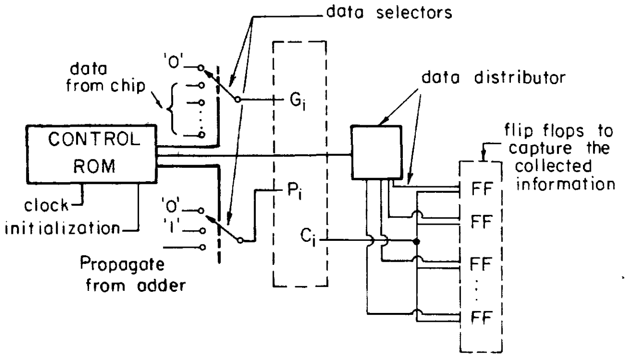


Figure 5

It should be observed that the delay in a chain lookahead generator is linearly proportional to the number of chips that are used. This can become serious. Suppose for purposes of illustration that the delay through an integrated circuit is 1 nanosecond, 1000 integrated circuits are used, and 100 time slices are required. Then 100 microseconds will be required to transmit all the information—the logic supported by the linkages will have to use a 100 microsecond clock. Consequently, the tree lookahead generator may be required. Suppose n integrated circuits are connected in a full binary tree, and realized in a three dimensional space as in Figure 4d. Then the propagation delay through space is proportional to 3 $\sqrt[3]{n}$ and the delay through gates, transmitters and detectors is proportional to $\log_2 n$. For the same illustration, the longest total gate delay (up and down the tree, which has 10 levels) is 20 nanoseconds and the delay for propagating the light through space would be only about 1.25 nanoseconds if the integrated circuits fit in a half inch cube. The logic supported by these links could have about a 2 microsecond clock. Thus, future systems might well require the tree lookahead generator.

We note also that the tree lookahead generator using the general broadcast-collector linkage or, the round-robin propagating linkage and pruning faulty cells has interesting properties and may be indicated.

End of argument

CONCLUSIONS

Changes in technology tend to aggravate the linkage problem by enabling the placement of more and more logic on a chip. However, the development of 1 GH_z Schottky detectors[1,2] and 5 GH_z MESFETS for pulse-code modulation of LED's as well as very fast (15ps transition times) Transfer Electron Logic Devices[3] invites the use of very fast optical links to solve, or at least ease, the linkage problem.

Herein, we have analyzed the use of such very fast serial linkages from the point of view of the organization (in the context of architecture, organization and realization). We have noted that some six linkage structures exist, and should be implemented in the time multiplexed slices of this fast optical link. We have shown a way to effect these using a lookahead generator. And we have shown that the lookahead generator can be implemented in two different ways using optical links. The lookahead generator is, we feel, the key to using optical links for interconnecting integrated circuits because it can realize the diverse types of linkages required for that application.

The optical link requires some consideration in the design of systems using it. Generally, chips that talk to other chips should be close by in the chain, or in the same subtree. This invites some study on algorithms that will place the chips in optimal order to most efficiently use the link. Alternatively, hierarchical organizations of modules should be designed where a module is a collection of logic that shares a single bus. This is similar to contemporary modularization where a module is a collection of logic on the same card or rack, since cards or racks are intuitively designed to keep down the number of interconnections. Thus, top-down modular design using optically linked modules should be similar to contemporary top-down modular design using printed circuit card modules and racks. Finally, the timing of transfers on the link will involve an inherent one step delay. A direct, near zero delay, link will be hard to realize since the data to be transferred will generally have to be ready at the first time slot, and all the data will finally be transferred by the last time slot in each cycle of time slots. Processes may have to be pipelined to effectively use this link. However, these considerations are not insurmountably difficult in systems design.

The optical link will have some advantages too. The connections are now programmed, so that re-connection amounts to re-programming. It will not be necessary to rewire a board to correct an error or make a field modification in the connection. Judging from the acceptance of microprocessors in digital design because programming is more flexible than re-wiring, a programmed link may be very desirable too.

ACKNOWLEDGMENTS

The author is indebted to Joe Zelikovitz and Lori Ackerman for pointing out the potential for, and devices for, an optical link between integrated circuits.

REFERENCES

1. Wang, Chitong, "Investigation of a New Grating type Gold-n-type Silicon Schottky-Barrier Photodiode for 0.4-1.1 ym Photodetection," Ph.D. Thesis, University of Florida, 1973.
2. Wang, C. T., and S. S. Li, "A New Grating type Au-n Si Schottky Barrier Photodiode," *IEEE Trans. Electron Devices,* Vol. ED-20, 1973, pp. 522-527.
3. Derman, S., "Progress in Gigabit Logic Reported for Super-fast Switching Uses," *Electronics Design*, Vol. 15, July 19, 1976, pp. 34-38.
4. General Instruments Corporation, "UART Universal Asynchronous Receiver-Transmitter," Description Flier, March 1974.
5. Bell, C. G. and A. Newell, *Computer Structures: Readings and Examples*, McGraw-Hill, 1971.
6. Nisnevich, L. and E. Strasbourger, "Decentralized Priority Control in Data Communication," *Proc. Second Annual Symposium on Computer Architecture,* January 1975, pp. 1-6.
7. Bell, G., H. McFarland, B. Delangi, J. O'Laughlin, R. Noonan, and W. Wulf, "A New Architecture for Minicomputers—The DEC PDP-11," *AFIPS Proc. SJCC,* Vol. 36, 1970, pp. 657-675.
8. Lipovski, G. J., "A Course in Top-down Design of Digital Processors," submitted to *Computer*.
9. Lipovski, G. J., "A Question of Style," submitted to the *Computer Architecture Newsletter,* Vol. 5, No. 4. pp. 32-38.
10. Lipovski, G. J., "On a Stack Organization for Microcomputers," *Proc. of the Euromicro Workshop*, Nice, France, June 1975, pp. 137-147.
11. Anderson, J. A. and G. J. Lipovski, "A Virtual Memory for Microprocessors," *Proc. Second Annual Symposium on Computer Architecture*, January 1975, pp. 80-84.
12. Lipovski, G. J., "On a Varistructured Array of Microprocessors," *IEETC*, Vol. C-26, No. 2, pp. 125-137.
13. Lipovski, G. J., "The Architecture of a Simple, Effective, Control Processor," to appear in the *Proceedings of the Second Annual Euromicro Symposium,* Venice, Italy, October 1976.
14. Gundlack, R., "Fiber Optic Developments Spark Worldwide Interest," *Electronics,* Vol. 49, No. 16, pp. 81-104, August 1976.
15. King, F. D., "High-radiance LED's Have Linear Response to Analog Inputs," *Electronics*, Vol. 49, No. 16, August 1976, pp. 92-94.
16. Love, R., "High Performance Cables Achieve Zero Failure at Rated Fensile Strength," *Electronics,* Vol. 49, No. 16, August 1976, pp. 88-89.
17. Barnoski, M., "In Systems with 20 or More Terminals, Star Couplers Outperform Tee Types," *Electronics,* Vol. 49, No. 16, August 1976, pp. 102-104.
18. Intel Corp., *Preliminary Specification of the Intel 8080,* application note.
19. Intel Corp., 8008, *8 Bit Parallel Central Processing Unit,* application note, June 1972.

UNIX on a micro-processor

by H. LYCKLAMA
Bell Laboratories
Murray Hill, New Jersey

ABSTRACT

A modified version of the UNIX Operating System has been written to run on the LSI-11 micro-computer with 20K words of primary memory and floppy disks for secondary storage. This configuration permits most of the UNIX user programs to run on the LSI-11 micro-computer. The main programming language used is the structured high-level language, C. A background process as well as foreground processes may be run. A set of subroutines has been written to interface to the file system on the floppy diskettes. Asynchronous read/write routines are also available to the user.

The LSI-UNIX system (LSX) has appeal as a stand-alone system for dedicated applications. It also has many potential uses as an intelligent terminal system. The decreasing costs of the hardware components make such a system a potential candidate for a very powerful and inexpensive personal computer system.

INTRODUCTION

The UNIX Operating System[1] has enjoyed a wide acceptance as a powerful general-purpose time-sharing system. It supports a large variety of languages and subsystems. It runs on the Digital Equipment Corporation PDP-11/40, 11/45, and 11/70 computers. These are all 16-bit word machines and have a memory management unit which makes multi-programming easy to support. The UNIX system is written in the system programming language , C.[2] In fact most user programs and subsystems are also written in this language. Other languages and subsystems supported include Basic, Fortran, Snobol, TMG and yacc (a compiler-compiler). The file system is a general hierarchical structure supporting device independence. The system runs in about 20K words of memory and user programs may be up to 32K words in size.

With the advent of the DEC LSI-11 micro-processor[3] it has become desirable to transport as much as possible of the software developed for UNIX to this machine. One of the biggest problems faced is the lack of a memory management unit, thus limiting the total address space of both system and user to 28K words. The challenge then is to reduce the 20K word operating system to 8K words and yet maintain a useful operating system. This limits the number of device drivers as well as the system functions which can be supported. The secondary storage used is floppy disks. The operating system was written in the C language and provides most of the capabilities of the standard UNIX operating system. The system occupies 8K words in the lower part of memory leaving up to 20K words for a user program. This configuration permits most of the UNIX user programs to run on the LSI-11 micro-computer. The operating system (LSX) allows a background process as well as foreground processes.

The fact that a minimum system can be configured for about $7000 makes the LSI-UNIX system an attractive stand-alone system for dedicated applications such as control of special hardware. The system also has appeal as an intelligent terminal and for applications which require a secure and private data base. In fact, this is a personal computer system with almost all of the functions of the standard UNIX time-sharing system.

This paper describes some of the objectives of the LSX system as well as some of its more important features. Its capabilities are compared with the powerful UNIX time-sharing system which runs on the PDP-11/40, 11/45 and 11/70 computers,[4] where appropriate. A number of current and planned applications are described in some detail. A summary and some thoughts on future directions are also presented.

WHY UNIX ON A MICRO-PROCESSOR?

Why develop a micro-processor based UNIX system? The increasing trend to micro-processors and the proliferation of intelligent terminals make it desirable to harness the UNIX software into an inexpensive micro-computer and give a user his own personal computer system. There are a number of factors to be considered in doing this:

1. cost of hardware
2. cost of software
3. UNIX software base
4. size of system.

The hardware costs of a computer system have come down dramatically over the last few years (even over the past few months). This trend is likely to continue in the foreseeable future. Micro-processors on a chip are a reality. The cost of primary memory (e.g., dynamic MOS memory) is decreasing rapidly as 4K-bit chips are being replaced by 16K-bit chips. There exists a large amount of expertise in PDP-11 hardware interfacing. The similarity of the Q-bus of the LSI-11 micro-computer to the UNIBUS of other members of the PDP-11 family of computers makes this expertise available.

The software development costs continue to increase since the development of new software is so labor intensive. It is difficult to estimate the cost of writing a particular software application program. Until automatic program writing techniques become widely understood and used, this trend is not likely to be turned around any time soon. Thus it becomes imperative to take advantage of as much software that has already been written as possible. A tremendous amount of software has been written to run under the UNIX operating system. It seems wise to use as much of this as possible. The operating system developed for the LSI-11 micro-computer supports most of the UNIX user programs which run under UNIX time-sharing, even though LSX is a single-user system. Thus most of the software for the system is already available, minimizing the cost of software development.

With the advent of some powerful micro-processors, the size of a computer system has shrunk correspondingly. Small secondary storage units (floppy disks) are also becoming increasingly popular. In particular, DEC is marketing the LSI-11 micro-computer which is a 16-bit word machine with an instruction set that is compatible with the PDP-11 family of computers. It is conceivable that in the next five years or so the power of a mini-computer system will be available in a micro-computer. It will become possible to allow a user to have a dedicated micro-computer rather than a part of a mini-computer time-sharing system. LSX is a step in this direction. This will give the user a cost effective interactive and powerful computer system with a known response time to requests, since the machine is not time-shared. A dedicated one-user system can be made available to guarantee "instantaneous" response to requests of a user. There are no unpredictable time-sharing delays to deal with. The system has applications in areas where security is important. A user can gain access to the system only in the room in which the system resides. It is thus possible to limit access to a user's data.

Local text-editing and text-processing features are now available. Other features can be added easily. Interfaces to special I/O equipment on the Q-bus for dedicated experiments can be added. The user then has direct access to this equipment. Using floppy disks as secondary storage gives the user a rather small data base. A link to a larger machine can provide access to a larger data base. Interfaces such as the DLV11 (serial interface) and the DRV11 (parallel interface) can provide access to other computers.

One of the main benefits of using the UNIX software base is that the C compiler is available for writing application programs in the structured high-level language, C. The use of the powerful command interpreter (sh) is also a great asset. A general hierarchical file system is available.

The LSX system has two main areas of application:

(1) control of dedicated experiments
(2) intelligent terminals.

As a dedicated experiment controller, one can interface special I/O equipment to the LSI-11 Q-bus and both support and control the experiment with the same LSX system. The applications as an intelligent terminal are manyfold:

(1) development system
(2) general text-processing applications
(3) form editor
(4) two-dimensional cursor-controlled text editor.

HARDWARE CONSIDERATIONS

The hardware required to build a useful LSI-UNIX system is minimal. The absolute minimum pieces required are:

LSI-11 microcomputer (with 4K memory)
16K memory (e.g., dynamic MOS)
EIS chip (extended instruction set)
Floppy disk controller with one drive
DLV11 serial interface
Terminal (e.g., TTY-33)
Power supply
Cabinet.

A more flexible and powerful system is shown in Figure 1. An actual total system is shown in Figure 2.

The instruction set of the LSI-11 micro-computer is compatible with that of the members of the PDP-11 family of computers with the exception of 10 instructions. The missing instructions are provided by means of the EIS chip. These special instructions may be generated by high-level compilers and it is advantageous not to have to emulate these instructions on the micro-processor. The instructions include the multiply, divide and multiple shift instructions.

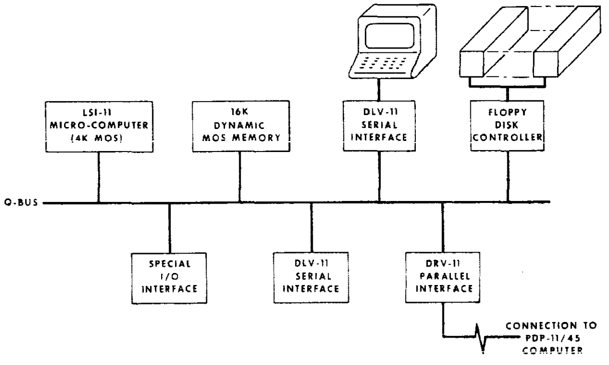


Figure 1—LSI-11 configuration

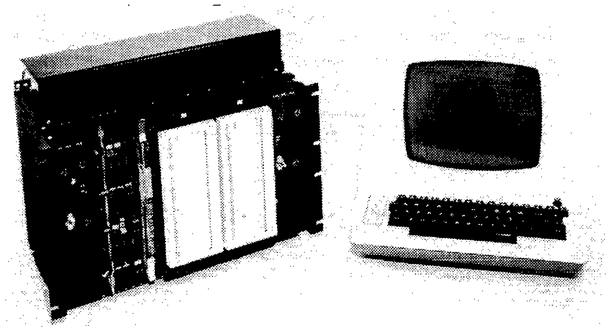


Figure 2

A floppy disk controller with up to 4 drives is shown. At present there are only a few controllers for floppy disks which interface to the LSI-11 Q-bus. The typical rotation time of the floppies is 360 RPM, i.e., six times per second. All floppies have 77 tracks, however the number of sectors and the size of sectors is variable. The comparative data for the various floppy diskettes are as follows:

	DEC	BTL	AED
sector size (bytes)	128	512	512
sectors per track	26	8	16
number of tracks	77	77	77
total capacity (bytes)	256256	315392	630784
DMA capability (y/n)	no	yes	yes
max. transfer rate (bytes per second)	6656	24576	49152

The outside vendor (AED Systems[5]) supplies dual-density drives for an increase in storage capacity. The DEC drives are IBM compatible and hence have less storage capacity. We have chosen to build our own floppy disk controller for some special Bell System requirements.[7] The advantages of DMA capabilities are obvious as regards to ease of programming and transfer rate. If IBM format compatibility is important, the throughput and capacity of the system are somewhat diminished.

At least one serial interface card is required to provide a terminal for the user of the system. Provided the terminal uses the standard RS232C interface, any terminal is satisfactory. For quick editing capabilities, CRT terminals are appropriate. For hard copy, either the common TTY33 or other terminals which run at higher baud rates may be more suitable.

The choice of memory depends on the importance of system size and whether power-fail capabilities are important. Core memory is of course non-volatile but it takes more logic boards, more space and is therefore more expensive than dynamic MOS memory. Dynamic MOS memory does not take as much space, is less expensive and takes less power, but its contents are volatile in case of power dips. Memory boards up to 16K words in size are available[6] for the LSI-11 micro-processor at a very reasonable price. The memory costs are likely to keep decreasing in the foreseeable future.

Another serial or parallel interface is often useful for connection to a larger machine with a large data base and a complete program development and support system. It is of course necessary to use such a connection to bootstrap up a system on the LSI-11 micro-computer. The central machine in this case is used to store all source for the LSX system and to compile the binary object programs required.

The system hardware is flexible enough so that, if necessary, a bus extender may be used to interface special devices to the Q-bus. This provides the ability to add special-purpose hardware which can now be controlled by the LSX system. In a later section we describe a TV raster scan terminal which was built for editing and graphics applications.[8] Other systems have interfaced special signal-processing equipment to the Q-bus. As DEC provides more of the interfaces to standard I/O peripherals, the applications will no doubt expand.

LSX FILE SYSTEM

The hierarchical file structure of UNIX is maintained. The system makes a distinction between ordinary files, directories and special files. Device independence is inherent in the system. Mounted file systems are also supported. Each file system contains its own i-list of inodes which describe the files. Each inode contains the size, number of links and the block numbers in the file. Space on disk is divided into 512-byte blocks. In contrast with the UNIX file system, two types of ordinary files are allowed. The UNIX-type file inode contains the block numbers which make up a file. If the file is larger than eight blocks, the numbers in the inode are pointers to the blocks which contain the block numbers. This requires two accesses to the disk for random file access. LSX recognizes another type of file, the contiguous file, in which the inode contains a starting block number and the number of consecutive blocks in the file. This requires only one disk access for a random access to a file. This is important for slow access devices such as floppy disks. The layout of the disk is also crucial for optimum response to commands. By locating directories and inodes close to each other, file access is measurably improved over a random distribution on disk.

There is no read/write protection on files. File protection is strictly the user's responsibility. The user is essentially given super-user permissions. Only execute and directory protection is given on files. Group id's are not implemented. File system space is limited to the capacity of the diskette in use. The list of available inodes is not dynamically created by the system (as in UNIX), but is created when the file system itself is created or salvaged.

LSX SYSTEM FEATURES

The LSX operating system is written in the C language and as such bears a strong resemblance to the multi-user

UNIX system developed for the PDP-11/40, 11/45 and 11/70 computers. The total system occupies 8K words of memory and has room for only 6 system buffers. Because the C compiler itself requires up to 12K words of user address space, it is possible to run the C compiler using only 20K words of total memory. It is possible to increase the system size if more capabilities are required in the operating system since the total memory space available to the system and user is actually 28K words. More system buffers could be provided in the system. If the system is kept to 8K words, a 20K word user program could be run. However, this requires more swap space, which is at a premium.

The system is a single-user system with only one process running at any one time. A process is defined as the execution of an image contained in a file. However, a process may fork up to two levels deep, giving rise to a total of three active foreground processes. The last process forked will run to completion first. More foreground processes could be run but this would require more swap space.

The command interpreter, the Shell, is identical to that used in the UNIX system. The file name given as a command is sought in the current directory. If not found, "/bin/" is prepended and the "/bin" directory searched. The "/bin" directory contains all of the commands generally used. Standard input, output and diagnostic files are supported. Re-direction of standard I/O is possible. Shell "scripts" are also executed by the command interpreter.

"Pipes" are not supported in the system, but pseudo-pipes are supported in the command shell. Pipes provide an interprocess communication channel in the UNIX time-sharing system. These pseudo-pipes are accomplished by expanding the shell syntax " | " to ">·_pf;<·_pf". Providing that sufficient disk space exists, the pipe implementation is transparent to the user.

The system automatically mounts a user file system on a second diskette if so desired. The "mount" and "unmount" commands are not available to the user. Thus a reboot of the system is necessary to mount a new user diskette. The system diskette is normally configured with swap space and temporary file space. User programs and files may reside on the system diskette if a user diskette is not mounted.

The size of memory available and the lack of memory protection (i.e., memory segmentation unit) have put some restrictions on the capabilities of the LSX operating system. However these are not severe in the single-user environment in which the system is run. Profiling is not provided in the system. Timing information only becomes available if a clock interrupt is provided on the LSI-11 event line at 60 times per second. Only one character device driver is allowed at present as well as only one block device driver. No physical I/O is provided for. There is also no read-ahead on file I/O. Only 6 system buffers are provided and the buffering algorithm is much simpler than in UNIX. Interactive debugging is not possible, but the planting of break-point traps and post-mortem debugging of a core image is possible. All user programs must be relocated to begin execution at 8K in memory. This required modifications to the UNIX link edit (ld) and debugger (db) programs. Most other differences between LSX and UNIX are transparent to the user.

BACKGROUND PROCESS

It is possible to run a background process on LSX while running a number of foreground processes to get some concurrency out of the system. The background process is run only while the current foreground process is in an input wait state. Two new system calls have been added to LSX, "bground" and "kill," to enable the user to run and remove a background process. Only one background process is allowed to run and it is not allowed to fork another child process; however, it may execute another program. The background process may be either compute-bound or perform some I/O functions, such as outputting to a hard-copy terminal.

STAND-ALONE ROUTINES

Under LSX it is possible to run a dedicated program (<12K words) in real time using all of the conveniences of the UNIX system calls to communicate with the file system. For programs which require more than 12K words of memory or which require more flexibility than provided by the LSX system, a set of subroutines have been written to provide the user a UNIX-compatible interface to the file system without using the LSX system calls. A user is given more control over his program. Disk I/O issued by the user is buffered using the read-ahead and write-behind features of standard UNIX. A much greater number of system buffers is provided than is possible in the LSX system. Eight of the standard file system interface routines are provided. The arguments required for each routine and the calling sequence are identical to those required by the UNIX system C-interface routines. These include: *read, write, open, close, creat, sync, unlink* and *seek.* Three unique routines: *saread, sawrite* and *statio* are provided to enable the user to do asynchronous I/O directly into buffers in the user's area rather than into system buffers. These additional routines allow a user to start multiple I/O's to/from multiple files concurrently, do some computation and then wait for completion of a particular outstanding I/O transfer at his convenience. A "load" program under LSX enables the user to load his stand-alone program which must start execution at location 0 in memory.

A DEVELOPMENT SYSTEM

One system disk has been configured to contain a fairly complete program development system. The development programs include:

editor
assembler
C compiler

link editor
debugger
command interpreter
dump program

as well as a number of libraries which contain frequently used routines for use by the link editor. It is thus possible to compile, run and debug application programs completely on-line without access to a larger machine. In a typical application, the contents of the system disk remain quite stable, whereas all user programs are maintained on a permanently mounted user diskette. For minimal systems it is possible to run with only one diskette. Due to the lack of protection, it is possible to crash the system. However in practice, the use of the high-level language C minimizes the number of fatal bugs which actually occur, since the stack frame and program counter are quite well controlled.

In our particular installation, it is often convenient to use the satellite processor system[9] to aid in the running and debugging of new user programs. This is possible since programs running in the LSI-11 satellite microcomputer behave as if they are running on the central machine with access to its file system. This emulates the environment on LSX quite closely. Thus a program may be compiled on a central machine supporting the C compiler, run on the LSI-11 micro-computer and debugged. When the program has been completely debugged, it is possible to load the program onto the floppy file system using the stand-alone routines (described previously) and the satellite processor system. The program may then be run under LSX.

TEXT PROCESSING SYSTEM

Another area of application for the LSX system is as a personal computer system for text processing. Files may be prepared using the editor and run off using the UNIX nroff command with a hard-copy device. This system disk includes programs such as:

editor	
cat	output ascii files
pr	print ascii files
od	octal dump files
roff	
nroff	
neqn	mathematical equation FORMATTER

The file transfer program referred to in the previous section enables one to transfer files to/from a machine with a larger data base. A user's files may be maintained on his personal mounted diskette. If a hard-copy device is attached to the computer as well as the user's interactive terminal, hard-copy output can be obtained using a background process while editing another file in the foreground.

FORM EDITOR PROGRAM

Another area of application for which LSX seems well-suited is for the entry and retrieval of data records by computer-naive users. We have available an intelligent terminal[8] which has some advanced features particularly suitable for this application. It is shown in Figure 3. It has scrolling capabilities, cursor control and user labelled buttons below the TV screen. The buttons can be used for cursor positioning as well as other dedicated functions defined by a user program. This terminal is well-suited for the input of data into computer-displayed forms. Protected fields are implemented in software rather than in hardware as for the TTY40 terminal. A general-purpose form entry program has been written for this terminal.[10] Another program "mkform" is available to enable a user to compose a form on the TV screen interactively. The form is then used with the "fentry" program to create, update and delete entries in a data base whose record structure depends only on the structure of the form and is independent of the "fentry" program. The LSX system provides the underlying support and file system for these programs. The programs are designed to be very easy to use for computer-naive users.

TWO-DIMENSIONAL SCOPE TEXT EDITOR

The TV terminal described above is also well-suited for a two-dimensional text editor. The interactive two-dimensional cursor control features allow one to move the cursor anywhere on the face of the TV screen. The editor available on the UNIX system has some very powerful features. It is desirable to use these in the scope editor as well. Therefore the scope editor features have been imbedded in the existing UNIX text editor. Thus the user is capable of going back and forth between the standard UNIX editor features

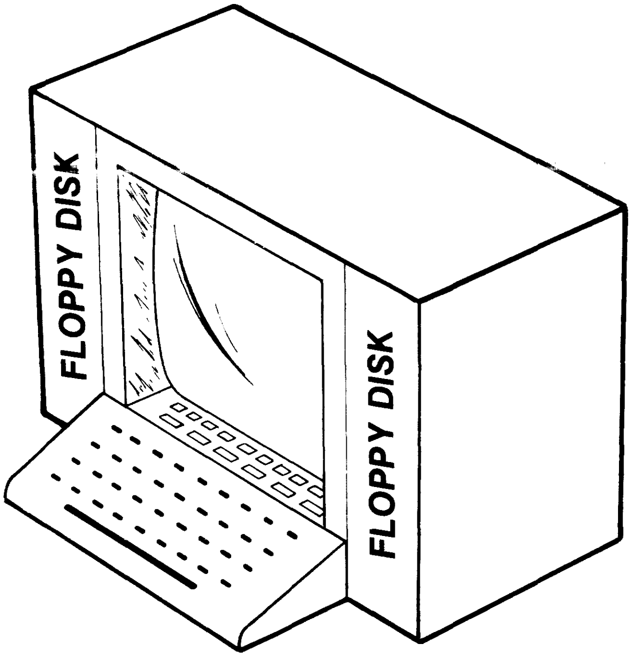


Figure 3

and the additional scope editing features. The labels on the buttons below the TV screen tell the user what mode he is in and what functions are available to him. Complete cursor control is available. A window into the file being edited is displayed on the TV screen. The user has the ability to insert, remove and replace a character at the current cursor position or delete the remainder of the line to the right of the cursor on a per line basis. The user may also insert or delete whole lines or break a line in two. Block deletes, copies and moves are also available by means of three marks which may be set in a file. A position command is available to move any section of the file onto the TV screen window (26 lines). Further work is being done on this editor.

SUMMARY

The LSI-UNIX system is currently being used for research in intelligent terminals and in stand-alone dedicated systems. There are plans to use this system for further research in other areas of Bell Laboratories. Hard-copy features have yet to be incorporated into the system in a clean fashion. Currently, our system is connected to a larger machine using the Satellite Processor System. More general connection to larger machines or possibly to a network of machines has yet to be investigated. The LSX system also has potential uses in multi-terminal or cluster control terminal systems where multi-tasking features are important. These application areas have only been looked at superficially and warrant further investigation.

As a development system, LSX functions quite well. The response to most programs is only a factor of four or so slower than on the conventional mini-computers. This is due mainly to the slow secondary storage devices used by LSX. Optimization of file storage allocation on secondary should improve response somewhat.

The advent of large memory boards (64K words) will make the mapping of memory necessary to take full advantage of this large address space. This will enable the running of multiple processes without the need for swapping a process out of primary memory. This should also improve the response of the system and increase the number of uses to which it can be put.

There is a necessary loss of some functionality in the LSX system because of the size of the memory address space available on the LSI-11 computer. However as a single user system, most of the functions are still available to the user. As an intelligent terminal system, a microprocessor with all of the UNIX software available is indeed quite a desirable "intelligent" terminal.

ACKNOWLEDGMENTS

The author is indebted to H. G. Alles for designing and building both the initial PERTEC floppy disk controller and the novel TV terminal. These two pieces of hardware have provided much of the motivation for doing the LSX system in the first place and for doing research in the area of intelligent terminals in particular. Many of the application and support programs described here have been written by Eugene W. Stark. John S. Thompson wrote a floppy disk driver for the AED floppy disk controller to facilitate bringing up the LSX system on these disks. The author is grateful to J. C. Swartzwelder and D. R. Weller for their efforts in putting together the first LSI-11 system.

REFERENCES

1. Thompson, K. and D. M. Ritchie, "The UNIX Time-Sharing System," *Comm. ACM* 17, July 1974, p. 365.
2. Ritchie, D. M., "C Reference Manual," Bell Labs Memorandum.
3. DEC LSI-11 Processor Handbook, 1976.
4. Thompson, K. and D. M. Ritchie, "UNIX Programmer's Manual—6th Edition," May, 1975.
5. Advanced Electronic Design Double-Density Floppy Disk Subsystem.
6. Monolithic Systems.
7. Alles, H. G. "An LSI-11 Controller for the Pertec Floppy Disk," Private communication.
8. Alles, H. G. "A TV Terminal for the LSI-11 Microcomputer," Private communication.
9. Lycklama, H. and C. Christensen, "Emulation of UNIX on Peripheral Processors," Bell Labs Memorandum.
10. Stark, E. W., "System for Entering Data Through Computer-Displayed Forms," Private communication.

Using LSI processor bit-slices to build a PDP-11—A case study in microcomputer design*

by T. M. McWILLIAMS, S. H. FULLER and W. H. SHERWOOD
Carnegie-Mellon University
Pittsburgh, Pennsylvania

ABSTRACT

In this article we give the design and evaluation of the CMU-11: a fully operational implementation of the PDP-11 computer architecture built with Intel 3000 Schottky bipolar microcomputer bit-slices. This project was initiated to test in detail the claims that LSI processor bit-slices simplify the design of microprogrammed processors. The CMU-11 executes approximately 240,000 instructions per second, which is about 63 percent the speed of the PDP-11/40 and twice the speed of the LSI-11.

We explore in some detail the additional logic that was added to enable the Intel 3000 circuits to emulate the PDP-11 instruction set. We specified full DEC Unibus compatibility and 29 percent of the integrated circuits used to implement the CMU-11 were required to provide buffering and control of the Unibus. The other main sources of inefficiency were the lack of arithmetic overflow logic in the bit-slices and the organization of the microinstruction control store. We show how improved LSI circuits in this area can substantially reduce the size (and cost) of the processor.

The set of design aids currently available at Carnegie-Mellon University was of critical assistance in this project and we include a critique of our use of these design aids to show their utility in prototype design efforts.

INTRODUCTION

Several semiconductor manufacturers have recently developed high speed LSI circuits that are designed to simplify the construction of microprogrammed processors and device controllers. These integrated circuits are called "bit-slices" because they implement two or four bits of the registers, arithmetic units, and primary data paths of a processor. This article presents the design and evaluation of the processor built at Carnegie-Mellon University that uses the Intel 3000 bit-slices and that is microprogrammed to emulate the PDP-11 computer architecture.[1,2]** The purpose of this project was to investigate the assertions of semiconductor manufacturers that their LSI bit-slices would in fact simplify the design and construction of processors.

Rather than specify a new architecture (i.e., instruction set) for this experiment in processor design, we decided to reimplement an established computer architecture: the PDP-11. We chose the PDP-11 architecture for several reasons. Using an existing and well-known architecture would allow others to more easily evaluate the results of our experiment and kept us from consciously or unconsciously tailoring the processor architecture to fit the capabilities and idiosyncrasies of the LSI bit-slices. Another reason is that PDP-11's are in extensive use at Carnegie-Mellon Univ. in a wide variety of applications and, if our experiment was successful, the processor could be put to work on any one of several practical tasks. It was this second reason that helped establish a criterion that proved to be critical: we demanded that the processor we constructed support the standard DEC Unibus[3] that is common to all PDP-11's except the LSI-11. Finally, the PDP-11 architecture is an unusually good test of the capabilities of a bit-slice circuit family because it is a relatively complete architecture with numerous addressing modes and instruction formats.

In the next section we begin with a description of the design of the CMU-11 processor. We then discuss the performance, cost and implementation difficulties uncovered during the design and testing of the machine. In addition to the evaluation of the LSI bit-slice circuits for general-purpose processors, we are interested in the problems of computer design in general. For this reason, a fairly complete set of digital design automation aids are available at Carnegie-Mellon University: an interactive drawing package that generates engineering drawings, wire-lists, and aids in engineering changes; a digital simulation system that is interfaced to the drawing system; and microprogram assemblers. A later section of this paper reviews our

* This work was partially supported by the Advanced Research Projects Agency (ARPA) of the Department of Defense under contract F44620-73-C-0074, monitored by the Air Force Office of Scientific Research.

** We gratefully acknowledge the donation of 3000 microcomputer sets by both Intel and Signetics Corporations.

experiences with these design aids and we draw some conclusions concerning the process of designing and debugging prototypes of digital systems built with LSI circuits.

ORGANIZATION OF THE CMU-11

Figure 1 is a register-transfer level diagram of the CMU-11 microprogrammable processor. The processor's components are arranged in the diagram into three sections: the *data part, control part,* and *Unibus interface*. We were able to build the entire processor on a single board and Figure 2 is a top view of the CMU-11.

The data paths and working registers

The data part of the processor is designed around the 3002 (central processing element) bit-slice. A single 3002 circuit implements a 2-bit slice of the data paths and hence eight 3002's have been used in the CMU-11. Although not explicitly shown in Figure 1, the 3003 carry-lookahead circuit is also used. With the 3003, the 3002 array is capable of cycling through operations every 150 ns. However, other delays in the clock and control part dictate that the CMU-11 has a 200 nsec micro-cycle time. The eight general-purpose working registers of the PDP-11 architecture can be kept in the register scratchpad on the 3002's, and the three remaining internal registers, R8, R9, and T are sufficient for source and destination operand computations as well as other intermediate results. The Program Status (PS) and Instruction Register (IR) were not possible to locate within the 3002's without a severe loss in performance.

The relatively generous number of input and output lines of the 3002's are used to good advantage. The D⟨15 : 0⟩ and A⟨15 : 0⟩ buses feed the Unibus Data and Address lines respectively. In addition, the D bus allowed access to the extra data paths necessary to include the PS register and to facilitate the byte swap operation needed by many of

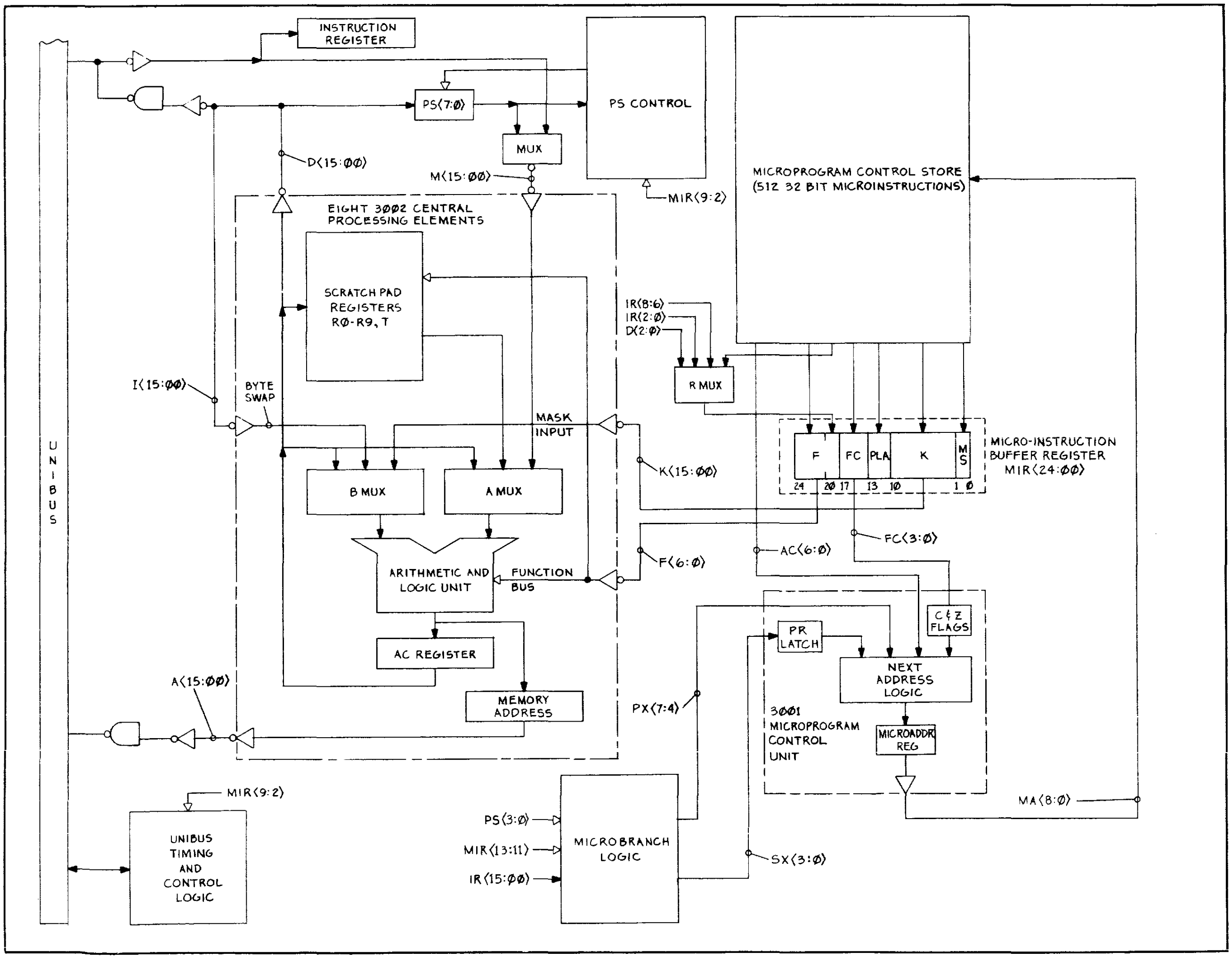


Figure 1—Register transfer level diagram

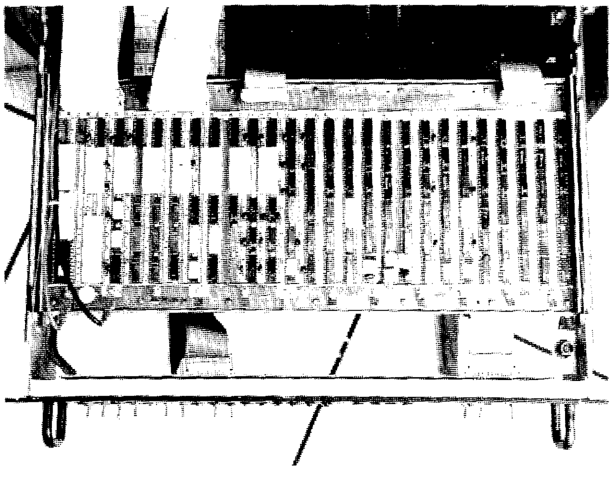


Figure 2—CMU-11 processor board

the PDP-11's instructions. The M⟨15 : 0⟩ bus is used as the principal data input bus. The Function bus, F⟨6 : 0⟩, specifies both the operation to be performed by the arithmetic/logic unit as well as the selection of the register in the scratchpad to be involved in the operation. The K⟨15 : 0⟩ bus is used to input masks or constants from the microinstruction. The 3000 circuit set makes frequent use of the K lines to specify masks (usually all zeros or all ones) that effectively extend the operation code on the Function bus.

Control part

The control part of the CMU-11 uses the 3001 Microprogram Control Unit and a 512 word control store** with 32 bit microinstructions. Figure 4 shows the format of the microinstruction and Table I briefly describes the function of each of the fields. A microinstruction buffer register was included in the design to allow the overlap of the fetch of the next microinstruction with the execution of the current microinstruction which is a common technique to improve the performance of microprogrammed processors.

The "next-address logic" of the 3001 has been augmented by additional microbranch control logic external to the 3001. This external logic uses the contents of the Instruction Register, the condition codes in the PS, and the PLA field from the microinstruction register to determine the AC⟨6 : 0⟩ lines to input to the 3001.

The other major section of control logic that had to be added to the design was the Processor Status logic to control the setting of the 4-bit condition code in the PS register and control access to the PS. In fact, the PS register is defined as primary memory location 177776 in the PDP-11 architecture requires special logic to load and store the PS.

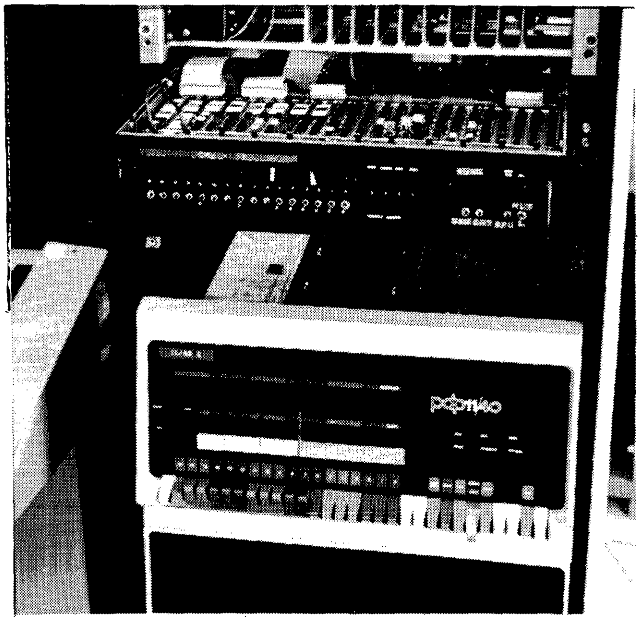


Figure 3—CMU-11 system with associated PDP-11

Interface to the unibus

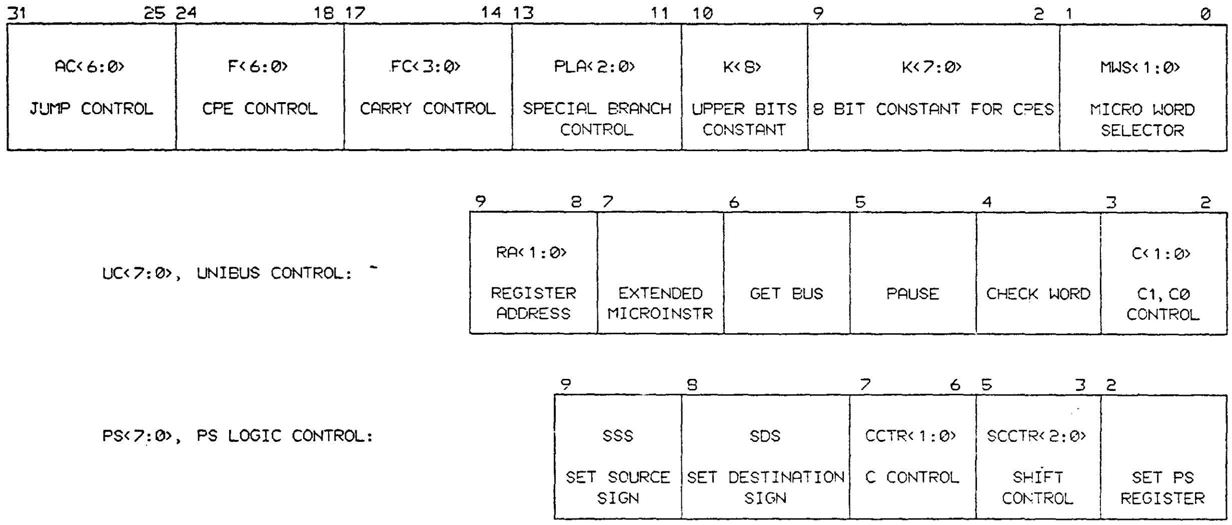
A significant fraction of the components of the CMU-11 are devoted to the support of the Unibus. Given the demanding electrical requirements of the Unibus, the tristate A, D, and M lines of the 3002 array could not be directly attached to the Unibus. Instead, separate transceiver packages had to be used to provide this buffering.

Due to the asynchronous operation of the Unibus and interrupt and non-processor requests (i.e., direct-memory access request via the Unibus) it was not practical to drive the Unibus directly from fields in the microinstruction. Instead, a bus control and timing section added to the processor. The rest of the processor interfaces to this control unit via the UC⟨7 : 0⟩ field in the microinstruction. See Table I for a description of the functions of the subfields within UC⟨7 : 0⟩.

Console functions

In place of a standard front panel, the CMU-11 has front panel functions accessible from a standard teletype attached to the Unibus. Memory locations can be examined and loaded by typing the octal address followed by a slash. The current value is displayed and a new value may be entered if desired, followed by a carriage return. The processor may also be started and continued from the teletype and there is a halt switch on the front panel which causes the machine to return to the console microprogram.

** In order to expedite the debugging of the microprogram for the CMU-11, we built a fast, simple writable control store for the CMU-11. 45 nsec access time, 1024 bit RAM packages were used to assure a writable control store as fast as the final ROM control store. The writable control store is interfaced to a Unibus (of a PDP-11 other than the CMU-11) for initial loading of microprograms. Figure 3 shows the CMU-11 interfaced to the supporting PDP-11 and writable control store.



31 25	24 18	17 14	13 11	10	9 2	1 0
AC<6:0> JUMP CONTROL	F<6:0> CPE CONTROL	FC<3:0> CARRY CONTROL	PLA<2:0> SPECIAL BRANCH CONTROL	K<8> UPPER BITS CONSTANT	K<7:0> 8 BIT CONSTANT FOR CPES	MWS<1:0> MICRO WORD SELECTOR

UC<7:0>, UNIBUS CONTROL:

9 8	7	6	5	4	3 2
RA<1:0> REGISTER ADDRESS	EXTENDED MICROINSTR	GET BUS	PAUSE	CHECK WORD	C<1:0> C1, C0 CONTROL

PS<7:0>, PS LOGIC CONTROL:

9	8	7 6	5 3	2
SSS SET SOURCE SIGN	SDS SET DESTINATION SIGN	CCTR<1:0> C CONTROL	SCCTR<2:0> SHIFT CONTROL	SET PS REGISTER

Figure 4—Microinstruction format

This use of a teletype for a console is similar to the console teletype used by the LSI-11.[4] In order to make it easier to maintain the processor, we have added a microprocessor console which displays the microprogram address and allows the microprocessor to be single-stepped. The microconsole proved invaluable for debugging the prototype processor.

EVALUATION OF CMU-11 DESIGN

The critical questions to be asked about this design concern cost and performance. It has been fairly easy to evaluate the performance of the CMU-11 by looking at several representative instruction times and by running a set of benchmarks on the machine. Evaluating the cost of the CMU-11 has been more difficult. Rather than try to compare the price of existing PDP-11 implementations with the cost of the CMU-11, we chose instead to compare it with other PDP-11's with respect to circuit complexity. The other significant costs, i.e., development costs, are discussed in a later section.

Performance of the CMU-11

The CMU-11 runs at a microinstruction cycle time of 200 nsec. The specifications for the Intel 3000 microcomputer

TABLE I—Description of Microinstruction Fields

MWS⟨1 : 0⟩ := MI⟨1 : 0⟩	*Micro Instruction Selector.* Specifies if MI⟨9 : 2⟩ should define a constant, unibus control, or PS control.
K⟨8 : 0⟩ := MI⟨10 : 2⟩	*Literal.* K⟨7 : 0⟩ is a byte constant used by the least significant byte of the K input lines of the 3002 array. K⟨8⟩ is extended to feed the most significant byte of the K input lines.
UC⟨7 : 0⟩ := MI⟨9 : 2⟩	*Unibus Control*
UC⟨1 : 0⟩	*C1, C0 Control.* Specified the C1 and C0 lines on the Unibus.
UC⟨2⟩	*Check Word.* Tests whether a word address is specified in Unibus operation.
UC⟨3⟩	*Pause.* Halt processor clock until completion of Unibus operation.
UC⟨4⟩	*Get Bus.* Request access of Unibus for a data transfer.
UC⟨5⟩	*Extended Micro Instruction Code.* If set, defines alternate meaning for PLA⟨2 : 0⟩.
UC⟨7 : 6⟩	*Register Address.* Specified which input register address multiplexor should be used.
PS⟨7 : 0⟩ := MI⟨9 : 2⟩	*Processor Status Control*
PS⟨0⟩	*Set PS Register.* Controls loading of PS.
PS⟨3 : 1⟩	*Shift Control*
PS⟨5 : 4⟩	*Carry Control*
PS⟨6⟩	*Set Destination Sign.* Controls latching of sign of destination operand in flag external to 3002's.
PS⟨7⟩	*Set Source Sign.* Analogous to PS⟨6⟩.
PLA⟨2 : 0⟩ := MI⟨13 : 11⟩	*Special Branch Control.* Used by microbranch logic to tell which fields of IR and PS to examine for branch conditions.
FC⟨3 : 0⟩ := MI⟨17 : 14⟩	*MCU Flag Control.* Controls testing and setting of flags in 3001 (MCU).
F⟨6 : 0⟩ := MI⟨24 : 18⟩	*CPE Control.* Drives Function Bus of 3002 (CPE) array.
AC⟨6 : 0⟩ := MI⟨31 : 25⟩	*Address Control.* Connected directly to the AC⟨6 : 0⟩ bus of the 3001 (MCU). This is the one field of the micro instruction not buffered in the micro instruction register. (The Microprogram Address Register internal to the MCU performs the buffering function.)

family state that it is possible to build a 16 bit minicomputer with a 150 nsec. cycle time. However, given our objective to design as cost-effective an implementation as possible, we avoided the sensitive and complex timing circuits that would be required to approach a 150 nsec. cycle time.

If we had used clocks with sufficient buffering and pulse shaping, a worst-case analysis shows that with the particular IC packages used in the CMU-11, we could approach a 149 nsec. cycle time with Intel 3000 packages and a 126 nsec. cycle time with Signetics' version of the 3000 set. We have in fact replaced the Intel 3000 circuits with the Signetics circuits and although the CMU-11 continues to run reliably at 200 nsec., we cannot reduce the cycle time below 200 nsec.: the critical path is in the control part and not the 3002 array.

Tables II and III show the execution time for six of the most frequently executed instructions and the eight addressing modes of the PDP-11. The instructions in Table II assume a register-to-register operation (i.e., a source and destination mode of 0). Table III shows the additional time that is added to the instruction execution time for the various source addressing modes.† The destination mode times are about the same as the given source mode times.

In order to measure the performance of the CMU-11 for various instruction mixes, several benchmarks were collected and run on the CMU-11, an LSI-11, and a PDP-11/40. Four benchmarks were collected that attempt to span a reasonable range of applications common to minicomputers:

Quicksort. This is a program that uses Hoare's quicksort procedure to sort a set of 16 bit integers. The benchmark also includes a pseudorandom number generator to provide the initial data.

Trigonometric Functions. A set of trigonometric, floating-point routines. We do not assume the existence of a floating point option on any of the processors and hence this benchmark heavily exercises software floating point emulation routines.

Partial Differential Equations. A program that uses a straightforward iterative relaxation technique to solve a partial differential equation over a two-dimensional space. Fixed-point values are used.

Text Searching. Searches an input string for names in a symbol table. This benchmark makes extensive use of the byte and compare features in the instruction set.

TABLE II—Execution Times of Common Instructions

	Basic execution time (microseconds)		
Instruction	LSI-11	CMU-11	PDP-11/40
MOV	3.50	2.06	0.90
CMP	3.50	2.19	0.99
ASL	3.85	2.46	0.99
ADD	2.46	3.85	0.99
BRX (branch)	3.50	2.82	1.76
(no branch)	3.50	1.48	1.40
JSR	6.40	4.39	2.94

Table IV shows the execution times on the LSI-11, CMU-11, and PDP-11/40 for each of the four benchmarks. From these results we see the CMU-11 is approximately twice as fast as the LSI-11 and 63 percent of the speed of the PDP-11/40. As expected, there is a moderate amount of variation in the relative performance of the three machines for the different benchmarks. The two dominant effects that can be seen in Table IV are that the PDP-11/40 design has optimized register-to-register operations more than either the LSI-11 or the CMU-11 (as demonstrated in the partial differential equation benchmark). Byte operations are more efficiently performed in the CMU-11 because of its byte-swap data path provided by the D and I buses. The last line in Table IV is the data published by O'Loughlin[5] in an article comparing the different DEC PDP-11 implementations.

It is mildly disappointing that the CMU-11, built with Schottky TTL bit-slices, could not equal the performance of the PDP-11/40, built with standard TTL circuits. The next two sections will examine in detail where performance was lost (and gained) in the CMU-11 design. Before continuing with this review of the design, we turn to a brief discussion of the cost of the CMU-11.

A principal objective of the 3000 microcomputer bit-slice packages is to simplify the design of processors like the CMU-11. Table V is a summary of the complexity (measured in integrated circuits) of the CMU-11. There are two columns in Table V. A simple count of the number of integrated circuit packages used in the CMU-11, and a column that converts the design to "16-pin equivalent" packages (a measure of the size of the design in a standard

TABLE III—Execution Times for the Source Addressing Modes

Addressing mode	LSI-11	CMU-11	PDP-11/40
0: Register	0.00 μsec	0.00 μsec	0.00 μsec
1: Register Deferred	1.40	1.21	0.78
2: Autoincrement	1.40	0.64	0.84
3: Autoincrement Deferred	3.50	1.91	1.74
4: Autodecrement	2.10	1.00	0.84
5: Autodecrement Deferred	4.20	2.28	1.74
6: Indexed	4.20	1.78	1.46
7: Indexed Deferred	6.30	2.99	2.36

† In particular, the times in Table III are the source addresses modes time for the CMU-11 as measured on the BIS instruction. Addressing times on the other instructions are similar to the BIS times.

TABLE IV—Performance of CMU-11 Relative to Other PDP-11's

	Execution times relative to PDP-11/40*					
Benchmarks	LSI-11	11/10	11/20	CMU-11	11/40	11/45
Quicksort	2.88 (366)			1.48 (188)	1.0 (127)	
Partial Diff. Eqn.	3.48 (268)			1.75 (135)	1.0 (77)	
Trig. Functions	3.36 (111)			1.57 (52)	1.0 (33)	
Text Searching	2.76 (204)			1.45 (107)	1.0 (74)	
Average	3.1	—	—	1.6	1.0	—
O'Loughlin's Data	—	2.32	1.85	—	1.0	0.91

* Numbers in parentheses are the absolute run times in seconds for the benchmarks.

unit). Table VI gives a breakdown of the actual cost of the CMU-11 at January, 1976 prices.

It is surprising that less than 20 percent of the design is now in the data part of the processor: the part of the processor largely implemented with the LSI bit-slices. A larger part of the design, 29 percent, is needed just to interface to the PDP-11 Unibus.

In order to put the 144 package complexity of the CMU-11 in perspective, the IC package counts for other PDP-11's are: PDP-11/10—203 packages; PDP-11/40—417; and PDP-11/45—696. The LSI-11 is able to implement the basic processor in 42 packages but does not interface to a Unibus. It is clear that the bit-slices do not approach the economy of the Western Digital NMOS microcomputer circuits which were specifically designed to emulate the PDP-11.

Another measure of the degree to which the CMU-11 processor can efficiently emulate the PDP-11 architecture is given by the size of the microprograms. Table VI gives the size of microprograms for several PDP-11 processors. It is somewhat surprising that the CMU-11 uses fewer bits in its control store than any of the other processors except the LSI-11. This is in large part due to the fact the 11/10, 11/40, and 11/45 use MSI arithmetic/logic packages that did not have as useful a set of primitive operations as the 3002 ALU.

TABLE V—Integrated Circuit Statistics

Processor component	No. IC packages	No. 16 pin equivalent packages	
DATA PART			
3002 (CPE) Array	8	20	
PS and Instruction Registers	6	6	
Misc.	4	5	
	—	—	
subtotal	18	31	(19%)
CONTROL PART			
Control Store ROMs	8	8	
Micro Instruction Register	10	10	
3001 (MCU)	1	3	
Microbranch logic	26	27	
PS Control	16	16	
Misc.	18	18	
	—	—	
subtotal	79	82	(52%)
UNIBUS INTERFACE			
Bus Tranceivers and Inverters	19	19	
Unibus Control	28	28	
	—	—	
subtotal	47	47	(29%)
Total	144	160	

SOME PITFALLS FOUND IN IMPLEMENTING THE PDP-11 with the 3000 BIT-SLICES

Since the CMU-11 project was started, a number of different bit-slice chips have become available whose orga-

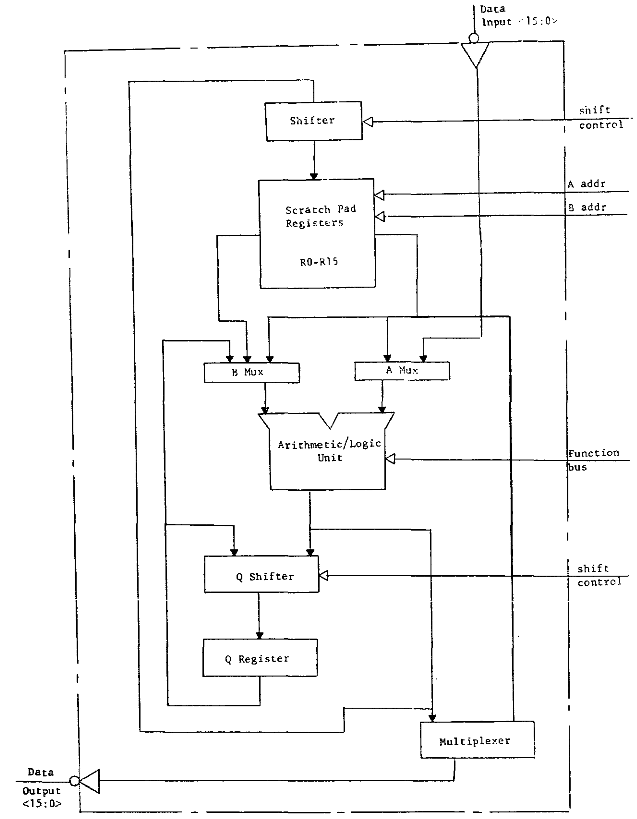


Figure 5—The Am2901—A 4 bit bipolar microprocessor slice

TABLE VI—Cost Breakdown for CMU-11

	Prices	
Components	Single Units	Quantities of 100+
LSI Microcomputer parts	$207	$125
(Intel 3001, 3002's, 3003)	(184)*	
PROMS		
(3601, 3602, 3604, 745168)	204	136
SSI/MSI Parts	179	158
	—	—
Integrated Circuit Subtotal	590	419
Augut Wirewrap Board	379	(use printed circuit)
Wirewrapping	107	—
Total	$1076	—

* Signetics prices

nizations are significantly different from the 3000 circuits and which provide an interesting contrast. Two of the more interesting bit-slice chips are the Advanced Micro Devices Am2901 and the Monolithic Memories Inc. MM16701. These bit-slice chips have a very similar data path organization with only minor differences, the Am2901 being the faster device. Because of the similarity of these devices, we will limit the discussion here to the Am2901, but all of the microinstruction sequences discussed will work on both bit-slice sets.

The basic data path of the Am2901 is shown in Figure 5. The chip contains a register file of 16 4-bit accumulators and an accumulator extension register, the Q register. In one microinstruction, two operands can be read out of the register file, passed through the ALU, the result can be written shifted left or right, and written back into the register file. In parallel with this, there is an addressing mode which controls the RAM and Q shifters allowing the output of the ALU and the Q register to be right shifted simultaneously, which is well suited for the inner loop of multiply or divide instructions.

I/O Buses

The main advantage of the 3000 bit-slice over the Am2901 is its five fully parallel data buses for transferring data in and out of the chip. It has two tri-state output buses (the A and D buses) and three input buses (M, I, and K). If the minicomputer to be emulated has a fairly short I/O and memory buses, the 3000 buses can directly drive them, resulting in a substantial savings in bus driver packages. In the CMU-11, we needed to drive a DEC Unibus, so we had to use separate bus drivers and receivers. Once external bus drivers are added, the advantage of the two output buses for the address and data is minimal, because an equivalent external address register can be loaded as fast as the existing internal address register and combination bus drivers/latches are available (e.g., Am2905). The savings realized by having three input buses is the cost of adding eight dual 4-to-1 line multiplexer chips at the input to the bit-slice chips. The saving achieved with the five buses in the 3000 bit-slices over the Am2901's single input and single output bus is 12 16-pin circuits, plus three bits in the control store (two for the select lines on the input multiplexer, and one to control loading of the address register).

Arithmetic overflow with the 3000

One of the biggest problems encountered with the PDP-11 implementation using the 3000 bit-slice was detection of arithmetic overflow. The 3000 bit-slice has no overflow output and the signals needed to directly detect overflow are not available at the external pin connections. This results in considerable overhead in emulating instructions which must detect overflow (e.g., instructions that set the V bit in the PS register of the PDP-11). The CMU-11 overflow handling was implemented with two external flip-flops which contain the signs of the source and destination operands. After an instruction is fetched its operands are first fetched either from memory or from the register stack, and are put in the source and destination registers within the 3002. As the operands are fetched, the source and destination flip-flops are set to the signs of the operands. When an instruction is executed the overflow logic can use

TABLE VII—PDP-11 Control Store Sizes

LSI-11	PSP-11/10*	CMU-11	PDP-11/40*	PDP-11/45*
22 bits × 512 words (includes console)	40 bits × 239 words	32 bits × 287 words (without console) 414 words (with console)	56 bits × 251 words	64 bits × 256 words

* [O'Loughlin 1975]

the signs of the operands and result to detect overflow. This technique works well when the operands are from memory, but really slows down the register-to-register operations because the operands have to be moved to the AC so their signs can be latched in the external source and destination sign flip-flops.

The sequence of instructions needed to emulate a register-to-register add is shown in Figure 6. The first instruction in the sequence loads the source operand into the AC, in order to get its sign out of the chip. The next instruction specifies for the source sign flip-flop to be set to the sign of the AC, and to store the AC into the T register. The following two instructions load the destination operand into the AC and set the destination sign flip-flop. The last two instructions do the add and store the result back in the destination register. Because of the multiple use of fields in the microinstruction it is not possible to specify that a register address comes from the instruction register in the same microinstruction that sets either the source sign or destination sign flip-flops, or which sets the condition codes. If the microprocessor were to be redesigned to allow this, the register-to-register add could be done in three rather than six microinstructions with the 3000 chips. However, we would pay for this performance improvement by having to use a wider microinstruction. The Am2901 provides external access to the overflow detect output on the chip and the register-to-register add can be done with only one microinstruction, resulting in a considerable speed increase over the 3000 chips.

Example of a multiply instruction

The inner loop of a 16 bit integer multiply instruction on the 3000 chips requires either three or six microinstructions, depending on whether that cycle is a double register shift and add, or just a shift. The high order word of the product is stored in the AC register, and the low order word is stored in the T register. Initially, AC is zero, and T holds the multiplicand. For each iteration of the multiply, the loop count is decremented and if the low order bit of the T register is a 1, then the multiplier is added into the AC and the AC and T registers are shifted right. Because the 3000 cannot add a register to the AC without also putting the result in the register, it takes three microinstructions to perform the inner loop addition.

For the Am2901, the inner loop of the multiply can be done in two microinstructions with no external loop counter, and in one with an external counter. This is possible because the Am2901 in one microinstruction can add two general registers together, shifting the result and the accumulator extension register right one bit. A similar speedup also occurs for division.

```
ILR     SR              ;AC←Source Register
SDR     T, 1, SETSS     ;T←AC and SET Source Sign
ILR     DR              ;AC←Destination Register
NOP     SETDS           ;SET Destination Sign
ALR     T, SETCC        ;AC=AC+T and SET Condition
                        Codes
SDR     DR, 1           ;Destination Register←AC
```

Figure 6—Microsequence example: Register-to-register add with overflow detect

ADDITIONAL COMMENTS ON THE CMU-11 DESIGN

The 3000 microcomputer circuits are not the only area in which to look for improvements in the CMU-11 design. A major source of complexity was the Unibus interface (29 percent of processor's packages). The 3002 bit-slices provide tri-state drivers for their A and D lines and if Unibus compatibility is not essential, the outputs from the 3002 circuits could directly drive a memory and I/O bus of moderate size. If synchronous operation of the memory bus is adequate, further simplification of the bus interface section of the processor is possible.

A number of integrated circuit packages are now available that could help simplify the design of the control part of the processor. Most significantly, 4K bit PROM's appropriate for use in the control store are now available with internal latches for use as a microinstruction buffer. This would eliminate the need for the separate latches used in the CMU-11's microinstruction register. A related optimization to the CMU-11 would be to move from the partly encoded microinstruction format of the CMU-11 to a wider, fully horizontal format. The random logic needed to decode an encoded microinstruction is simply more expensive than the extra bits in the control store needed for the horizontal format.

We attempted to use programmable logic arrays (PLA's) in our initial design, but converted to ROM's when the PLA's we were designing with were discontinued. By now, however, several useful PLA's are readily available. For example, thè Signetics FPLA, with its 16 inputs, is well suited to the decoding of PDP-11 instructions.

Below are the gains that might be expected in a second iteration of the CMU-11 design:

CMU-11	160 IC packages
Non-Unibus Design	128
Integrated ROM/MIR and horizontal microinstruction format	113
Convert to Am2900 circuits	95

COMPUTER-AIDED DESIGN TOOLS

Aside from freeing the designer of bookkeeping and clerical tasks, the main advantage of any design automation system is its inherent ability to maintain correct and consistent documentation (prints and wirelists) and the reduced turnaround time for design iterations. The fact that the total prototype development time for the CMU-11 was 39 (40 hours) man-weeks is an example of the savings possible with even modest design automation aids.

Description of facilities used at CMU

The Stanford University Drawing System was used to enter the schematic print set with a graphics display terminal. The drawing package includes a set of satellite programs to extract information for wirelists and cross-reference tables from its data base. Incorporated in the system are libraries of integrated circuit definitions which contain not only the pictorial representation of the gates but also pin section information and some loading data. Hard copy

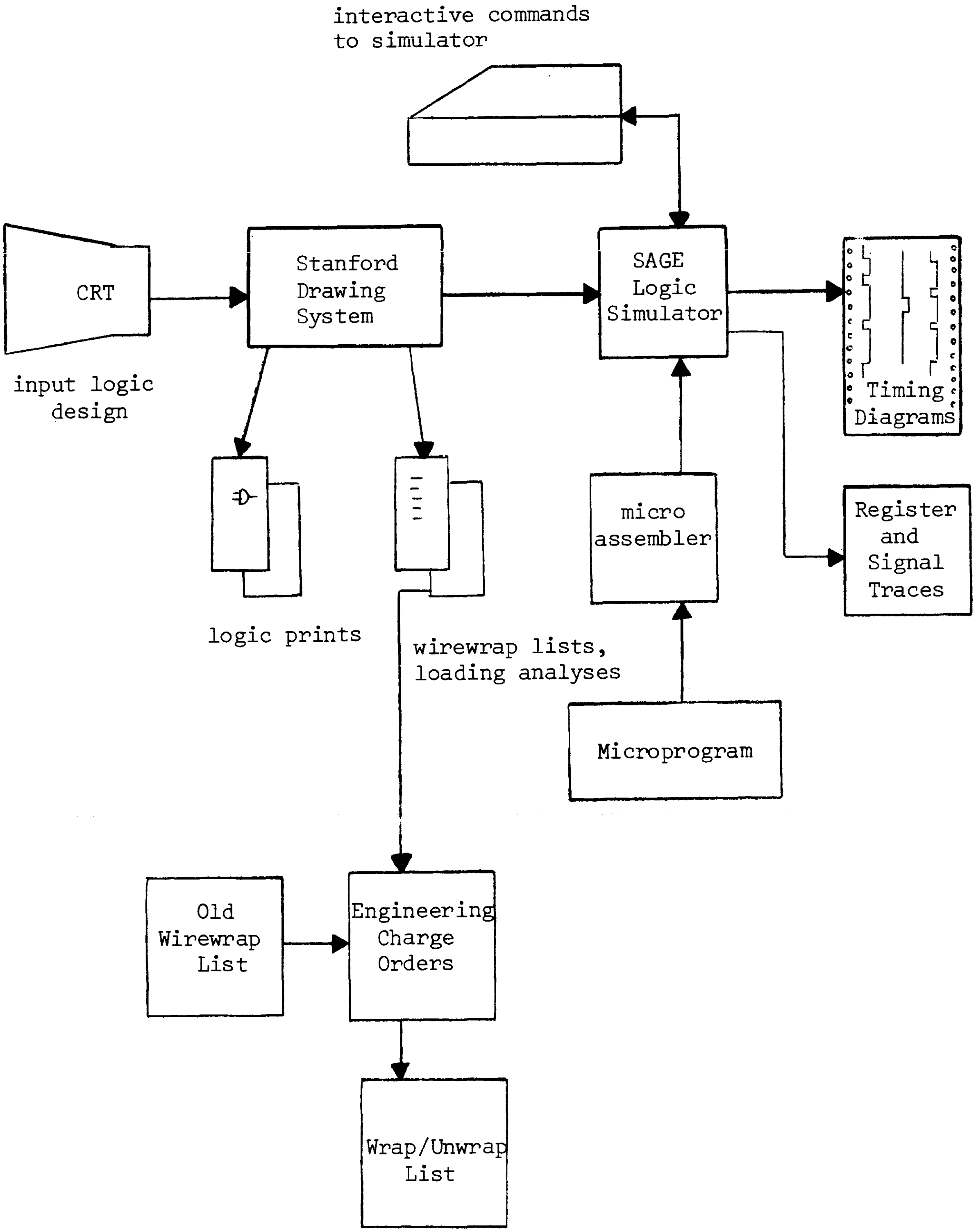


Figure 7—CAD system at CMU

prints were conveniently generated by an XGP (digitally controlled Xerox Graphic Printer). The wirelist program can search the data base interactively for specific information or produce complete tables of run lists, stuff lists, error reports (wire-anding violations), and loading analyses, which all proved extremely helpful.

The logic simulator used was SAGE (Simulation of Asynchronous Gate Elements), which is a four-state {0, 1, high impedance (tri-state buses), and undefined (initialization and uncertainty in delay parameters)} gate-level simulator. It reads the data base directly from the output of the Stanford Drawing system. This proved to be of utmost convenience, since it allowed a turnaround time in the order of five minutes for print set corrections. SAGE has models in its libraries for the TTL and Schottky families and special routines were written by us to emulate the 3000 microcomputer set. This allowed improvements in the efficiency of the simulation execution. Macro facilities are also available for quickly defining MSI circuits from more basic logic gates. The results of the simulations are in the form of register and signal reports and timing/trace diagrams.

Debugging with the simulator

About 95 percent of the original design errors were eliminated through the use of the simulation program. Naturally, not all combinations and sequences of instructions can be simulated, but a standard PDP-11 diagnostic program was run in addition to a number of other programs. A total of about 100 milliseconds' worth of CMU-11 compute time was simulated before debugging on the actual hardware began.

The limitation here was that the SAGE simulation of the CMU-11 required about 10^6 seconds of CPU time on a PDP-10 to simulate 1 second of CMU-11 execution. We simply could not afford to consume more than about 30 hours of CPU time for this project.

Whatever amount of time is spent on simulation, the simulations cannot be exhaustive and the final set of errors must be tracked down with more extensive tests on the real machine. We discovered eight to ten errors in the actual CMU-11. However, when an error was found in the physical machine, the simulations were again run to help track down the bug through the use of timing traces and other results. The correction was then entered into the machine print set and the simulator was re-run before implementing the change on the processor wire-wrap board or in the microprogram.

An example of the worth of the computer aided design system was when a major implementation change was made when several ROM's were incorporated into the design to replace a discontinued PLA (programmed logic array). Our design aids were essential in effecting this change within four man-days. In order to recover so quickly from such a massive wiring change, an ECO wrap/unwrap program was run using the old and new wirelists produced by the drawing package. Thus, at all times during development, the processor reflected the exact connectivity of the print set.

Several of the errors discovered on the real machine were timing errors that were not caught in the simulation debugging. These errors were not detected because the simulation models did not consider the effects of loading on the propogation delays and only maximum delays in all gates were used as an approximation to worst case conditions. In fact, if time had permitted, minimum and typical (Gaussian distributed) parameters should also have been tested. However, we again face a fundamental problem with simulation in that the computation time becomes excessive as different sets of delays are simulated to find worst case conditions.

CONCLUDING COMMENTS

The CMU-11 project was initiated as an experiment in constructing general purpose (mini) processors with LSI bit-slice components. Table VIII is a summary of the results. As the table shows, the CMU-11 was implemented with significantly less components (IC packages) than either the PDP-11/10 or the PDP-11/40, which are processors built with MSI components, and the performance of the CMU-11 falls between these two MSI processors. However, the economy of implementation is not nearly as significant as was realized with the LSI-11 although the CMU-11 is able to perform at twice the speed of the LSI-11. The LSI-11 is a processor implemented with NMOS LSI microcomputer packages in which the entire data path (with 8 bit data paths) was put in a single package and both the control and data packages for the LSI-11 have been specialized to efficiently emulate the PDP-11 architecture.

Earlier we discussed improvements that are possible in the CMU-11 design and argued that a second iteration on the design could boost the performance to that of the PDP-11/40 and could be implemented in about 95 rather than 144 packages. To achieve a more cost effective design than this will require either the development of some LSI control circuits specific to the processor's instruction set or will

TABLE VIII—Summary of Comparison between CMU-11 and Other PDP-11 Implementations

Parameter	LSI-11	PDP-11/10	CMU-11	PDP-11/40
Microcycle time (nsec)	400		200	140,200,300
Relative Execution Times	3.2	2.32	1.6	1.0
IC Packages	42	203	144	417
Control Store Size (bits)	11264	9960	9184	14056

require the specification of a new computer architecture tailored to make the most efficient use of the functions provided in the LSI circuits.

REFERENCES

1. Intel Schottky Bipolar LSI Microcomputer Set: 3001 Microprogram Control Unit, 3002 Control Progressive, Element, and 3003 Carry Lookahead Generator, Intel Corporation, Santa Clara, California, 1975.
2. *Introducing the Series 3000 Bipolar Microprocessor,* Signetics Corporation, Sunnyvale, California, 1975.
3. *PDP-11 Peripheral Handbook,* Digital Equipment Corporation, Maynard, Mass., 1973.
4. *LSI-11, PDP-11/03 Processor Handbook,* Digital Equipment Corporation, Maynard, Mass., 1975.
5. O'Loughlin, J. F., "Microprogramming a Fixed Architecture Machine," *Infotech State of the Art Report* 23, Infotech Information Limited, Maidenhead, England, 1975, 205-224.
6. *Am2900 Bipolar Microprocessor Circuits,* Advanced Micro Devices, Inc., Sunnyvale, California, 1975.
7. Fuller, S. H., T. McWilliams, and W. Sherwood, *CMU-11 Engineering Documentation,* Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pa., 1976.
8. *PDP-11/05/10/35/40 Processor Handbook,* Digital Equipment Corporation, Maynard, Mass., 1973.

Organizing and training for a new software development project—That big first step

by DANIEL FREEDMAN, DONALD GAUSE and GERALD WEINBERG
Ethnotech, Incorporated
Lincoln, Nebraska

ABSTRACT

This is a position paper describing several rules for the start-up of a new software development project. Management methods and attitudes are suggested which reduce the problems of misestimating people skills and task difficulty, appointing the wrong person as team leader, measuring conformity to specifications and adjusting to unforeseen changes. Establishing an environment which encourages continual, on-job education through the informal and formal review mechanism is advocated in favor of formal training programs. Team building and the concept of organizing for self-organizing teams is discussed.

> "*Remember, it's always darkest in the middle of the night.*"
>
> —The Fonz

INTRODUCTION

Our topic, as originally assigned, was "What is the *best* way to organize, recruit, and train for a new software development project?" The *best* way to organize this paper is by discarding that topic for one or two others:

a. What are better ways of getting started than we now often do?
b. What are some of the errors that ensure a *bad* start?
c. Given limited resources (and presentation time), what are the most important areas for concentration of management attention?

Our organization, Ethnotech, is a small one, with limited resources for consulting and training in software development. Like many of our clients, we have neither the time nor money to accumulate sufficient experience in software development to state the "best" way of doing any non-trivial aspect—and getting started properly is most assuredly non-trivial. We have, therefore, concentrated our own work in several areas—areas in which present practice is the poorest and which therefore promise greatest returns for a small investment. These areas are largely in the human side of the business, as our name suggests. Our paper will concentrate on these areas.

RULE 1—START WITH WHO YOU HAVE

Any successful project begins with a realistic appraisal of problems and resources. Put another way, more projects fail for lack of realism at the outset than for any other reason. There are two principal forms of this misestimation:

a. misestimating your people
b. misestimating the task

Let us first consider the people.

Because programming work is traditionally hidden from the view of all but a single person, there is no reliable way to estimate the capabilities of each technical person. Some programmers are considered aces because they talk a good program or spend a lot of time finding fantastic bugs that never should have been created in the first place. Others quietly and competently go about their work, unrecognized by their management. When the time comes to organize a project, the manager is working largely at random in matching people to responsibilities. We have seen, for example, numerous failures of chief programmer teams because of misplacement of individuals, such as

a. placing an inadequate person as chief
b. appointing the more adequate person as backup, even though the chief may be reasonably qualified

In order for a chief programmer team to work, the best person—who must also be an adequate person—must be the chief. If someone inadequate is appointed, the team quickly finds out— if they don't already know. In true team work, the open inspection of one another's work reduces the misestimation of people. Therefore, once teams have been well and truly established, the problem of assignment of responsibility diminishes.

This worthwhile feature of team organization is, unfortunately, lacking when an organization first decides to use the team approach for software development. We must then recognize that we *don't know* who we have—that our present assessment of relative strengths *may* be nothing more than a muddle of impressions, prejudices, and mistakes. Although this is a hard pill for many managers to swallow, swallow they must if they are to achieve successful software development with any regularity.

Of course, sometimes a manager is lucky in assigning people to responsibilities. Once off to a lucky start, new projects can succeed by simply following the adage: "Reward the successful with more challenging assignments." In an organization with a history of unlucky software development, some other strategy is needed. Unfortunately, we know of no other way to *reliably* select the *true* technical leader, so we recommend a strategy that doesn't require reliable selection by the management.

The approach that seems to work here is the *adaptive team*. The manager selects the members of the initial team by whatever means seem plausible, but avoids investing too much management prestige in the particular choice. No specific "chief" or "backup" is appointed. Team makeup is made conspicuously subject to revision as the team develops. The team members, with occasional gentle assists from the manager, gradually adapt the team membership and responsibilities to the task at hand, in the light of increasingly accurate appraisals.

RULE 2—NEVER ALLOW THE PROBLEM TO BECOME UNDEFINED

To someone outside the computing business, it must seem astonishing that people would be asked to make reliable estimates for building a product when

a. They don't know what it is they're building.
b. They don't have any reliable way of measuring how much has been built.

If the task to be done is misestimated at the outset, how can we expect the project to be well managed?

Much has been spoken and written about the importance of obtaining accurate specifications early in a project, so we need not labor that point. Suppose, for the moment, that we did somehow obtain accurate and complete specifications at the outset. Once the project gets into full swing, everyone knows

a. The specifications will change.
b. People will be too busy, personally too interested, or technically too ignorant to appraise the conformity of the partial project to the specifications.

The only known way to ensure a coherent project in the face of these problems is through regularly scheduled *formal reviews* of conceptually manageable portions of the project. When a specification is passed from one group to another, a formal review marks its passage. When a change is proposed to the specification, it must *invariably* be filtered through a formal review. When a team claims to have completed a certain portion of assigned work, it must be reviewed by outsiders before it can be considered part of the completed system.

Review by qualified outsiders assures that no team can drift away from project goals and accomplishments. *Periodic* reviews of *manageable* pieces ensures that the project as a whole cannot drift into undefined—and thus unmanageable—states.

But, of course, the project must be manageable to begin with, or it can hardly be made manageable by reviews or any other strategy. When we drop our assumption of accurate and complete initial specifications, we see immediately that the project *begins* with reviews. Even before teams are formed, in some cases, review groups can and must be put to the task of validating the completeness and accuracy of specifications—without which the project will certainly fail.

And, as a byproduct of these "pre-natal" reviews, all technical people begin to get sensible appraisals of one anothers' technical leadership skills. Thus, in attempting to control the task, you also begin to control the estimation of people.

RULE 3—INVEST EARLY AND KEEP INVESTING IN TEAM BUILDING

For successful software building, we must understand that a *team* is *not* the same thing as a *group*. A group is merely a collection of people in the same place at the same time, whereas a team is a collection of people sharing a past of working together and a future in which they will have to live with the consequences of present actions.

The contrast can be understood by thinking of sports teams. Winning teams are more than collections of talented individuals. Witness the singular lack of success of all-star "teams" when playing the true team that has won a championship. The all-stars, by definition, have far more talent, but seldom win. In programming projects, too much attention has been devoted to selecting a cast of all-stars, and not enough to getting the kind of collective effort that causes a team to build itself. No wonder we have seen so few programming teams and so many programming groups, sometimes mistakenly called "teams."

How does team-building take place? Though details vary, the underlying principle is always the same. Teams build by sharing experiences. You cannot build a Super-Bowl champion by having each player work out on a private practice field, and you cannot build a champion software building team when each member "owns" a private piece of the work. Only through mutual help and criticism can a team develop common understanding of objectives, accurate appraisals of individual abilities, patterns of work that involve the strengths and overcome the weaknesses of each team member, rapid and precise communication among team members, and all the other factors that make a healthy team so far superior a productive unit to any other group of people.

One way of establishing a team for a new project is to beg, borrow, or steal an already existing team. Building a new team takes time, and often means a fallow period in which the members will *seem* less productive than they would have been as individuals. Be sure, though, that the team was truly successful at its previous work, which is the best way to measure the progress of team building.

Another way of establishing teams is by keeping them intact, even though they have completed an initial project. IBM's New York Times project was rather large compared with many of the projects undertaken by the average DP organization. Indeed, the kind of programming done with the IBM Corporation is hardly typical of the programming done within the organization of IBM's average customer. If a new team had to be built for each new small project, few organizations could afford the team building effort.

IBM's experience of larger projects quite naturally led to team organizations because team building overhead was easily buried in large development costs. Organizations with many small projects will have to preserve their investment in team building by providing each team with a number of projects, either in serial, in parallel, or both. Though projects come and go, the team abides.

Conversely, throughout the lifetime of a team, members will come and go, rotating through the organization, leaving, retiring, or dying. By continually investing in teams, we invest the project with independence from the vicissitudes of the life of individuals. Rarely will the project find itself lacking someone experienced in some important aspect. To be sure, the comings and goings of team members add to the cost of maintaining teams, but only through teams can we, in effect, achieve redundant programming.

RULE 4—LET YOUR TECHNICAL LEADERS DO TECHNICAL LEADING

Regardless of the particular structure chosen for the team ~~by itself or by management, there are fairly universal roles~~ and responsibilities that must be assumed by someone. In particular, of course, is the role of "leader," but there are many aspects to the leader role. In the Chief Programmer Team as described by Mills, most of these aspects are bundled together in a single person backed up by another, similar, person. In practice, we more frequently find no single person capable of exercising all these aspects at the same time, for a variety of reasons.

One common reason is the heaping of too much supervision work on the few technically competent people. As a project grows, there is a tendency to try to accomplish a growing task by increasing the number of people on a team. Brooks has demonstrated why adding workers late in a project actually destroys working capacity, but there are other reasons to avoid this seductive management move. For one thing, the team, to the extent it is left alone to do so, will develop more capacity as the project progresses—without adding members. For another, each additional person to supervise—especially new persons—puts an additional burden on the leadership capacity of the team.

No team should have more than five members, except during transitions when an old member hasn't yet left but a new one has joined. This limit is well supported both by social science literature and software development experience. When the team exceeds this number, communication overhead rises. Either communication breaks down or nothing *but* communication takes place.

What about smaller teams? Starting with two, three, or four is a way to allow for growth within the project, but it's better to allow for growth in teamwork capacity, rather than in sheer membership bulk. Get the team members together as early as possible and start them working on something meaningful, so that team development can take place. Smaller teams place an additional burden on their leadership, for there may not be sufficient diversity to solve the diverse technical problems that the team will face.

Given the proper size team—three to six at the outside—the technical leader has a chance to exercise a critical role, including speaking for the group, interfacing with management, making technical decisions, teaching, providing access to resources, keeping the team calm, getting the team excited, being wise, making everybody laugh, and just doing whatever needs doing that nobody else seems able to do at the moment.

These roles may or may not be *vested* in a single person, but every team member will at times exercise a few of them. Some person on the team may be designated "chief" or similar title, and may even have management responsibilities. Quite often, however, it is best not to burden one person with too many expected roles, as this may leave insufficient time and energy for adapting to the unexpected.

Managers who tie up the technical leader's time in order to get "informed" are preventing the leader from leading. The formal review process is a more reliable source of much information a manager tries to get from the technical leader through time-consuming personal discussion.

Managers who have formerly held technical skills—espe~~cially those who still hold them—are easily tempted into~~ meddling in the technical leader's domain. When a football team has an inexperienced quarterback, the coach may be tempted to send in all the plays. But if that's the best technical leader you can muster, perhaps your project is doomed to failure in advance. In a recent championship game, one of the quarterbacks remarked, "I don't think I'd respond very well to getting plays from the coaches upstairs. There's a certain chemistry that takes place when the quarterback selects the play and tells 10 teammates, 'Here's what we'll do and now let's make it work.' I think that's extremely important to a team."

In the early stages of a project, a manager may be relatively well informed technically—at least when compared with the analysts, designers, and programmers. Also, the manager may have considerable spare time to devote to making technical decisions for the team and the would-be technical leaders. As the project progresses, however, the manager's relative technical skill will diminish, and spare time will be a vanishing commodity.

When it's no longer possible to make technical interventions, the meddling manager—like the coach who sends in plays from the bench—will find that the quarterback lacks the experience, confidence, and desire to select the right plays. Good performance in the game comes from good management prior to game time—not from sending in the winning play when the crowd is roaring loudest.

RULE 5—ALLOW ADEQUATE TIME AND SPACE FOR LEARNING

There will *always* be some unknown, uncertain areas in a software project. Every project should start with an assessment of its store of ignorance. This ignorance inventory warns how much uncertainty to allow for, and how much uncertainty in that uncertainty.

Before the project is finished, someone will have to *learn* all those things about which we are now ignorant. We must allow, from the outset, sufficient time and resources for that learning to take place. If we cannot, then we are gambling, not managing. It may be necessary to gamble if the stakes are high, but when you gamble you have to be prepared to lose, and lose often.

If we are building a system similar to one with which we have considerable experience, the margin for learning may be small. We can make reasonably tight guesses as to required people, machine time, and other resources. If the project is not similar to something we've done in the past, we cannot accurately estimate any of these requirements. In some cases, we can postpone the project itself in favor of a "research" project designed to pin down the looser ends—to educate us about the task before us. If we cannot politically afford a distinct research effort, we must lard the project with slack in which the learning can take place.

The need for education on projects with a large research component explains both the success and failure of certain efforts that have been called "top down." On well-understood projects, the top down work can take place solely in the design phase without much harm, because the designers can anticipate what the implementors will face. As the research component grows, however, the number of problems first recognized in "implementation" grows proportionately. By *implementing* top down, and by *postponing lower level design decisions* until they are actually needed in implementation, we gain the room necessary to learn what we don't know and to change the project accordingly.

For instance, we can modify the number of teams needed as the lower levels become successively better defined. Adding independent teams need not cause any major integration effort, particularly if the teams have already invested wisely in their own development. Implementing bottom-up, however, we would need the maximum number of teams at the beginning, long before we were in a position to make reasonable evaluations of the actual work load.

The learning for which we have made room takes place primarily on the job. Perhaps it ill-behooves a small company specializing in training to say so, but very little useful learning—proportionately—comes in classroom situations. When teams are in use, the proportion and quality of on-the-job learning is even greater.

It has been our experience, and the experience of our clients, that the best investment in education is that devoted to getting teams started—analyst teams, programmer teams, programmer/analyst teams, development teams, maintenance teams, training teams, documentation teams, design teams, . . . it really doesn't matter. The average project contains more than enough information and wisdom to be successful—if only it can be brought to bear in the right place at the right time.

Installations, groups, or individuals tend to develop a unique style of working—whether in the use of a particular language, style of development, tools, or work habits. Wherever there are communication boundaries, useful information is prevented from moving from one work style to another. We have found installations where people were totally ignorant of a particular language feature or programming method. Even worse are installations where one or several people know the feature or method but have never managed to communicate it to those who really need it.

Time after time, when such a shortcoming was pointed out in an informal review between team members it spread to the entire team in a few hours. When seen in a formal review, the new feature or technique became the property of the entire installation in a few days—without explicit expenditure for "education." Allowing adequate time and space for learning, then, is largely a matter of allowing adequate time and space for teams and formal reviews. These methods prove much more effective forms of education than sending people to schools—with an increased psychological advantage. Instead of learning a concept in a vacuum, we learn in the context of solving a particular problem—a problem guaranteed relevant to our project.

CONCLUSION—ORGANIZE FOR SELF-ORGANIZATION

The earliest decisions in a software development project have the greatest potential impact. The approach we have abstracted here is one of avoiding decisions whose impact will be limiting, in favor of decisions which will increase the problem-solving capability of the project organization. Instead of organizing for the project, organize the people who will organize the project, unencumbered by premature restrictions, uninformed intervention, counterproductive social organizations, and unmeasurable goals.

One major self-organizing structure is the *team.* Above the level of the team, and providing a second level of self-organization, is the *review process.* By organizing the various members of the project into mutually reviewing teams, we create a project which will

a. react swiftly and effectively to changes in specifications and constraints

b. learn to become increasingly productive, both individually and collectively
c. produce finished code which is readable and modifiable, progressing in steps which are measurable and controllable
d. enjoy a professional peer group environment, with less turnover and more work satisfaction

Critical to the team is the *technical leader,* or, rather, *technical leadership.* In some teams, most of the leadership is bundled into one or two members, as in the Chief Programmer Team concept. The true quality of a programming project environment, however, is measured most accurately by the amount of *participation* in technical leadership. In a business where one tiny mistake can cost millions, or one tiny idea can be worth millions, there is much to be lost by excluding people *a priori* from contributing their creative technical talents.

To be sure, a project usually begins—either well or badly—with the selection of a small number of conspicuous technical leaders—individuals who will have a recognizable influence on the shape of the project. If the project is small enough to need no others, well and good. Otherwise, the project's success will stand or fall on its ability to incorporate other contributors into its problem-solving processes. Certain would-be leaders have personal characteristics which stand in the way of others making contributions. Certain would-be managers have similar characteristics which interfere with the technical leaders. You might think that the one would learn from the experience of trying to work under the other, but nothing is certain when it comes to learning.

With proper social organization—teams and review procedures—the personal characteristics of a few leaders become less critical to the ultimate project success. Moreover, a healthy environment will lead to the growth of more and better technical leaders as time goes on, so later projects will become less and less critical. If only we can make the right start—or restart—now, starting new projects will grow easier and easier as time goes by. Indeed, *starting* projects will not be a problem at all, for one project will flow smoothly out of another in the process of creating more sensibly integrated, humanized systems. In the future, we'll know our profession has matured when we're no longer infected with that adolescent preoccupation with starting *new* things, or with finding the *best* way. Instead, we'll be looking for a better way of doing what we're doing now, which is needed badly enough.

The choice of new software development methodologies for software development projects

by EDWARD YOURDON
YOURDON Incorporated
New York, New York

ABSTRACT

Data processing managers have a number of new "structured" methodologies to assist them in EDP software projects: structured programming, structured design, HIPO, top-down development, structured analysis, structured walkthroughs, and chief programmer teams. Since many of these methodologies are still considered new and "experimental," it is often difficult for the manager to determine which of the methodologies should be used on a software project.

This paper briefly reviews each of the new structured methodologies. It then makes suggestions about the use of the methodologies for new projects, concluding that the use of informal walkthroughs is probably the best way for the manager to introduce the methodologies into an organization that has no previous experience with them.

The point is also made that "research-and-development" projects have different trade-offs than "bread-and-butter" projects. For projects that have hard deadlines and budgets, a number of trade-offs are suggested in order to help the manager decide which of the structured methodologies should be employed.

INTRODUCTION

The data processing project manager of the 1970's has an impressive array of new "structured" methodologies which promise to improve the productivity of his programmers and analysts, as well as improving the reliability, maintainability and overall quality of the finished product.

Unfortunately, there are so *many* "new" methodologies that the manager may not know which methodology—or combination of methodologies—he should employ on a new project. That choice is made all the more difficult because there is little or no documented evidence to *prove* the effectiveness of the new methodologies. Indeed, the problem is even worse: a variety of exaggerated claims in the popular EDP trade journals has made many a manager so skeptical that he may be unwilling to experiment with *any* of the new methodologies.

The purpose of this paper is to provide some useful advice to the project manager who finds himself in this position. A brief description of the more widely-known "structured" methodologies is given; following that, some suggestions are given as to the sequence and manner in which the new techniques can best be employed.

AN OVERVIEW OF THE STRUCTURED SOFTWARE DEVELOPMENT METHODOLOGIES

The collection of new software development methodologies is sometimes referred to as PPT (Programmer Productivity Techniques), or IPT (Improved Productivity Techniques), or SPT (Structured Programming Techniques). Almost every data processing organization has a slightly different understanding of the specific techniques which comprise the overall collection; however, the most common ones appear to be the seven described below.

Structured programming

Structured programming is widely regarded as the "first" new development methodology. Based on some theoretical work by Dijkstra, Böhm and Jacopini in the mid 1960's,[1-3] it has recently been discussed in a number of textbooks[4-9] and literally hundreds of papers in the computer literature.

A number of people have begun using the phrase "structured coding" to emphasize this methodology's most important features: procedural logic based on combinations of IF-THEN-ELSE, DO-WHILE and "sequence" structures. Such logic usually eliminates the need for GOTO statements, or unconditional branching instructions—a fact which has resulted in structured programming being given the nickname "GOTO-less programming."

Proponents of structured programming claim that structured code is easier to comprehend; it therefore tends to be more maintainable, and is more likely to be *correct* code. It is generally agreed that structured coding adds an overhead of 5-10 percent to the memory requirements and execution time of the program; however, there have been several cases where structured code has been *more* efficient than unstructured code, simply because it is better organized.

Structured design

Structured design is usually considered a "newer" discipline than structured programming, even though many of its concepts have been discussed by EDP professionals for several years. A 1974 paper in the *IBM Systems Journal*[10] marks the beginning of "real" interest in the subject; that paper was quickly followed by a number of books[11–14] and the usual plethora of papers in the popular journals and conference proceedings.

Structured design is usually described as the process of deciding which modules, interconnected in which way, will best solve some well-stated problem. Its emphasis is on techniques for identifying "good" modules (good, that is, from the viewpoint of maintenance and modification, rather than execution speed or memory requirements), and on systematic "cookbook" methods for deriving "good" designs for common types of EDP problems.

Proponents of structured design claim that it has the same virtues as structured programming: greater reliability, improved maintainability, and greater comprehension of how the system works. Some limited experiments suggest that structured design has a far greater impact on maintainability than structured programming, since it concentrates on building systems from small, highly-independent, single-purpose modules.

HIPO and other documentation techniques

Along with the recent interest in structured programming and structured design, there has been a great deal of interest in some new documentation techniques which can help describe the *procedural* logic represented by structured programming, and the *architectural* design represented by structured design.

The most widely known documentation technique is known as HIPO—an abbreviation for "Hierarchy, plus Input, Process and Output." Originally developed by IBM, it has recently been described by Katzan[15] in a sufficiently thorough fashion for EDP managers to consider using it as the documentation standard on new projects. An alternative diagramming technique, known as "structure charts," is described by Yourdon and Constantine.[12]

To document procedural design—that which has classically been documented with flowcharts, decision tables and narrative English text—such techniques as pseudocode (also known as "program design language[16]) and Nassi-Schneiderman diagrams[17] have gained popularity in some organizations. Other organizations have abandoned detailed documentation altogether, feeling that *any* method of detailed documentation will suffer from the problem of obsolescence.

Top-down development

Many data processing organizations were introduced to the concept of "top-down design" at the same time they were introduced to structured programming. This kind of design approach has also been referred to as "stepwise refinement," or "levels of abstraction," or "divide and conquer." Only recently has it become evident that one can easily design a bad system in a top-down fashion.

Meanwhile, there has been a great deal of discussion about the manner in which one should *implement* a well-designed system. In contrast to the classical approach (now referred to as "bottom-up") of unit testing low-level modules, and then integrating them into larger entities, it is now becoming popular to work in the other direction. That is, the "top-down" approach to implementing systems requires the coding and testing of the top-level (or "executive") module *first*, with the lower-level modules taking the form of "stubs" (a typical example of a stub is a module which exists immediately without doing any real processing.) Subsequent development of the system involves the substitution of real modules for the stubs.

Proponents of top-down implementation claim that it has a number of benefits, many of which are "political" in nature. The top-down approach tends to distribute system testing and integration throughout the entire project, rather than saving it for the end of the project. It also tends to expose major interface problems *early* in the project, rather than leaving them until the end of the project. Equally important, the top-down approach usually allows the project manager to demonstrate a *working* subset of the system to the customer at an early date, and the existence of a working subset also allows him to survive deadline crises more gracefully. It has also been observed that the top-down approach tends to distribute the requirements for testing resources more evenly throughout the project; by contrast, the "bottom-up" approach to testing usually requires large amounts of testing resources (e.g., computer test time) toward the end of the project—and it may be physically impossible to schedule, say, 25 hours per day of computer test time.

Structured analysis

With the advent of structured programming and structured design, it became clear that the major unsolved problem was that of the user: specifically, the problem has been that of figuring out what the user wants, so that a good system can be designed (using structured design), coded (using structured programming), implemented (using top-down development), and documented (with HIPO or structure charts).

Structured analysis addresses this problem. Its basic objective is to provide a formal description of the user's requirements, expressed in logical terms (i.e., with as little reference as possible to the peculiarities of a specific machine, a specific data base management system, etc.), using standard tools and building blocks. Its key ingredients are communication tools to improve the communication between analyst and user, and a new approach to the "systems development life cycle" that encourages both user and analyst to view the development of a software system as an *iterative* process, rather than a sequential one.

Since it is one of the newer disciplines, there is less literature on structured analysis than on structured programming or structured design. We expect that the few papers and books that are currently available[18–20] will increase substantially in the next few years.

Structured walkthroughs

The concept of walkthroughs, or "code reviews," seems to have its historical origin in Gerald Weinberg's classic book, *The Psychology of Computer Programming.*[21] Since then, it has been discussed in a number of places[22,23] and is regarded one of the more important of the new structured methodologies.

In their simplest form walkthroughs are a somewhat informal procedure for reviewing the correctness and quality of the analysis, design, code, test data and documentation associated with a software project. The review is normally carried out by the programmer's peers, rather than his supervisors; indeed, the review is normally done by *all* of the members of the project team.

Proponents of walkthroughs claim a number of benefits; increased reliability of the delivered product; more comprehensible and maintainable code; greater learning and sharing of information among team members; and a greater chance that a partially completed program can be salvaged if a programmer leaves in the middle of the project.

Chief programmer teams

Originally referred to as the "superprogrammer team" in the mid 1960's[24] the chief programmer team concept first attracted wide-spread attention on the New York Times project,[25] where it was used by IBM in conjunction with structured programming, top-down implementation and a variety of other techniques. Since then, it has been discussed in a variety of publications.[26,27]

The basic concept of a chief programmer team is to organize a software development project around a person who has (a) programming abilities substantially greater than—e.g., an order of magnitude greater than—other programmers in the organization, (b) ability to provide the documentation for the code, the operational procedures and the user manuals for the system, and (c) the ability to supervise a team of specialists include a "copilot" (an apprentice chief programmer), a "language lawyer" (an expert in the programming language or operating system or data base management system being used), a "toolsmith" (a person who develops useful debugging packages or other software development tools for the specific use of the project), a "librarian" (a person who organizes and controls the source programs, object programs, listings, and other documents associated with the project).

Proponents of the chief programmer team approach point out that it is merely taking advantage of some well-known facts about differences in programmer abilities.[28] In addition, they point out that the concentrated talents of one superprogrammer makes it possible for a medium-sized software development project to be accomplished with a much smaller group than would otherwise be necessary; consequently, the project manager can expect far fewer communication problems than he might otherwise expect.

SUGGESTIONS FOR INTRODUCING THE NEW STRUCTURED METHODOLOGIES

Unfortunately, it is not possible to give a simple algorithm in this area. We cannot easily say, "First you should introduce structured programming, then you should use structured design," nor can we say, "If you are working on a payroll system, then you should definitely use chief programmer teams; on the other hand if you are developing a real-time telecommunications system, you should use only structured walkthroughs."

On the other hand, the structured methodologies have been introduced into enough organizations that we can draw some general conclusions from their experiences. These are given below.

Trying to implement all of the new structured methodologies at once will generally be a disaster

Some organizations can actually pull off such a feat. After reading about the new methodologies, or getting a presentation from their friendly hardware vendor, they decide to use all of the new methodologies at once. As one might expect, this is more likely to happen in the smaller EDP organizations—those with only half a dozen programmer/analysts—and is *not* very likely to occur in the larger organizations.

Sometimes, though, an organization will decide to try all of the new structured methodologies on a single project; this is quite common when the organization decides to use the new methodologies as an experiment in a so-called pilot project. Even in a limited situation like this, it usually turns out that an attempt to experiment with half a dozen new methodologies at once leads to chaos and confusion.

The reasons are obvious enough. Structured programming and structured design are not simple concepts, and a lot of concentration is needed to make them work right. If the programmers are also trying to implement walk throughs—which require a great deal of psychological energy, too!—*and* chief programmer teams, as well as adjusting to the concept of a librarian relieving them of their clerical work . . . well, it will be a wonder if they get any of it right.

Techniques which involve organizational change are often the most difficult to implement

Some organizations will find it difficult to *ever* implement chief programmer teams, librarians and walkthroughs. The point here is that even if the project manager *can* convince his organization to try the chief programmer team concept, or librarians, or walkthroughs, he will probably find *that* difficult as his first new methodology. The author's experience has been that it is somewhat easier to introduce a relatively innocuous *technical* concept like structured programming *first*—that doesn't threaten anyone's empire, and is not likely to be at odds with current organizational philosophies.

Once the project manager has demonstrated that structured programming, top-down implementations and structured design are good ideas, *then* he'll probably be in a strong enough political position to say to the big boss, "Listen, the last three structured methodologies that I introduced to the company turned out to be winners. Why not gamble a little now, and let me try something like the chief programmer team concept?"

Structured code without structured design is often worthless

A number of organizations have found recently that structured programming (or, more specifically, structured *coding*) is a great idea but that it is not enough. If the modules in an EDP system are too large, too complex, and too interconnected with one another, then maintenance problems will persist regardless of the presence or absence of GOTO statements.

This raises some interesting *political* consequences. If the EDP organization has been doing things in a backwards fashion for years, *and* if the project manager introduces the new structured methodologies with great fanfare and promises of spectacular improvements, then the *first* new methodology should indeed demonstrate spectacular improvements.

And if the project manager tries structured programming *alone*, he might not achieve such spectacular improvements. The author's experience on a few EDP projects lately has been that the initial productivity and reliability will seem quite impressive, but the long-term maintainability of a system produced with nothing more than structured coding may not be very impressive at all.

The moral: It may make good sense to begin with structured design *first*—and when that is working properly, *then* introduce structured coding. Once the project manager has overcome all of the objections and battles and problems associated with structured design, it will be almost trivial to introduce structured programming.

There is a more important reason for this suggestion: *good* design and *mediocre* coding is a tolerable state of affairs; *mediocre* design and *good* coding, on the other hand, is not a good formula for success. And if the project manager thinks that his project team has energy, intelligence and enthusiasm to tackle only one new methodology, then structured design should get preference over structured coding.

Top-down design and implementation are often a good way of introducing the new structured methodologies

It is frequently observed that many of the benefits of top-down implementation are "political" in nature. It allows the project manager to demonstrate a working subset of his system to the user at an earlier point in time; it allows him to survive deadline crises more gracefully; and it allows him to schedule testing resources (e.g., computer test time) in a more manageable fashion.

These benefits are *very* noticeable to the user community, to higher levels of management, to the computer operations manager, and to various other people in the organization. For that reason alone, many EDP managers have decided that the top-down approach is a good way to introduce the new structured methodologies in their organizations.

Keep in mind that this approach *can* backfire. Unfortunately, many programmers view top-down implementation as an invitation to begin coding *before* they have done any real design. Especially on the first few projects, the manager should beware of this danger.

The most successful approach has often been informal walkthroughs

There is a strong argument for informal walkthroughs as the project manager's first venture into the new structured methodologies. Note the emphasis on "informal" walkthroughs—not necessarily with all the "bells and whistles" that are normally suggested (see, for example, the detailed procedures suggested in Yourdon's "Standards for Structured Walkthroughs" [23]).

Why would informal walkthroughs be a good way to get started with the new structured methodologies? For the simple reason that the project manager can't trust any individual programmer to understand and implement any of the other methodologies *by himself*. By forcing everyone to *talk* about their designs and their code—in an informal, low-key, non-threatening fashion—the manager can maintain some kind of quality control when he most needs it.

This is a point that needs emphasizing. If the manager has 30 programmers, and if he gives them all the standard textbooks on structured programming, they are almost guaranteed to read 30 different (and almost mutually exclusive) things. They will write 30 different kinds of structured programming—some good, some mediocre, and some downright *bad* (indeed, probably even worse than the kind of code that was written before structured programming came along). And if nobody looks at their code (which is the current state of affairs), the manager will never know who *really* understands structured programming, and who doesn't.

If the project manager begins by establishing an environment of exposing *everyone's* code to public discussion, then he will ensure that a relatively *uniform* version of top-down implementation, structured design, and structured programming can be implemented later on.

CONCLUSION

In the final analysis, only the project manager can decide which of the new structured methodologies he wants to introduce on a project. The suggestions in this paper can do nothing more than make the manager think about trade-offs that have been observed in other EDP projects; it is up to the manager to apply those trade-offs to his own project.

One of the most important questions the manager must ask himself is whether the new structured methodologies should be considered as a set of experimental "R&D" concepts, or whether they are to be considered down-to-earth *practical* concepts, with an immediate payoff.

Indeed, some organizations deliberately use the new structured methodologies on experimental "pilot" projects, with no preconceived ideas about which ones will work and which ones won't. In such an environment, the manager should use any and all of the methodologies that are of interest to him; our only caution is to arrange the pilot project in such a way that the impact of *each* new methodology can be measured in some crude fashion.

If the manager is involved in a "real" project—with real deadlines, real budgets, real users with real needs, and real penalties if the project fails—then he should be considerably more cautious about the new methodologies he employs. In this case, he will have to take into account his own perceptions about such things as:

a. The political climate within his organization—will the manager be given any encouragement if it is seen that he is "experimenting" with new technologies?
b. The nature of his project team—are they enthusiastic enough, experienced enough, and bright enough to try new methodologies while simultaneously working against a real deadline and budget?
c. The "learning curve" of the new methodologies—even with the best group of programmer/analysts, some time will be required to begin using the new methodologies properly. Is the project large enough and long-living enough to accommodate an initial investment in "learning" in return for a long-term payoff in productivity, reliability and maintainability?
d. The perceived "payoff" of the new methodologies—are they really as good as the popular EDP journals say they are? The manager has to make his own judgment of the impact on structured design, structured programming and walkthroughs on his project; this may be influenced by the nature of the applications, and various other factors.
e. The alternatives—if the manager elects *not* to use the new methodologies, what else can he use? On a simple EDP project, the manager may decide that the project is *guaranteed* to fail with the conventional methodologies; in such a situation, the manager may decide to "go for broke," and try all of the new methodologies.

REFERENCES

1. Böhm, C. and G. Jacopini, "Flow Diagrams, Turing Machines and Languages with Only Two Formulations Rules," *Communications of the ACM,* May 1966, pp. 366-371.
2. Dijkstra, E. W., "Programming Considered as a Human Activity," *Proceedings of IFIP Congress 65,* Spartan Books, Washington, D.C., 1965.
3. Dijkstra, E. W., "Go-To Statement Considered Harmful," Letter to the Editor, *Communications of the ACM,* March 1968.
4. Dahl, O. J., E. W. Dijkstra, and C. A. R. Hoare, *Structured Programming,* Prentice-Hall 1972.
5. Wirth, N. *Systematic Programming: An Introduction,* Prentice-Hall, 1973.
6. Yourdon, E. *Techniques of Program Structure and Design,* Prentice-Hall, 1975.
7. McGowan, C. L. and J. R. Kelly, *Top-Down Structured Programming Techniques,* Petrocelli/Charter, 1975.
8. Yourdon, E., C. P. Gane and T. Sarson, *Learning to Program in Structured COBOL,* YOURDON Incorporated, 1976.
9. McCracken, D. *A Simplified Guide to Structured COBOL Programming,* Wiley & Sons, 1976.
10. Stevens, W. G., G. J. Myers, and L. L. Constantine, "Structured Design," *IBM Systems Journal,* May 1974.
11. Myers, G. J. *Reliable Software Through Composite Design,* Petrocelli/Charter, 1975.
12. Yourdon, E. and L. L. Constantine, *Structured Design,* YOURDON Incorporated, 1975.
13. Jackson, M. A., *Principles of Program Design,* Academic Press, 1975.
14. Warnier, J. D., *The Logical Construction of Programs,* H. V. Stenfert-Kroese, Leiden, Holland, 1974.
15. Katzan, H. Jr., *Systems Design and Documentation: An Introduction to the HIPO Method,* Van Nostrand, 1976.
16. Caine, S. and E. K. Gordon, "PDL—A tool for software design," *Proceedings of the 1975 National Computer Conference.*
17. Nassi, I. and B. Schneiderman, "Flowchart techniques for Structured Programming," *ACM SIGPLAN Notices,* August 1973, pp. 12-26.
18. Gane, C. P. and V. Weinberg, *Structured Analysis,* YOURDON Incorporated, in press.
19. Yourdon, E., "The emergence of structured analysis," *Infosystems,* February 1976.
20. Gane, C. P., "Structured systems analysis and the training of systems analysts," *Proceedings of GUIDE 41,* November 1976.
21. Weinberg, G. W., *The Psychology of Computer Programming,* Van Nostrand, 1971.
22. Fagan, M. E. "Design and code inspections and process control in the development of programs," IBM, report IBM-SDD TR-21.572, December 1974.
23. ———, Technical Report Number 3: "Standards for Structured Walkthroughs," YOURDON Incorporated, 1976.
24. Aron, J. D., "The Superprogrammer Project," from *Software Engineering Concepts and Techniques,* Petrocelli/Charter, 1976.
25. Baker, F. T. "Chief Programmer Team Management of Production Programming," *IBM Systems Journal,* January 1972.
26. ———, "Chief Programmer Teams: Principles and Procedures," Report No. FSC 71-5108, IBM, Federal Systems Division, Gaithersburg, Maryland 20760.
27. Brooks, F. P., *The Mythical Man-Month,* Addison-Wesley, 1975.
28. Sackman, H., W. J. Erickson, and E. E. Grant, "Exploratory Experimental Studies Comparing Online and Offline Programming Performance," *Communications of the ACM,* January 1968.

Software development tools—Acquisition considerations—A position paper

by LEON G. STUCKI

Boeing Computer Services
Seattle, Washington

A NEED FOR TOOLS

The literature abounds now with many guidelines on current programming methodology. Today's software artisan is, however, often commissioned with more tasks to perform in a finite amount of time than can be accomplished using strictly manual techniques. Therefore, a knowledge of currently available automated programming tools is of key concern to today's software analysts and programming managers. The spectrum of tools currently available impacts the software life cycle at various points, for example:

(1) As an aid in ensuring that program specifications and standards are met,
(2) As a debugging aid,
(3) As a means of maintaining current and up to date documentation and configuration control,
(4) As an auditing aid for recording and checking acceptance test results, and
(5) As a performance measurement tool aimed at improving program efficiency.

GUIDELINES FOR EVALUATING TOOLS

The following guidelines are suggested for consideration when evaluating various generic classes of automated tools for a given project or for a given programming environment.

(1) Start early. Tool acquisition should be considered early in the software development cycle. Software tools cannot be commissioned in one day and installed on the next. Unforeseen problems often occur when installing tools at the user's site. Many tools must be customized for each new programming environment.
(2) Select easy to use tools. Care must be taken to select tools that are easy for the applications programmers to use. System interfaces and procedures must be easy to use, understand, and modify.
(3) Place tools in proper perspective. The use of tools must always be viewed in terms of supplementing, and not replacing, a good common sense approach to software development. The formulation of carefully designed test plans and good desk checking procedures are still of paramount concern.
(4) Remember that tools consume resources. In many cases, the application of automated tools initially increases machine costs. Some tools are very powerful and can be misused by applying them to classes of problems that could better be solved by simpler means.
(5) Select tools with a payoff. The utility of each potential tool should be understood by analyst and management alike. The use of an automated tool should assist in increasing the quality of the subject software, increasing user confidence, or increasing program performance in some way. Users should understand the benefits and cost associated with each tool in order to more fully optimize their utilization.

A SAMPLE EVALUATION OF SELECTED TOOLS

Table I was developed in the course of a lengthy investigation conducted in 1975–1976 by the author.[1] This table is by no means complete, it is simply offered as an example of some of the evaluation criteria involved in examining several of the more sophisticated automated programming tools.

Much work remains to be done in this area. We are still, for the most part, benchmarking tools to find their costs and guessing as to their potential benefits. Little empirical data has been gathered on the actual measured utility of applying tools. One recent set of experiments by the author[2] at UCLA pointed out some of the pitfalls often encountered in trying to assess the utility of tools. At the same time, however, it offered a prospect of some very interesting potential payoffs.

This presentation will offer a glimpse into the world of automated tools. Generic classes of tools will be considered. General remarks will be offered on various factors of tool utility. Specific mention will be made of selected findings from evaluation activities conducted by the author.

TABLE I—Summary of Test Results

TOOL	IMPLEMENTATIONS	SOURCE LANGUAGES ACCEPTED	STATIC / DYNAMIC	OPERATING COST 1-LOW 5-HIGH	EASE OF USE 1-EASY 5-DIFF.	FEATURES	LIMITATIONS
ATDG NASA-HOUSTON HOUSTON,TEXAS	UNIVAC	FORTRAN	STATIC	4	3	.DETERMINES VALUES OF PROGRAM VARIABLES TO PRODUCE OPTIMAL TEST CASES	.DIFFICULT TO INTERPRET OUTPUT
FORTUNE CAPEX PHOENIX,ARIZONA	IBM CDC	FORTRAN	STATIC	2	1	.EXACT STMT EXECUTION COUNTS .ESTIMATED EXEC TIME FOR EACH STATEMENT .NUMBER OF IF STATEMENT TRUE PATHS TAKEN .SUBROUTINE EXECUTION TIMINGS	.NO LONGER ACTIVELY SUPPORTED
NBS ANALYZER NBS WASHINGTON,D.C.	UNIVAC	FORTRAN	STATIC DYNAMIC	1	2	.SUMMARY OF SOURCE STATEMENTS BY FORTRAN TYPE .EXECUTION COUNTS FOR PROGRAM SEGMENTS	
PET MCDONNELL-DOUGLAS HUNTINGTON BEACH CALIFORNIA	IBM CDC HONEYWELL G.E. UNIVAC	FORTRAN	STATIC DYNAMIC	2 - 3 DEPENDING ON OPTIONS	1	.EXACT STMT EXECUTION COUNTS .IF STATEMENT TRUE AND FALSE BRANCH COUNTS .MINIMUM/MAXIMUM VALUES ATTAINED BY PROGRAM VARIABLES .FIRST AND LAST VALUES ATTAINED BY PROGRAM VARIABLES .RELATIVE SUBROUTINE EXEC TIMINGS	
PFORT BELL LAB MURRAY HILL NEW JERSEY	IBM CDC BURROUGHS HONEYWELL OTHERS	FORTRAN	STATIC	1	2	.CHECKS SOURCE PROGRAM FOR ADHERENCE TO A PORTABLE SUBSET OF ANS FORTRAN .SUBPROGRAM COMMUNICATION CHECKED THROUGH COMMON & ARGUMENT LISTS .SUBPROGRAM CROSS REFERENCE OF ID TYPE,USAGE AND ATTRIBUTES	
PPE BOOLE AND BABBAGE SUNNYVALE CALIFORNIA	IBM	ALL	DYNAMIC	1	2	.SAMPLES EXECUTING PROGRAM AT REGULAR INTERVALS TO OBTAIN PERFORMANCE STATISTICS .MEASURES I/O WAIT TIME .MEASURES TIME SPENT EXECUTING SVC'S	. PERFORMANCE STATISTICS ARE NOT REPORTED AT SOURCE STMT. LEVEL
STRUCTURING ENGINE (SFORTRAN) CANE,FARBER, GORDON,INC. PASADENA CALIFORNIA	IBM	FORTRAN	STATIC	5	1	.PRODUCES A STRUCTURED FORTRAN PROGRAM AND LISTING FROM FORTRAN SOURCE .PREPROCESSOR AVAILABLE TO TRANSLATE STRUCTURED PROGRAM INTO FORTRAN FOR COMPILATION	. 1000K LOAD MODULE . CAN PRODUCE EXCESSIVELY LARGE STRUCTURED PROGRAMS
DISSECT MCDONNELL-DOUGLAS HUNTINGTON BEACH CALIFORNIA	PDP10	FORTRAN	STATIC	4	3	.TESTS PROGRAMS INFORMALLY BY COMPUTING VALUES OF PROGRAM VARIABLES ALONG SELECTED PATHS .ALLOWS SYMBOLIC EXECUTION OF PROGRAMS	.SOME USER TIME REQUIRED TO SET UP TEST CASES
JOYCE MCDONNELL-DOUGLAS HUNTINGTON BEACH CALIFORNIA	CDC	FORTRAN	STATIC	2	2	.CHECKOUT AND DOCUMENTATION AID .MICROFILM FLOWLIST WITH ALL TRANSFERS INDICATED BY ARROWS AND ALL DO LOOPS INDICATED BY BRACKETS .COMPLETE CROSS-REFERENCED PROGRAM GLOSSARY	
DAVE COMP. SCIENCE DEPT. UNIV. OF COLORADO BOULDER, COLORADO	IBM CDC	FORTRAN	STATIC	5	2	.CHECKS FOR A WIDE VARIETY OF SOURCE ERRORS .ISSUES WARNING MESSAGES FOR POSSIBLE ERRORS	.REDUNDANT OUTPUT MESSAGES

FUTURE NEEDS

Much work remains to be done in the area. One current need is for more test environments (Software Engineering Laboratories) where various tools and techniques can be selectively applied and assessed under controlled conditions. A set of cost/benefit matrices is needed to aid in the quantitative evaluation of automated tools.

REFERENCES

1. Stucki, L. G. et al, "A Methodology for Producing Reliable Software," McDonnell Douglas Astronautics Co., Technical Report MDC G6210, March 1976.
2. Stucki, L. G., "The Use of Dynamic Assertions to Improve Software Quality," PhD Dissertation, UCLA Graduate School of Engineering, Computer Science Department, June 1976.

Understanding the developmental life cycle

by RAY CAUDILL
Air Force Data Automation Agency
Gunter AFB, Alabama

ABSTRACT

The paper supporting the topic above asserts that the software development manager is faced with a myriad of technical questions to which he must seek technical solutions. The paper makes a case for applying an adaptation of configuration management to the software developmental life cycle. This is done by first stating that the life cycle can, in fact, be divided into reasonably discrete parts even though the developmental life cycle is a continuum. The paper then gives an overview of Automated Data System Project Management, highlighting phases, reviews, deliverables, baselines and scaling. An alternative to this version of configuration management is also offered for the purpose of showing the flexibility of configuration management. The paper concludes by asserting that the elusive term "management visibility and control" becomes realistic and achievable through application of configuration management. Also, a better choice of tools and techniques can be made when gauging technical need against phases rather than the entire developmental life cycle as a whole. Lastly, the paper concludes that progress measurement and management reporting actually can be achieved through documentation of the results of reviews of deliverables expected from each phase.

INTRODUCTION

The title of this session is "Software Management: How to Start a Software Development Project." This is one of the less technical sessions and is aimed at the manager, new or otherwise, who is about to begin a development effort. A senior analyst/programmer, given project leadership for the first time, is confident of success. He's been there. He's done the "grunt" work. He knows how to "build" software. However, he has never "managed" a software development effort. There is a difference. His confidence in his eventual success transcends his superiors and staff alike. Because of all this show of confidence, everyone predicts high marks for this project to be finished on time, within budget estimates, and that it will totally satisfy the customer. Amid all this optimism, the manager is asking questions like: What skills do I need?; Which of the many tools and techniques on the market are right for me?; and, How do I know my plan is holding? In short, he's doing what most people think he should be doing. That is, he is seeking technical solutions to technical problems. He just may be making his first serious mistake. He may be beginning on an elevated technical plane much too quickly.

My suggestion is that he hold the technical questions a little longer and ask and seek a solution to a less technical one first. He should ask himself this all-too-rare question first: "What are the *Component Parts* of this *thing* called the Software Developmental Life Cycle?" If he can see through or past the "whole" and actually discern the smaller, somewhat discrete, parts of the developmental life cycle, and plan and manage accordingly, he will possess and be able to articulate a much greater understanding of the whole. I firmly believe, then, that he has a much greater chance of a level of success that just might match his initial optimism.

Before proceeding, let's briefly explore some of the reasons why software development efforts fail anyway. You each have your own opinions, some of which relate to technical complications and perhaps to the employment of specific tools and techniques. However, I'd like to make some of my own observations as to why failures occur.

Lack of management visibility and control

Management at all levels "assumes" that the manager closest to the development effort does have visibility and control of the total effort and that a simple query will bring a clear and concise response as to project planning and progress. Some project managers do have such visibility and control, but there is absolutely no reason for you or anyone else to assume they do. If your thought and planning processes don't or can't transcend the "whole" and see the parts of the software development life cycle (which is the only place where visibility and control exist) then why should you expect another manager, technical or otherwise, to do so?

Pressures from higher authority

Here's a serious problem for most of us. Let's say you've laid out a positively foolproof developmental plan. Higher management then decides to step up your schedule by six months, or to reduce your budget by some large amount. Well, you have no choice now but to modify that great plan you had. You guessed it, it's that alternative plan that is doomed to failure from the start. But why should that be so? If your plan is by phases, or parts of the whole, you can modify each accordingly and the basic plan still holds. You can't just cut out and throw away, or ignore a part of the developmental cycle just because of a reduction in time, people, or money. To say it another way, there are certain actions or activities which must occur in order to develop a complete computer software system. You cannot cut out any one of those activities; you can only develop less system. The alternative (the one doomed to failure) is to develop the same system but with less quality.

Now for a couple of "truisms" or "propositions" to cement my position for you and to give you a baseline, or point of departure for the remainder of this paper.

Proposition 1: The developmental life cycle is a continuum. Dictionary Definition: A continuum is a "succession, or whole, no part of which can be distinguished from neighboring parts. . . ."
Proposition 2: The developmental life cycle is a succession of discernible, manageable, and measurable parts.

These propositions appear contradictory. I happen to believe both and hope to convince you of the same. More of us than care to admit have been involved in failures because we saw only the "whole" and managed accordingly rather than seeing through the "whole" to its "parts" and dealing with them on an individual basis.

PHASE RECOGNITION

There are as many phases in the developmental life cycle as you'd like there to be. I'll show you two examples of phase use later, but for now just visualize your entire project as having the following six phases:

1. The idea and its approval.
2. Refine the idea into a user requirement.
3. Design the application system.
4. Develop, code, and test programs.
5. Total system testing.
6. Operate the system.

It is only after one recognizes these "parts," now referred to as "phases," that the technical questions referenced earlier take on some perspective. You're able now to apply questions to specific phases of the software development life cycle and deal with them accordingly. Also, those rather elusive terms such as "management visibility and control" begin to take on true meaning. Planning each phase singly, then putting them together can make for a more detailed and a more accurate plan. You've all witnessed this negative example of project visibility and control. A project is assigned to a group of people with the singular guidance: "Have it on the air in 18 months." You, and they, are quite naturally 1/18 more nearly complete at the end of every month. Suddenly, in the 16th or 17th month, you find that you/they are only 50 percent finished. You're forced to accept that assessment even though you're too embarrassed to ask the question: "50 percent of what?"

THE ADVANTAGES OF PHASES

There are many advantages to development by phases: Here are a few:

- Each phase has a relationship to all others and to the entire project (the "one small step at a time" concept). Comprehension of each makes the "whole" comprehensible.
- Each phase must yield some deliverable product in order to terminate.
- User/developer reviews of deliverables for technical acceptability become the measure of progress and the method of reporting that progress to management.
- Responsibilities/accountability for specific phases, and deliverables can be identified.
- User or design changes can be controlled and accounted for.
- Finally, dissecting the developmental life cycle into deliverable-oriented phases with appropriate reviews can:
  - — Provide early warning of technical problems/slippages.
  - — Provide setting for controlled user/developer relations.
  - — Provide for some realistic assessment of where the effort stands relative to plans.
  - — Make visibility and control realistic.
  - — Enable the development manager to respond to serious changes directed by higher management echelons.
  - — Lessen the trauma of implementation.

CONFIGURATION MANAGEMENT

Of course, what I'm leading up to is a set of concepts and principles commonly referred to as configuration management. For years, configuration management has been the tool of managers of ultra-large efforts such as shipbuilding and aircraft manufacturing. In the last few years, however, more and more people are applying the same concepts and principles, with modifications, to software development. My first thought, upon reading about configuration manage-

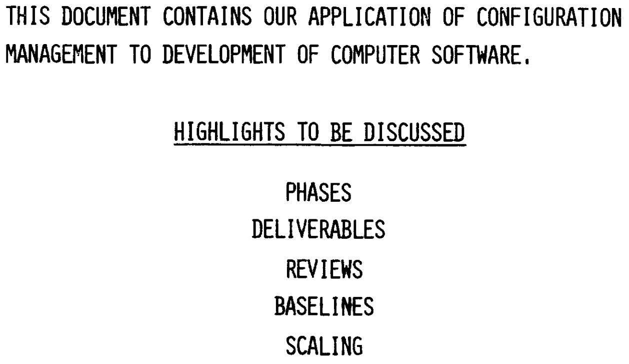
"A DISCIPLINE APPLYING TECHNICAL AND
ADMINISTRATIVE DIRECTION TO AND
SURVEILLANCE OF . . ." THE SOFTWARE
DEVELOPMENTAL LIFE CYCLE.

Figure 1—Configuration management

ment for the first time, was that it is overkill to the nth degree. It is only if you allow it to be. The concepts and principles have to be tailored and tempered to meet the requirements and scope of each project. Figure 1 is a simplistic definition of software configuration management and shows the aspects of it this paper will develop. Those who have dealt with configuration management in the development of weapons systems may believe that we have strayed rather far from what you know as configuration management. I do not believe we have. Those of us who developed the version I'll touch on were always mindful of phasing, deliverables, reviews, etc.

AUTOMATED DATA SYSTEMS PROJECT MANAGEMENT

We are using configuration management now in software development. I believe it is safe to say we have had developmental successes and a failure or two. With no particular case in mind, perhaps we would have had fewer failures plus more efficient successes had we become acutely aware of the potentialities of configuration management many years ago. There are various elements within the software development field which have been exploring and in fact implementing configuration management for the last couple of years. Unfortunately, it takes considerable time to get a process of this magnitude legislated and in use across the entire organization. We are now on the threshold of the broadest possible usage of the concepts and principles of configuration management. I'll spend the next few moments giving you an overview of our approach to it. In the interest of brevity, it will be precisely that, an overview. Figure 2 shows the portions of the document on this subject which will be discussed in this paper.



THIS DOCUMENT CONTAINS OUR APPLICATION OF CONFIGURATION MANAGEMENT TO DEVELOPMENT OF COMPUTER SOFTWARE.

HIGHLIGHTS TO BE DISCUSSED

PHASES
DELIVERABLES
REVIEWS
BASELINES
SCALING

Figure 2—Automated data systems project management

PHASES

We recognize these phases: Conceptual, definition, development, test and operation. Figure 3 shows graphically the phases and what occurs during each. Here follows a brief narrative description of each:

Conceptual phase

The purpose here is to identify and fully analyze in detail the user requirement. This also involves project planning and gaining approval for project development.

Definition phase

During this phase, the total system is designed and cross-checked to assure that each system function satisfies a user requirement stated in the previous phase.

Development phase

All programs are designed, coded and tested during this phase. Various reviews are also accomplished to assure technical acceptability of the programs and to assure that products produced are as intended.

Test phase

The total system is tested in an independent environment as well as in the user environment if necessary during this phase. Audits are performed here to ascertain user acceptability of each deliverable product.

Operation phase

The system is put into full operation. Feedback from the user is as required to gain from his experience

PHASE DELIVERABLES

No attempt will be made here to list or discuss all deliverable products for each phase. The purpose here is to give you some insight into the type of deliverable product that is prepared and reviewed in each phase.

Conceptual phase

Data Automation Requirement—This is a statement of a problem needing attention and will include as much relevant detail about the problem as is available at the time.

Functional Description—This is the most detailed statement of the user/customer requirements. This is the docu-

CONCEPTUAL	DEFINITION	DEVELOPMENT	TEST	OPERATION
IDENTIFY AND ANALYZE THE USER REQUIREMENT	DESIGN THE TOTAL SYSTEM	DESIGN PROGRAMS, CODE AND TEST	TEST TOTAL SYSTEM IN INDEPENDENT AND USER ENVIRONMENT	OPERATE

Figure 3—Phases

ment used to cement a common understanding between the user and the developer and serves as the basis for system design.

Definition phase

System/Subsystem Specification—This "document" is the system design and includes such items as the total system design, interfaces with other systems, subsystem specifications, etc.

Development phase

Program Specifications—This document includes the logic of each computer program and is the basis for coding. It specifies the functions to be performed by the program, its inputs/outputs, and interfaces.

Coded Programs—Self explanatory.

Test Plans—As the name implies, this document includes plans for testing the total system independently and in the user environment.

Operator and User Manuals—These documents include everything the operator needs to know to *operate* the systems, and everything the user needs to know to "use" the software in his environment.

Test phase

Test Reports—The results of all test activities are recorded and fed back to the developers and the users.

Operation phase

User Feedback—There is an administrative procedure established to cover user feedback. However, letters, phone calls, and visits are all used for this purpose.

REVIEWS

There are reviews that must be accomplished throughout the software developmental life cycle. Figure 4 shows some of the reviews we have elected to use. Reviews are iterative. That is, if the product reviewed does not fulfill all requirements, it's back to the drawing board with suggested or directed changes. Also, it is important to point out that the user/customer is involved to some degree, as a direct participant, or as an observer, in virtually every review. Hence, the user is always apprised of progress and he always knows exactly what he's going to get. Secondly, if he changes his mind about his requirements late in the cycle, he has full understanding and appreciation that it may cost him in terms of his desired on-line target date. However, by keeping him involved, he's more likely to give you well thought out requirements to begin with. He's less likely to say what so many have said in the past—"just give me something and I'll tell you whether I like it or not."

Conceptual phase

Systems Requirements Review—Through this review, the user and the developer reach a common understanding of the user requirements, of conceptually what is to be developed, and of the plan for doing so.

Definition phase

System Design Review—The purpose of this review is to gain both user and developer approval of the total system design and to assure that the design satisfies the requirements approved in the previous phase.

Development phase

Preliminary Design Review—The purpose of this review is to approve a portion of the total system for program development. For a smaller system, this review will be combined with the System Design Review.

Critical Design Review.—The purpose of this review is to review and approve the individual program specifications prior to coding. This review, as well as the others, prevents that historic problem of the "rush to the code sheet."

Product Verification Review—The purpose of this review is to determine that computer software was developed in accordance with specifications.

Test phase

System Validation Review—The purpose of this review is to ascertain that the system performs the function for which it was designed.

There are various audits which occur during the Develop-

CONCEPTUAL	DEFINITION	DEVELOPMENT	TEST	OPERATION
IDENTIFY AND ANALYZE THE USER REQUIREMENT	DESIGN THE TOTAL SYSTEM	DESIGN PROGRAMS CODE AND TEST	TEST TOTAL SYSTEM IN INDEPENDENT AND USER ENVIRONMENT	OPERATE
SYSTEM REQUIREMENTS REVIEW	SYSTEM DESIGN REVIEW	PRELIMINARY DESIGN REVIEW	SYSTEM VALIDATION REVIEW	
PURPOSE: ASSURE USER/ DEVELOPER COMMON UNDERSTANDING OF REQUIREMENT AND PLANS/ PROGRESS	PURPOSE: REVIEW/APPROVE TOTAL SYSTEM DESIGN, PLANS AND TRACEABILITY	PURPOSE: REVIEW/APPROVE PORTION OF TOTAL SYSTEM FOR FURTHER DEVELOPMENT AND TRACEABILITY	PURPOSE: ASSURE THAT EACH SEGMENT OF THE SYSTEM OPERATES/ PRODUCES AS SPECIFIED HAS TRACEABILITY	
		CRITICAL DESIGN REVIEW		
		PURPOSE: REVIEW/ APPROVE PROGRAM SPECIFICATIONS PRIOR TO CODING		

NOTE: THIS CHART REPRESENTS ONLY A SELECTED SAMPLE OF REVIEWS WHICH CAN BE USED.

Figure 4—Reviews

ment and Test phases. These are generally conducted with the user as a final physical check on all deliverable products. Audits usually serve as a portion of the "grunt" work that goes into the System Validation and Product Verification Reviews. Secondly, all reviews are accomplished with the rationale that each product reviewed must be traceable back to some previously reviewed and approved document.

BASELINES

A baseline serves as a point of departure for future development or action. Hence, once the system design (system/subsystem specification for example), has been reviewed and approved, it is baselined. All future program development must satisfy that baselined document. This isn't to say that changes are forbidden. They certainly are not, but they are controlled and are subject to the same reviews and approvals as were the original documents. Baselines are generally established at the conclusion of a phase, but there is no reason a specific document cannot be baselined anytime that it suits your plans. The point I wish to make is that baselines are flexible, just as any other principle of software configuration management is. You must determine for yourself what your baselines are to be. That means you must determine what documents are to be baselined, when they are to be baselined, and by which review. This aspect alone forces you into some pretty finite project planning.

PROJECT SCALING

I never intended to make you think that every project undergoes all these reviews, and must produce all the documents. Remember the statement about "overkill" that I made earlier in this paper. For smaller projects, reviews can be combined or eliminated and so can documents. However, this is done only after adequate planning. I'll provide only one example which gives you an idea of how we would scale the procedures of configuration management to a specific project. This example shows what reviews will be accomplished for a project based upon project manyears. The following Decision Logic Table is self-explanatory and happens to be for projects which would be developed for multiple site implementation.

Project Scaling (Example)

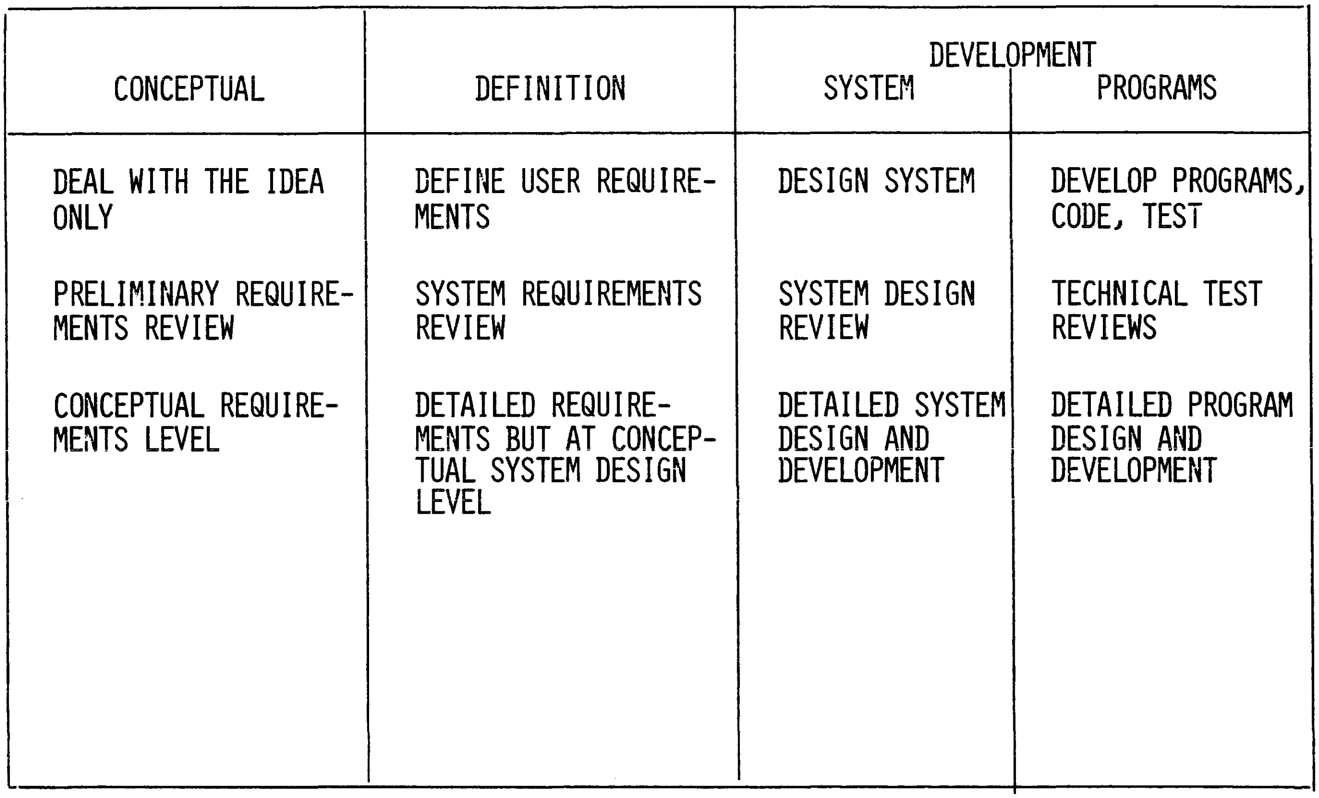
If Project Manyears are:	1-<10	10-<25	25-<50	50->
Then Perform:				
System Requirement Review	×	×	×	×
System Design Review	×	×	×	×
System Validation Review	×	×	×	×
Product Verification Review		×	×	×
Preliminary Design Review			×	×
Critical Design Review			×	×
Audits				×

ANOTHER EXAMPLE OF APPLIED SOFTWARE CONFIGURATION MANAGEMENT

My purpose is to convince you that your chances of success are probably greater if you accept the concepts and principles of configuration management and tailor them to suit your situation. With that in mind, I'd like to show you another variation of what I've just shown you in the last several charts. This variation uses the same phase names but with different accomplishments in each. It does not use the same reviews either, but a study of the document from which it came revealed that the same activities generally occurred. The following comparative chart will show you the differences between these two variations.

APPROACH #1	COMPARATIVE PROCEDURE
CONCEPTUAL	*CONCEPTUAL*
Deals with everything the example does plus the functional requirements are defined in detail.	Deals only with the idea at the conceptual level and is used for project approval.
DEFINITION	*DEFINITION*
System design occurs.	Detailed user requirements analysis occurs.
DEVELOPMENT	*DEVELOPMENT*
Develop programs.	Develop both the system and programs, but done in discrete segments of the phase.
TEST	*TEST*
Essentially the same.	Essentially the same.
OPERATION	*OPERATION*
Essentially the same.	Essentially the same.

Figure 5 shows a comparative example of phasing along with a brief statement as to what occurs in each phase. This comparative procedure was included to show an example of how software configuration management can be varied. The same phase names were used in both the versions above, but the activities which occur in the first three are quite different. For example, in our version, the Conceptual Phase covers everything from the initial idea, project approval through the detailed user requirements definition. The comparative version uses the Conceptual Phase to deal only with the basic idea and the approval for its development. The Definition Phases in the two versions also vary considerably. In our version, the system is designed, whereas in the other, definition means detailed definition of



CONCEPTUAL	DEFINITION	DEVELOPMENT	
		SYSTEM	PROGRAMS
DEAL WITH THE IDEA ONLY	DEFINE USER REQUIREMENTS	DESIGN SYSTEM	DEVELOP PROGRAMS, CODE, TEST
PRELIMINARY REQUIREMENTS REVIEW	SYSTEM REQUIREMENTS REVIEW	SYSTEM DESIGN REVIEW	TECHNICAL TEST REVIEWS
CONCEPTUAL REQUIREMENTS LEVEL	DETAILED REQUIREMENTS BUT AT CONCEPTUAL SYSTEM DESIGN LEVEL	DETAILED SYSTEM DESIGN AND DEVELOPMENT	DETAILED PROGRAM DESIGN AND DEVELOPMENT

Figure 5—Another example of phasing

the user requirements. There are other differences as well. The point I wish to reiterate is that Software Configuration Management is flexible and should be tailored to suit your needs.

SUMMARY

The introduction to this paper asserts that many technical problems in need of technical solutions beset the manager ~~as he contemplates a developmental effort. However, this~~ paper further asserts that the obvious technical questions can best be addressed if put into proper perspective. A way of doing that is to deal with the whole (developmental cycle) by separating it into its component parts or phases. We have adopted the concepts and principles of configuration management toward that end. The example of another variation of configuration management was briefly discussed to assure you that you have much latitude in applying the concepts and principles of configuration management.

CONCLUSION

The developmental process is a continuum. However, I have concluded that the application of the concepts and principles of Software Configuration Management should be given priority consideration when beginning a software development effort. Planning, controlling and developing then become phase/review/deliverable oriented and progress measurement and reporting become realistic and meaningful. Specific advantages of applying configuration management to the software development life cycle are:

- Planning, controlling and visibility become more realistic because reviews of specific deliverables must be accomplished throughout the cycle.
- Plans which have been prepared around the phased concept are easier to change.
- Technically oriented tools and techniques can be selected and applied to phases or groups of phases with more confidence.
- A workable user/developer relationship can be identified and controlled.
- The biggest pay-off of all:
  - — We data automation people can now "articulate" to others precisely what the developmental life cycle actually is.
  - Greater confidence from management and users.
  - System developers' credibility can be enhanced.
  - Can stop or at least impede the "rush to the code sheet."
  - Can articulate the need for a somewhat greater investment up front which will increase both speed and accuracy of the systems work and program development.

Management of large scale computer program production

by H. S. WOODGATE

International Computers Limited
Reading, Berks, England

ABSTRACT

This paper describes a management/computer interactive system for the planning and control of the development of computer software programs on a large scale.

Linked planning (PERT type) networks are used to represent the basic information model. This composite network is formed within the computer from individual networks using computerized library techniques. The model then represents the overall definition of the projected work load.

This information model is then manipulated by varying both the planning data (resources available, etc.) and scheduling parameters. Multi-project scheduling methods, using advanced resource allocation procedures, are then employed to rapidly obtain basic planning information (manpower loading, completion dates, costs, etc.).

These basic concepts are established within a framework of a flexible system which responds to changes in product requirements, resource availability levels and project priorities. The interaction between management and the computer based network planning system enables the work to be effectively planned and provides a sound basis for subsequent control.

INTRODUCTION

The management of large scale computer program production poses special problems which are not readily solved by manual methods. The large volume of planning data involved, the technical complexity of the end product, the uncertainty and variability of the time and resource estimates and the vulnerability to external constraints make it essential that computer driven methods of project management are employed.

The system described was developed for in-house use by International Computers Limited in an environment where several hundred programmers and systems analysts were engaged in the development of a multiplicity of computer programs. The integrated computer based systems evolved over a long period in response to increasing work volumes. Network planning (PERT) techniques were used for many years to schedule projects individually. However, the separate treatment highlighted the need to consider the various tasks collectively against a common manpower pool and development of the system proceeded to this goal. The principal stages in development are shown in Figure 1.

As the number of projects increased so did the variety of questions which management was called upon to answer. With a mixture of firm and tentative projects on hand, estimates of costs and timescales were requested for new items about which only the barest details were known. Answers to the first questions inevitably had to be hedged with qualifications regarding the assumptions made, and these invariably sparked off a series of further questions of the "what if" variety where changes in these assumptions were postulated.

Some fast, flexible, but reliable system was required to provide this information and it was from this necessity that the present methods developed. Included in the work groups were staff who had developed PERT programs, and there was an awareness of the potentiality of the technique. It was not surprising, therefore, that network planning should be selected as the basis for a scheme for scheduling and controlling the work of the whole activity.

THE PLANNING PROBLEM

A number of complex computer programs have to be produced using pre-determined manpower and computer time availabilities. The individual computer programs are required to be finished in a defined priority sequence, and there are interdependencies between different programs and also some external constraints. The various computer programs (some 50/60 in number) have similar production characteristics, but the work force is also required to undertake some additional work of a dissimilar nature

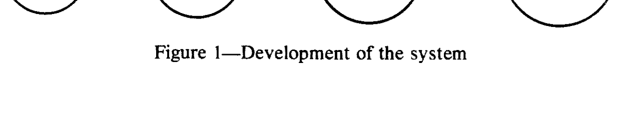

INDIVIDUAL PROJECTS PLANNED BY NETWORK
RESOURCE ANALYSIS AND COSTING
LIBRARY TECHNIQUES AND RESOURCE ALLOCATION
INTEGRATED SYSTEM FOR MULTI-PROJECT SCHEDULING

Figure 1—Development of the system

(maintenance of previously completed programs, technical support, preparation of manuals etc.). Some tasks can be considered as firm commitments, others are only tentative proposals for which production capacity must be reserved. The resulting work load is a mixture of these ingredients. This overview of the planning problem is shown diagrammatically in Figure 2.

The number of projects to be scheduled varies as time proceeds. As work in progress is completed, tentative proposals become firm commitments, new work items are added, state of dependencies change, availability of manpower alters (transfers and resignations, etc.).

Superimposed upon this constantly changing variety of work and resources is the infamous problem of estimating the work content of computer programming tasks. There can be few tasks for which it is more difficult to assess in advance the time scale by which a given number of persons achieve specific objectives. Uncertainty in the original estimates is therefore a significant factor in all planning decisions.

From this morass of indefinite and sometimes conflicting detail, project management needs to forecast certain highly significant information:

(a) Who should be working on what job and when?
(b) What will be the forecast completion date?
(c) What will it cost?

In connection with the allocation of jobs to people, certain basic limitations of human adaptability have to be borne in mind. Pre-eminent among these are:

(1) The need to make use of experience—an Operational Research Scientist may stumble through an accountancy routine but an accountant may not be so successful in O.R.
(2) Continuity of employment on the same task improves performance—changing individuals rapidly from job to job is invariably detrimental to progress. It is the nature of programming that it is possible to actually progress backwards by inappropriate actions.

Network planning is an obvious planning tool to employ in these circumstances. The breaking down of the overall task into individual activities minimizes the estimating error, and the computer calculated time analysis, resource allocation and costings enable the multitudinous elements of this particular jigsaw puzzle to be re-assembled with comparative ease.

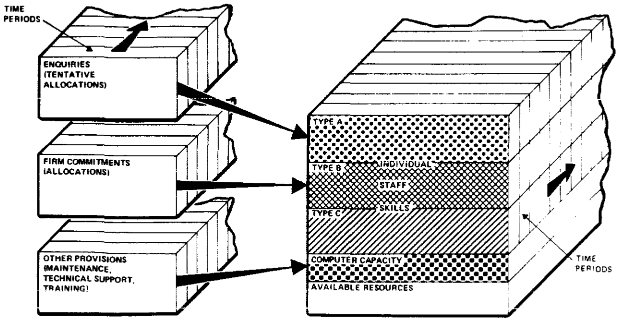

TIME PERIODS
ENQUIRIES (TENTATIVE ALLOCATIONS)
FIRM COMMITMENTS (ALLOCATIONS)
OTHER PROVISIONS (MAINTENANCE, TECHNICAL SUPPORT, TRAINING)
TYPE A
TYPE B
INDIVIDUAL
STAFF
SKILLS
TYPE C
COMPUTER CAPACITY
AVAILABLE RESOURCES
TIME PERIODS

Figure 2—The planning problem

The well tried methods are, however, not in themselves sufficient, as preparation of individual networks and estimates for each job is a problem and special techniques of network construction from pre-stored libraries are employed. Similarly, the established methods of resource allocation need special treatment to observe the relative priorities attached to individual projects and to ensure the (more efficient) continuity of work on similar jobs. To accomplish this, a form of multiproject/residual resource scheduling is employed.

The main features of this system are now described.

THE PLANNING SYSTEM

General approach

The general system flow is shown in Figure 3. Here it will be seen that there are two inputs (firm commitments and enquiries) and one principal output (quotations). Between

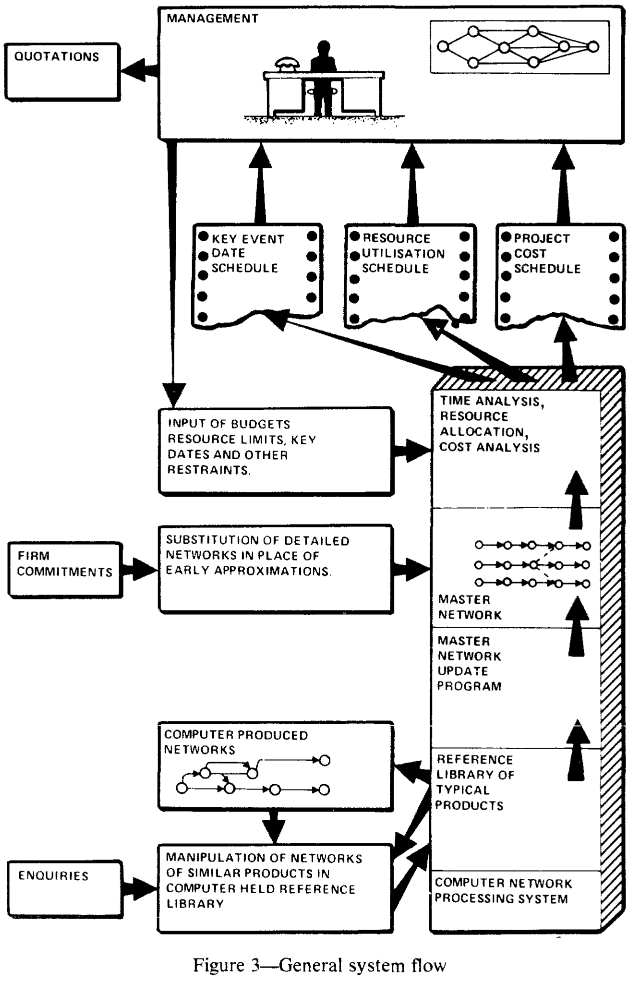

MANAGEMENT
QUOTATIONS
KEY EVENT DATE SCHEDULE
RESOURCE UTILISATION SCHEDULE
PROJECT COST SCHEDULE
INPUT OF BUDGETS RESOURCE LIMITS, KEY DATES AND OTHER RESTRAINTS.
TIME ANALYSIS, RESOURCE ALLOCATION, COST ANALYSIS
FIRM COMMITMENTS
SUBSTITUTION OF DETAILED NETWORKS IN PLACE OF EARLY APPROXIMATIONS.
MASTER NETWORK
MASTER NETWORK UPDATE PROGRAM
COMPUTER PRODUCED NETWORKS
REFERENCE LIBRARY OF TYPICAL PRODUCTS
ENQUIRIES
MANIPULATION OF NETWORKS OF SIMILAR PRODUCTS IN COMPUTER HELD REFERENCE LIBRARY
COMPUTER NETWORK PROCESSING SYSTEM

Figure 3—General system flow

these two extremes the planning system is an interaction between management, planners and a network processing system.

Central to the system is the "master network" which holds (in computer store) the events and activities of all work (firm and tentative) which is currently in the system. This is a multi-project network and it defines all project relationships, external restraints and relative priorities, as well as carrying time, resource and cost data appertaining to all stored networks.

The master network is maintained by adding and subtracting projects with the master network updating program. New sub-networks (one for each project) are added, either after being built up from the network reference library held in computer store, or from direct input of new material. Library networks are usually employed at the tentative stage when little is known about the project and personalized networks are submitted when the work has been studied in detail.

As the library methods means that a considerable part of network creation and editing takes place out of sight within the computer, network diagrams are not available in the usual way and it is necessary to provide means of visually checking the created network that will subsequently be submitted for processing. The system therefore provides for networks to be output on a digital plotter for this purpose. Comprehensive coding is built into the library of subnetworks so that selected portions can be examined in this way.

Network analysis is carried out in the normal manner and a series of resource allocations are performed. Usually both time limited and resource limited allocations are carried out with graded project priorities being applied by means of a residual resource scheduling technique. The results of selected allocations are then costed.

The main print outputs utilized are a key event date schedule, a resource utilization schedule, and a project cost analysis. These are interpreted by management for acceptability. The term "interpreted" is used deliberately in this context as management judgment must be exercised to compensate for the coarseness of the original estimates and the precision of the scheduling process.

The criteria by which the results are considered are:

(a) Acceptability of completion dates and project costs
(b) Effective (and sensible) utilization of manpower
(c) Minimal risk of disturbance by outside constraints, e.g., dependencies (which may be unreliable).

A satisfactory plan is achieved by a series of iterations in which various factors are varied and the network reanalyzed. Typical changes at this stage are:

(1) Variation of basic network relationships and dependencies
(2) Alterations to time and resource estimates on individual activities
(3) Changes to resource availability levels.

When a satisfactory set of results have been obtained, the responsible manager finally "interprets" the computer prognostication into a "quotation" which is dispatched to the enquirer.

In this description emphasis has been placed on the role of the manager in this cycle. The computer system only acts as a tool of management and does not automatically produce computer printed results which are followed blindly. The system acts as an extension of the manager's intellect; he manipulates it, he makes the major planning decisions, and he authorizes (and is responsible for) the results. Some features of the system are now described in more detail.

Network creation

In order to perform network planning it is first necessary to have a "network." This self-evident truth can, however, be qualified for computer processing to say that the necessity is for a "numerical representation of the network."

This modification, which acknowledges the nature of computing, is the basis of network library techniques. In this method, a number of module networks are reduced to digital form and stored on magnetic media (discs or tape). They can be recalled on command and edited with a minimum of effort on the part of the planner. The manipulation of library networks is shown diagrammatically in Figure 4.

In the system described, all computer programs are divided into "modules" and the overall network for the project is formed by combining a series of sub-networks, each representing a module. A library of standard module networks has been established to cover the main type of product encountered, e.g., large, medium, small, short (intensive working) timescale, long (low resource) timescale etc. A typical example of a library network is shown in Figure 5.

The planner selects the best combination of library networks and combines them together with linking dummy activities. During the selection of library networks, the planner will consult with production management to confirm the choice and identify any changes which are necessary. The library network is completed with time and

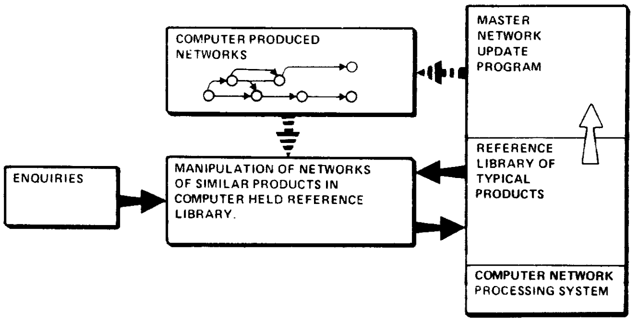

COMPUTER PRODUCED NETWORKS
MASTER NETWORK UPDATE PROGRAM
ENQUIRIES
MANIPULATION OF NETWORKS OF SIMILAR PRODUCTS IN COMPUTER HELD REFERENCE LIBRARY.
REFERENCE LIBRARY OF TYPICAL PRODUCTS
COMPUTER NETWORK PROCESSING SYSTEM

Figure 4—Network creation

resource estimates and thus changes to these may be necessary as well as the addition or deletion of activities.

Sometimes the changes will be of a "blanket" nature (i.e., increase all durations (and/or resources) by x percent) or only to specified activities. One convenient change is the addition of a "hammock activity," so called because it "swings" from end to end of the network. An example of a hammock activity can be seen in Figure 7 (between events A3–D3). No duration is specified on a hammock activity but a resource applied to this can be spread evenly over the project duration on a rate per time unit basis or given as a fixed amount, for the whole project.

Certain events on the module network are identified as "key events" for the purpose of summarizing the project time plan. These key events correspond with the same events on the "master network." This relationship is shown diagrammatically in Figure 5. Although the master network exists internally within the computer in full detail, it is summarized for management use. This relationship between module networks and the Management Summary Network is shown in Figure 6.

The Management Summary Network is, in fact, a skeleton of the Master Network which in turn represents the entire collection of sub-networks and exists for the purpose of providing management with a reference chart of all projects being analyzed simultaneously. The system is, therefore, multi-level in concept and the Master Network, together with a key event date report (from the computer analysis), provides a convenient reference for higher management. Key events are, of course, also the major milestones for reporting progress.

It is perhaps a weakness of the library technique of network creation that it is necessary to maintain a record of the changes made to library networks and some of the visual impact of the planning diagram is lost. One answer to this difficulty is the use of digital plotter output to recall the network as it stands after modification.

Digital plotter output

The manner in which digital plotter output can be used in the system is shown in Figure 4, and an example of digital plotter output is shown in Figure 7. (This sample has been simplified for illustration purposes.) Here the library network shown in Figure 5 has been modified to add the Hammock Activity (connecting events A3-D3) representing a supervisory overhead uniquely chargeable to this module. Points of interest from this illustration are that different types of line (continuous, broken and chain dotted) are used to represent activities, dummy activities, and hammock activities. Also different event shapes are used to identify start events, end events and key events.

When one has become familiar with the somewhat stylized representation produced by the digital plotter, a considerable amount of useful information can be quickly assimilated from these diagrams.

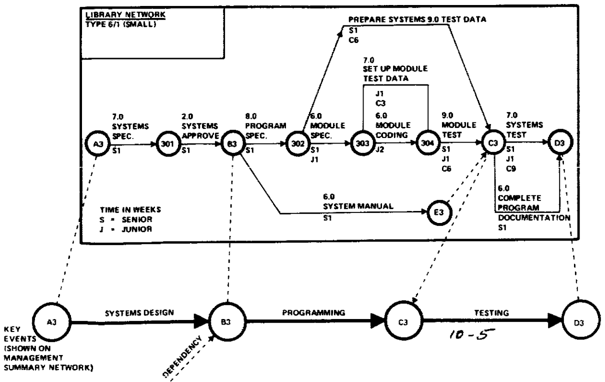


Figure 5—A library network (showing key events)

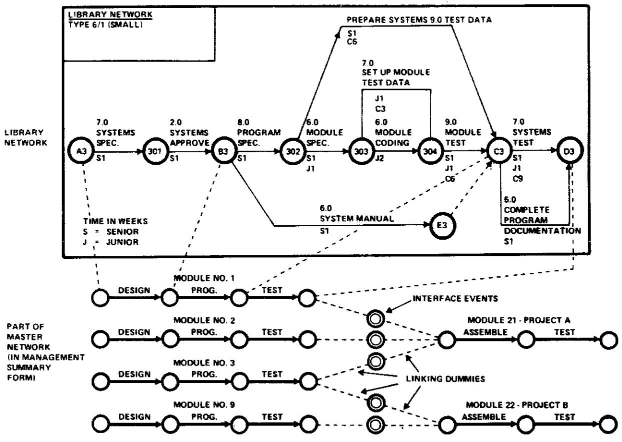


Figure 6—Relationship between module networks and master networks

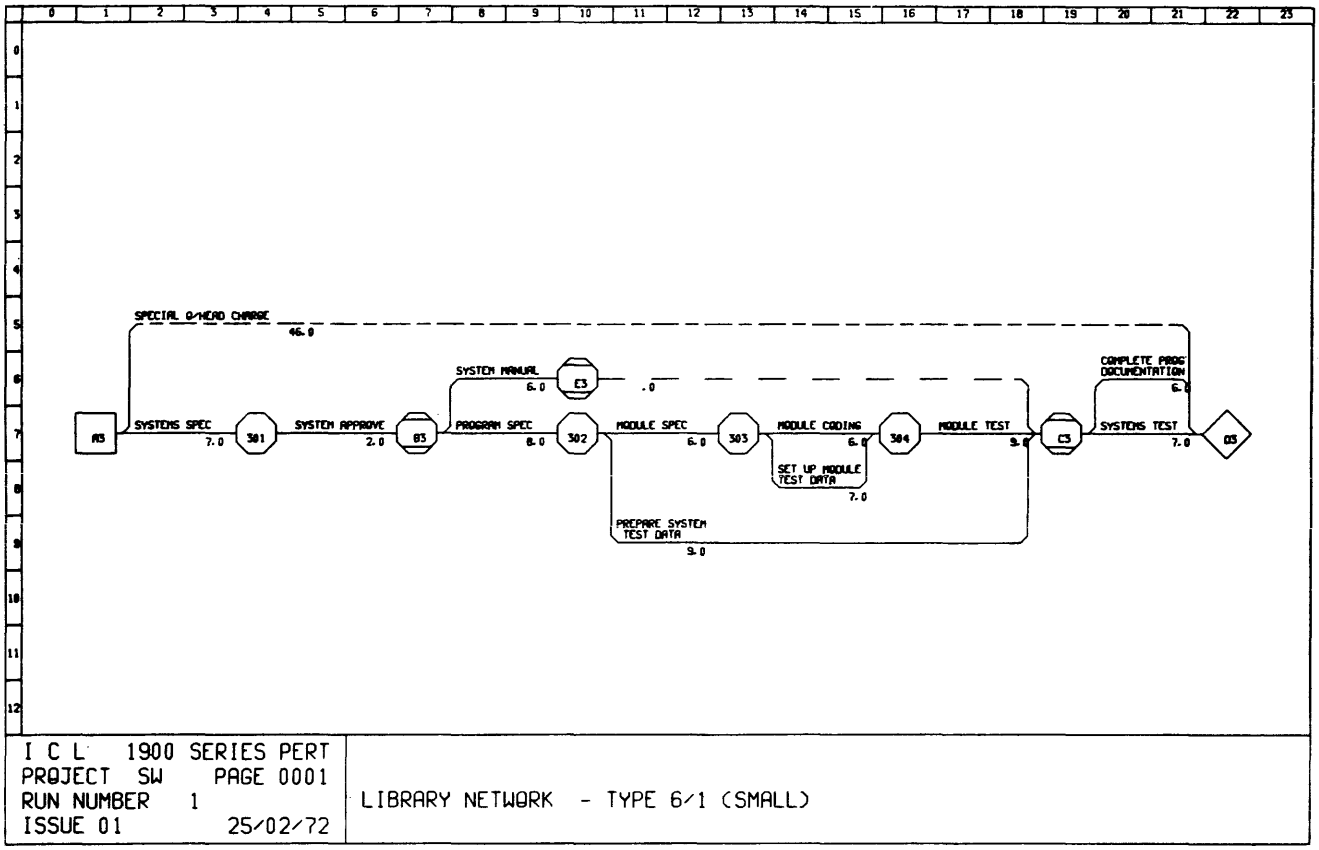
External dependencies

Computer program development work can seldom be carried out in complete isolation from external factors. This means that the work schedule must be constructed in a way which takes account of all external dependencies (or constraints).

Typical of these are completion date promises already made, dates of availability of specifications, special software or special hardware from customer or other work groups. This information is entered on the appropriate events, either as early or late imposed dates in the normal way.

Work scheduling technique

Operational managers who have used computer based resource allocation for the construction of work schedules will know that problems arise if blind obedience is given to the computer printed page. Computer programs have an infinite capacity for devising intricate work schedules yet sometimes miss the obvious. For example, no computer program will detect that a few hours overtime, worked in the weeks preceding an individual's three week holiday,



I C L 1900 SERIES PERT
PROJECT SW PAGE 0001
RUN NUMBER 1
ISSUE 01 25/02/72

LIBRARY NETWORK - TYPE 6/1 (SMALL)

Figure 7—Modified library network produced on digital plotter

could save three weeks on a project completion date by releasing a critical dependency, or how activities can be shortened by further sub-division of work in time critical areas.

Scheduling is, therefore, an interactive process between the manager who innovates and the resource allocation program which merely calculates. Without the aid of a computer the manager/planner is forced to think the whole problem through. Computer assistance is not sufficient reason to give up thinking altogether, and the thoughtful manager can bring individual style to the resulting work schedules even when computer programs are used for resource allocation. This interaction between the manager and the computer program is a feature of the system. The established methods of planning network scheduling have been described in general[1] but in this system several additional methods are employed to control the determination of the work schedule.

Work priorities

Firstly, the master network is brought up to date and checked out for data errors. All external dependencies are defined and time and resource calendars (holidays and overtime etc.) are established. Next the various projects are assigned to priority groups. Work already started and having near completion dates are given second priority. New committed work of a nonurgent nature is the third priority, and tentative work is al.ocated the two final priorities. In the scheduling process the priority groups (each containing several projects) are treated as separate projects which are scheduled against a common pool of resources. All projects in the first priority are scheduled in their entirety before those in the second priority are considered, and thus first priority projects obtain first claim on the available resources. During the scheduling of the first priority items, the resource availabilities are diminished by the extent of the quantity allocated and the remainder is passed on as the availability for the second priority projects. A similar procedure is followed for each successive priority category.

The process is termed "residual resource allocation" and is shown diagrammatically in Figure 8. It produces schedules which reflect the manner in which computer programming projects are conveniently run. By changing the priority associated with individual projects dramatic alterations to the resulting work schedules can be effected.

Work continuity

Computer programming requires intense concentration and the most effective way of employing the specialist skills

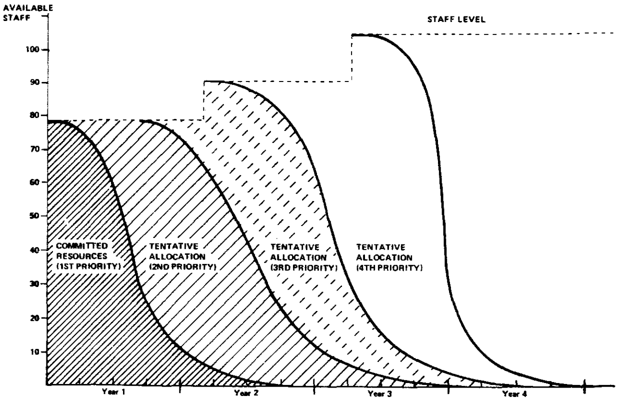
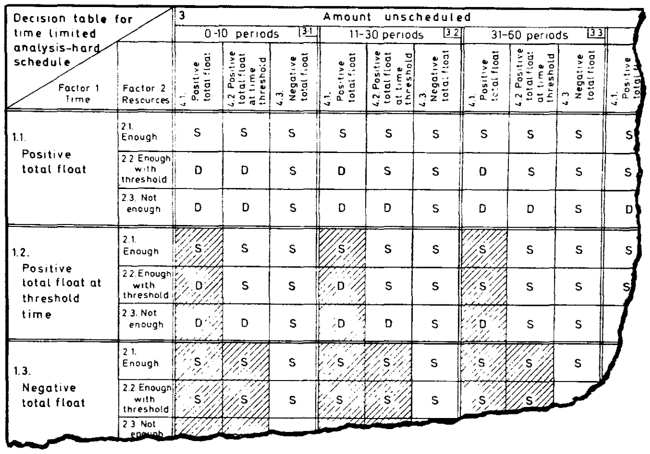


Figure 8—Scheduling residual resources



Decision table for time limited analysis-hard schedule		3 Amount unscheduled									
		0-10 periods 3.1			11-30 periods 3.2			31-60 periods 3.3			
Factor 1 Time	Factor 2 Resources	4.1. Positive total float	4.2 Positive total float at time threshold	4.3. Negative total float	4.1. Positive total float	4.2 Positive total float at time threshold	4.3. Negative total float	4.1. Positive total float	4.2 Positive total float at time threshold	4.3 Negative total float	4.1. Positive total
1.1. Positive total float	2.1. Enough	S	S	S	S	S	S	S	S	S	S
	2.2 Enough with threshold	D	D	S	D	S	S	D	S	S	S
	2.3. Not enough	D	D	S	D	D	S	D	D	S	D
1.2. Positive total float at threshold time	2.1. Enough	S	S	S	S	S	S	S	S	S	S
	2.2. Enough with threshold	D	S	S	D	S	S	S	S	S	S
	2.3. Not enough	D	D	S	D	D	S	D	S	S	
1.3. Negative total float	2.1. Enough	S	S	S	S	S	S	S	S	S	
	2.2. Enough with threshold	S	S	S	S	S	S	S	S		
	2.3 Not enough										

Figure 10—Segment of a typical decision table

involved is to establish a work group and then to minimize the distractions until the task is completed. Within the work scheduling program individual network activities, groups of activities or even whole networks can be designated "non-splittable" and thus maintain a cohesive effort on particular projects.

The scheduling decision

At first sight, the decision to schedule would appear to be a simple comparison between the resources required and the resources available at a particular point in time. In practice, however, the scheduling decision is surprisingly complex as it depends upon more than just resource availability. The condition of the other segments of the activity schedule must be considered as must the effect on the project end date of any scheduling delay. In the method described, this process is systemized by employing decision tables for the scheduling decision. The scheduling decision in respect of each activity is made by calculating various factors about the activity and looking up the decision (whether to schedule or delay) in a decision table which contains the appropriate decision for the situation defined by the calculated factors. This process is shown diagrammatically in Figure 9 and an extract from a typical decision table is shown in Figure 10. According to the resources available and the information obtained from this decision table, an activity is then either "scheduled" or "delayed." In this context "delaying" means leaving for scheduling at a later date.

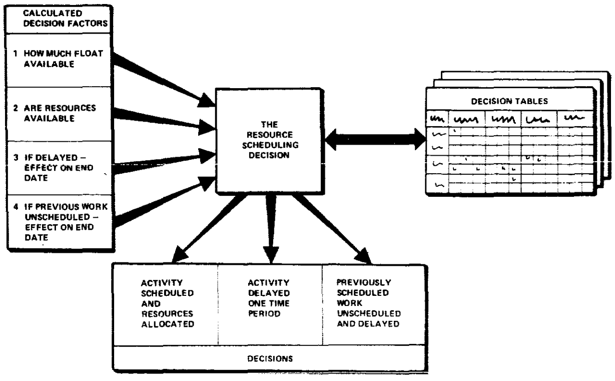

CALCULATED DECISION FACTORS
1 HOW MUCH FLOAT AVAILABLE
2 ARE RESOURCES AVAILABLE
3 IF DELAYED - EFFECT ON END DATE
4 IF PREVIOUS WORK UNSCHEDULED - EFFECT ON END DATE
THE RESOURCE SCHEDULING DECISION
DECISION TABLES
ACTIVITY SCHEDULED AND RESOURCES ALLOCATED
ACTIVITY DELAYED ONE TIME PERIOD
PREVIOUSLY SCHEDULED WORK UNSCHEDULED AND DELAYED
DECISIONS

Figure 9—Resource scheduling—Use of decision tables

The use of decision tables in this way provides an heuristic approach to the scheduling problem in that known successful decision patterns can be recorded (by means of a decision table) and then subsequently re-used for the production of further work schedules. Thus, empirical experience can be encapsulated within the computer program at the discretion of the manager. The decision tables can, of course, be referenced separately and thus the manager has another method by which to steer the formulation of work schedules into patterns acceptable to him.

Management information

Both "time limited" schedules and "resource limited" schedules are prepared and the most commonly used print-outputs at the planning stage are key event schedules and resource histograms. When they are found to be unsatisfactory, "problem modification" takes place. In this the planner and manager hold discussions on alternative variations to the network, resource availabilities, scheduled dates and priorities. Integration of management and the network planning system is greatest at this point and the flow of information is shown in Figure 11. In this interactive process, management skill is employed in suggesting acceptable variations to data and the planner's knowledge is used to anticipate the effect of changes. Several alternatives may be set up in this way and put collectively to the computer program and the resource allocation process repeated until satisfactory solutions are found.

Cost planning

When a satisfactory work schedule and utilization of resources has been achieved, the PERT Cost module is

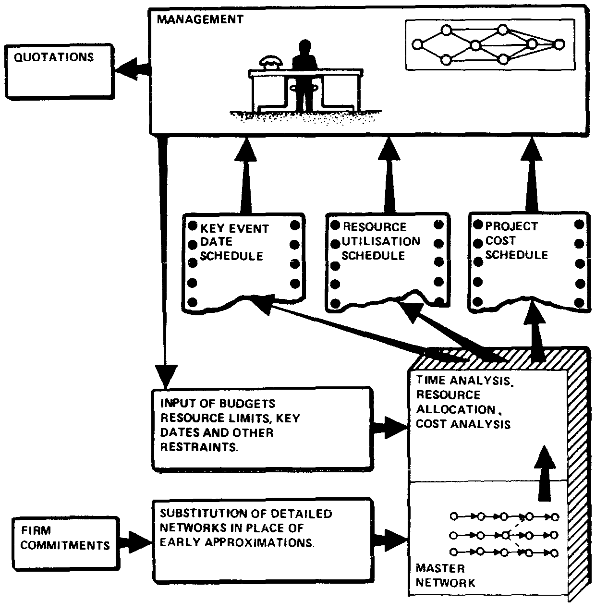

MANAGEMENT
QUOTATIONS
KEY EVENT DATE SCHEDULE
RESOURCE UTILISATION SCHEDULE
PROJECT COST SCHEDULE
INPUT OF BUDGETS RESOURCE LIMITS, KEY DATES AND OTHER RESTRAINTS.
TIME ANALYSIS. RESOURCE ALLOCATION. COST ANALYSIS
FIRM COMMITMENTS
SUBSTITUTION OF DETAILED NETWORKS IN PLACE OF EARLY APPROXIMATIONS.
MASTER NETWORK

Figure 11—Management/network planning system interaction

activated to produce individual project costs and cash flows associated with the total work load. The individual project costings form the basis of project budgets and the overall cash flows provide a comparison with departmental budgets.

CONCLUSION

The system described has established its worth as a management tool over several years. The number of projects controlled by the system at present is over 60 and collectively they represent a total value exceeding £3 million.

The planning of computer program production is a difficult task in which unforeseen pitfalls abound. Therefore, no claims are made for the automatic production of work schedules which can be followed absolutely. Similarly all calculated results are "vetted" and often "adjusted" before being offered for commitment. This qualification is, however, in no way intended to denigrate the system, but to accept that the unforeseen is unknown and allowance must be made for it.

It is a premise of this paper that the system is a tool of management—not a management system. The materials of the planner's craft are quantitative estimates of future activities and probable logical sequences. The manager must shape them as he can.

What is, however, claimed is a way of harnessing the advantages of network planning, and through its disciplines the unique power of the computer, for the better management of large scale computer program production.

REFERENCE

Woodgate, H. S., *Planning by Network,* (3rd Ed), Business Publications, London, 1977.

Test planning

by R. DEAN HARTWICK
Logicon, Inc.
San Pedro, California

ABSTRACT

The test procedures and program verification methods that should be used in planning a software development are presented. Planning considerations cover initial determination of test objectives, test planning criteria, the use of test tools based upon the anticipated design and application error set, and test standards. The presentation is directed to the manager who has had some software experience and wishes to be thorough in preparation. The test methodology stressed is test being performed parallel to program development, starting with an early (prior to code generation) analysis of program requirements and specifications, followed by static analysis of source code, execution analysis of computer program subelements, and integrated system testing. Included is a discussion of automated tools now being used to relieve test analysts of tedious code analysis tasks. Results of a study in which errors from 11 previous verification and validation projects were collected and categorized by severity and functional type are presented.

INTRODUCTION

The testing of computer programs is concerned with finding program errors that will unacceptably degrade program performance. Test planning often consists of organizing an increasingly more complicated series of tests after program development, following the rather simplistic notion that: "Errors will be introduced during program analysis, design, and coding and may be found (so that they may be eliminated) by program testing." In contrast, a well-planned test approach will start in parallel with program development, making it possible to introduce design and coding constraints that will assist testing, to plan the testing effort properly, and to detect errors at the earliest possible time so that they may be corrected with minimum time and effort. This concept looks at testing as a means of safeguarding program quality rather than as a way of measuring program errors.

This approach is desirable for two reasons: to lower the likelihood of budget and schedule overruns, and to minimize testing costs. Many computer programs have been developed late, cost more than budgeted to complete, and have worked so poorly as to degrade entire system performance; examples are contained in the well-known CCIP-85 study.[1] By catching design errors earlier, major iterations in program development may be reduced, lowering the probability of schedule and budget overruns. An additional advantage is that building quality mechanisms into a program should improve its performance when fielded. With regard to testing costs, these are frequently reported to be in the vicinity of 50 percent of the entire computer program development cost.[1-3] Considering test requirements early makes it possible to prepare a more efficient plan in which appropriate test tools and personnel are identified. Additionally, finding design errors at an early stage saves time for both program development and testing.

The remainder of this paper discusses how such a well-organized test plan is prepared. Considerations include the testing methodology to be employed as a function of phase of the program development, the selection of test tools to be used, and the organization and control of the test effort.

TEST OBJECTIVES

As software systems have grown in size and complexity, more programmers have been required, more personnel interfaces have been introduced, and more opportunities for error have been created. All of these factors have led to a nonlinear increase in the number of ways a computer program can malfunction, and the resulting software horror stories have received a far better press than the systematic science of computer program testing developed to prevent them. This systematic science has been given several different names, including Independent Verification and Validation (IV&V). The concept of IV&V has become widely accepted within the Defense Department and has proven its value; remarkably few software developments have failed to achieve their desired quality or have violated schedules and budgets when IV&V was used. The essential difference between IV&V and a formal test group is managerial. IV&V is performed by *independent* personnel employing *independent* test tools and techniques. Although this distinction will not be maintained hereafter and the remaining discussion will strictly address test methodology, IV&V should always be considered as a technique to be used in any new software development, even though its cost may seem to be high (typically 15 percent to 20 percent of the total software development cost). The immediate out-of-pocket expense associated with doing IV&V must be weighed against the assurance of quality and performance gained when it is used.

The fundamental step in preparing test plans that will help achieve high computer program quality is to state the test objectives at the outset of the development. The test objectives, which are independent of program size, significantly influence what test methods are to be applied to a given program, that is, what types of errors must be detected by testing. Test objectives encompass two different purposes: testing to gain a measure of how well the computer program will achieve objectives (performance evaluation) and testing to assure that only intended functions will occur (assurance analysis). These two purposes overlap somewhat, as shown in Figure 1, and the type of application influences the amount of effort devoted to each.

Performance evaluation establishes a measure of how well the program performs its intended functions. This measure will be a function of the application, user objectives, and expected program longevity. Some considerations in this respect might be: efficiency (timing and memory), maintainability (ease of modification), accuracy (numerical and logical), compatibility with system and user (convenience, vulnerability), and testability. The following definition of performance evaluation should prove helpful: Performance evaluation is a determination of the extent to which the examined software:

- Satisfies system requirements
- Satisfies program end item requirements
- Is designed and coded efficiently
- Degrades the performance of the system or the subsystem in which it operates

Performance evaluation will be adequate only to the extent that the system and end item requirements are defined. The process of working from system to end item requirements is also subjected to scrutiny by performance evaluation; for the end item requirements, when combined with requirements for other system components, may conflict with the system requirements. It should be pointed out that the third and fourth items above are very subjective and may involve tradeoffs; inefficiency and system degradation are usually demonstrated by counter-example.

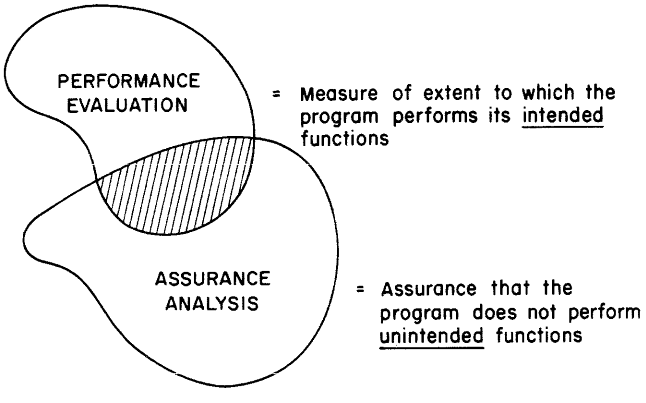

PERFORMANCE EVALUATION
= Measure of extent to which the program performs its intended functions
ASSURANCE ANALYSIS
= Assurance that the program does not perform unintended functions

Figure 1—Test objectives

Turning now to assurance analysis, the objective is to show that the computer program performs all intended functions and does not perform unintended functions that could degrade or compromise the safety or security of the system to which it belongs. The definition of assurance analysis might be generalized for all types of computer programs as follows. Assurance analysis is the determination of the extent to which the examined software contains coding which could contribute to:

- Unauthorized access to data files or program
- Unauthorized display of confidential data
- Failure to respond in a timely fashion to critical program conditions
- Operations that present a hazard to equipment or personnel

The objectives of performance evaluation and assurance analysis are to some extent interdependent. There will be program errors which have an impact on safety/security as well as on performance.

The management techniques, analysis tools and techniques, personnel qualifications, and configuration control procedures are basically the same for performance evaluation and assurance analysis. Only by assessing the potential impact of a program error can that error be categorized as related to the system's performance or to its safety or security. The errors are not distinguished by the means of discovery.

TEST PLANNING AND ORGANIZATION

Testing can be divided into six major management phases whose time-phasing relative to a typical system development cycle is illustrated in Figure 2:

- Program and personnel planning
- Test requirements definition
- Tool definition and development
- Test plan/procedures definition
- Testing and analysis (consisting, as will be shown subsequently, of program concept analysis, static code analysis, and code execution testing)
- Final report generation

As shown in the figure, the phases may overlap. Therefore, to prevent significant management problems it is important that the program and personnel planning phase be completed before undertaking any other phase. Detailed schedules for milestones and supporting activities should be established for the entire effort. These schedules should be detailed enough to provide management with a tool for measuring project progress, but also flexible enough to permit reaction to unanticipated problems. If the schedules have been properly established, project management will be

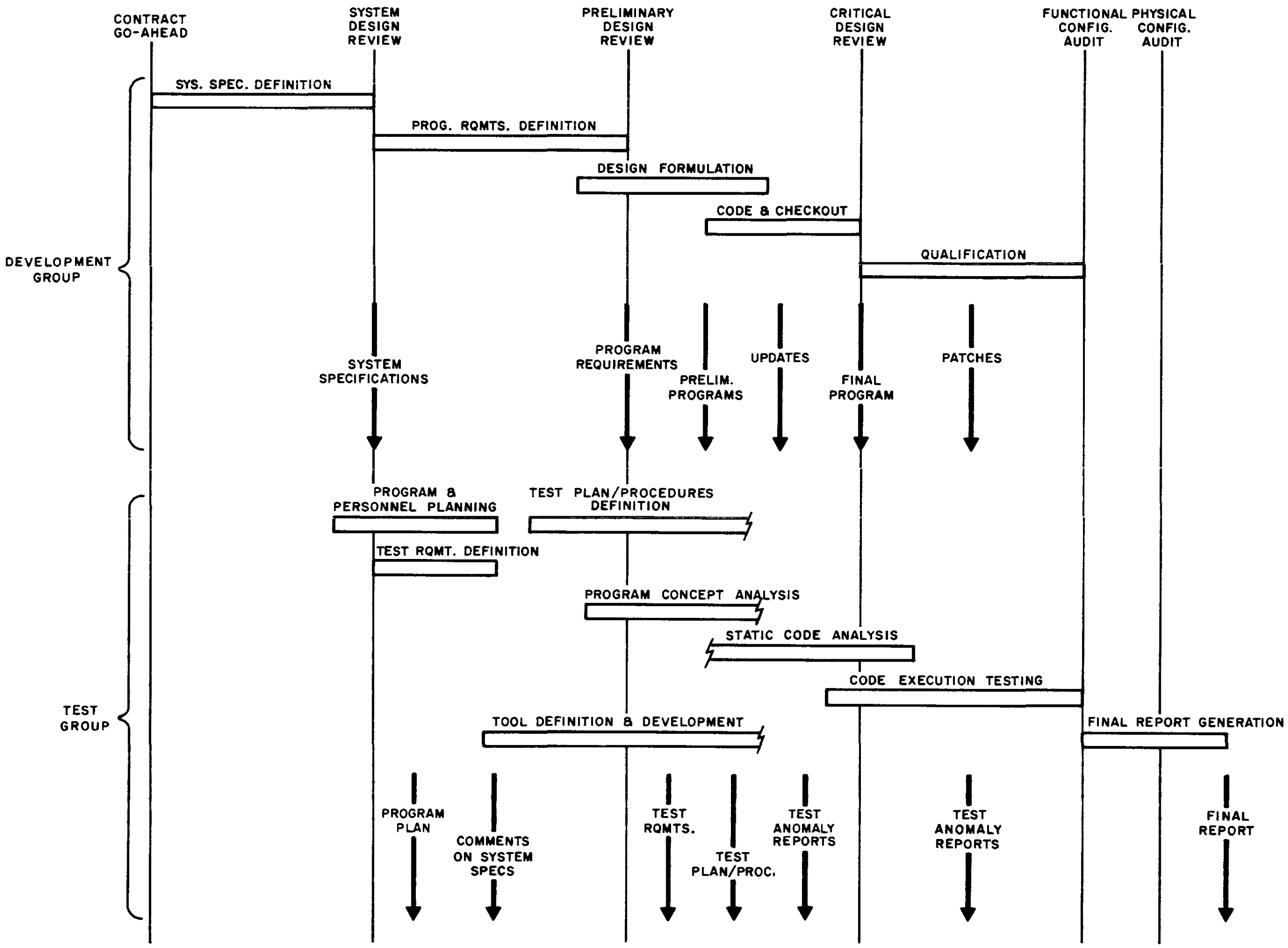


Figure 2—Test management milestones

able to detect potential problem areas before they become critical. The schedules should reflect the following types of information:

- Major project design and technical review
- Dates and contents of data package deliveries
- Dates of deliverable data items and other major milestones, allowing for rough drafts, internal review, editing, final copy preparation, and review and approval
- Dates for the completion of important test support software tools, including definition dates for the test tool requirements, design flows, description document, and user's manual
- Schedule dates and management approaches to ensure the timely review of activities that can only be scheduled upon completion of prerequisite milestones

Once the schedules have been prepared, each supporting activity can be manloaded and the qualifications of people to staff these activities established.

Test reviews, in which test progress is presented using the activity and milestone charts, should also be planned. Near-term milestones, usually those to be accomplished within the next three months, might require an inch-chart review—a weekly or daily breakdown of activities showing how the near-term milestone will be achieved. The inch-chart review ensures that nothing has been overlooked. Also, potential project pitfalls or failure conditions should be reviewed so that the necessary actions (for example, additional project staffing) can be immediately initiated to head them off.

Test requirements should be generated in the second management phase, test requirements definition. For assurance analysis, this entails clearly identifying program requirements whose violation could compromise system safety or security. For performance evaluation, it entails identifying the program requirements to be examined to measure the quality of program performance. Establishing the scope of what is and what is not to be tested is imperative because testing all program combinations is infeasible. As will be discussed, many of the requirements must result from an analysis of the types of errors that may be anticipated and the test methodology required to detect them. A high-level test methodology must be established here. It will then be amplified in defining test plans and procedures.

In the third management phase, tool definition and development, the test tools that will best accomplish the test objectives should be identified. The choice that management makes in its selection of test support software tools will be reflected directly in the quality and productivity of the test group. The advantages of automated test techniques and analysis methods are obvious, but frequently the project will lack sufficient time or money. Once the test support software tools have been identified, the test management team must establish the method of qualifying them, the configuration control procedures to be applied to them, and the schedule for time-phasing them into the analysis and testing effort.

In the fourth management phase, test plan/procedures definition, the details of how each test requirement will be tested and analyzed must be documented. The document should include the following types of information:

- Test support software or hardware test bed to be used
- Test scenarios
- Success criteria
- Detailed procedures for test implementation
- Control of data standards and software deliverables

The last item, control of data standards and software deliverables, is especially important. Test management should institute strict procedures to ensure that the data standards are not replaced with others, tampered with, or destroyed. At the completion of testing and analysis, a final check of all the working copies against the data standards should be performed.

In the fifth management phase, testing and analysis, the management task becomes one of monitoring project performance and maintaining control procedures on all input and output items. Test management should review all discrepancies issued by the test group and evaluate each for correctness. Whenever the changes are made to program requirements, program code, or support tools, a decision must be made whether it will be necessary to repeat tests and analyses already completed. The possibility of retest makes it imperative to maintain careful configuration control on the program requirements, program code, and support software.

As Figure 2 shows, the testing and analysis phase is broken down into three subphases: program concept analysis, static code analysis, and code execution testing. The functional procedures and tools used during each are discussed subsequently under selection of test methodology.

The sixth management phase, final report generation, may include certification of the program for operational use integrally or as a separate step. During the planning for the analysis and testing phase, management should establish procedures for recording results in a form usable for final report generation with minimal modifications. This effort allows a final review of activities to ensure that no test has been overlooked. For future efforts, it would be useful to accumulate statistical information about the number of errors detected, their relative importance, and the methods used to detect them.

ERROR SOURCES AND DETECTION METHODS

Software errors cannot be treated within the confines of hardware reliability concepts, nor can it be assumed that the detectability of computer program errors has profound theoretical considerations. A good test plan is based upon sound software error theory. Every application can be expected to have its own unique set of errors. It is the role of the test group to determine this error set; from it, the type and extent of testing to be performed can be determined.

Considerable work has been performed in accumulating and analyzing data to develop a software error theory. One study that will be illustrated here represented the results of analyzing and categorizing 1202 errors discovered in 11 projects.[4] These results are primarily, but not totally, obtained from IV&V activities and hence do not show how earlier development testing would affect results. The results were categorized by 12 different error types and classified by error severity, as shown in Table I. The severity levels were based on program performance, as follows; the greater the performance degradation, the more serious the error:

- *Catastrophic Error:* A coding error that would be fatal to the application in that it would effectively terminate program execution
- *Serious Error:* A coding or specification error that could severely degrade the program performance but would not be fatal to the application; examples are violations of timing, accuracy, safety, or stability requirements
- *Moderate Error:* An error that would not have major impact on program performance and that would probably not result in a system requirements violation
- *Trivial Error:* An error that would have no impact on program performance, e.g., errors within annotations on the program listing

For purposes of test planning, interesting observations can be made from an examination of Table I. For example, even though errors of the "branch & jump" variety occurred infrequently after checkout (4.5 percent of total errors), they tended to be serious (14.3 percent of catastrophic and 6.7 percent of all serious errors). Thus it may be concluded that any testing methodology to be selected should ensure that this type of error should be well analyzed and accounted for. At the other extreme are the "documentation" errors, which accounted for 7.9 percent of the total errors but no catastrophic or serious errors. This then might influence a test planner to reduce effort put into finding documentation errors in favor of intensifying effort elsewhere. One caveat to any such statistical approach can be seen by looking at the "incomplete or

TABLE I—Error Categorization Summary

Error Category	Total Errors	Catastrophic Errors	Serious Errors	Moderate Errors	Trivial Errors	% of Catastrophic	% of Serious	% of Moderate	% of Trivial	% of Total (excluding trivial errors)
Data/instruction access & storing	120	6	30	72	12	28.6	20.1	15.0	2.2	16.6
Equation computation & arithmetic	113	0	22	73	18	0	14.8	15.3	3.3	14.6
Branch & jump	32	3	10	16	3	14.3	6.7	3.3	0.5	4.5
Incorrect constant value & data formats	41	2	12	19	8	9.5	8.1	3.9	1.5	5.1
Violation of programming practices	118	0	2	22	94	0	1.3	4.6	17.0	3.7
Specification violation due to incorrect implementation	145	0	9	61	75	0	6.0	12.7	13.6	10.8
Timing & process allocation	44	0	14	25	5	0	9.4	5.2	0.9	6.0
Interruptibility & data coherency	45	2	10	32	1	9.5	6.7	6.7	0.2	6.8
Incomplete or erroneous specifications	340	1	18	82	239	4.8	12.1	17.1	43.2	15.6
Logic & sequencing	107	6	22	67	12	28.6	14.8	14.0	2.2	14.6
Documentation	96	0	0	10	86	0	0	2.1	15.6	1.5
Erroneous use of system hardware/software	1	1	0	0	0	4.8	0	0	0	0.2
Total	1202	21	149	479	553					
Total % (excluding trivial errors)		3.2	23.0	73.8						

erroneous specifications" category. Although this class contributed 43.2 percent of all trivial errors, one catastrophic error was found. It takes only one such error to totally degrade the usefulness of the program.

Once the test planner has a feeling for the types of errors that may be found within a particular program and understands the test objectives, he next must consider the test methodology to be used. It is useful to think of this methodology in terms of testable logical groupings within the computer program and the methods that can be used to detect the class of errors.

Table II gives the general test methods used to detect each category of error in Table I. Referring to Table II, it is clear that a variety of techniques may be needed to test even a single module. At a higher level of program complexity, the executive routine, the various functional modules are interconnected. Generally, the executive itself is modular; i.e., portions can readily be separated so that they can be used with the functional modules to form a functional subprogram. The modularity of the executive can generally be demonstrated by use of a flowcharter to construct a module-level flow diagram from the actual

TABLE II—Examples of Specific Detection Methods by Error Category

Error Category	Document Comparisons	Independent Coding & Simulation	Equation/Algorithm Derivation	Logic Analysis	Edit Source Code	Code Analysis	Execute Code on Simulation	Execute Code on Real Computer	Perform Branch Analysis	Analyze Test Case Results	Correctness Proofs	Flowcharting	Accuracy Study Processor
Data/instruction access & storing					90%	×	×	×			×	×	×
Equation computation & arithmetic		×	×	×		×	90%	×		×	×	×	×
Branch & jump				×	×	×	70%	×	×	×	×	×	×
Incorrect constant values & data formats					50%	20%	×	×		×			×
Violation of programming practices				×	×	90%							
Specification violation due to incorrect implementation	×					×				×	×	×	
Timing & process allocation				×			90%	×		×			
Interruptibility & data coherency			×		70%	×	×		×				
Incomplete or erroneous specifications	×	×	×	×			×	×		×			
Logic & sequencing				×		×	×	×	×	×	×		
Documentation	×		×		×	×							

× indicates at least one error discovered
Number indicates percentage of total errors in category found by this method

code. Such a diagram helps to demonstrate the correctness of both functional interrelationships (connection by data flow) and physical relationships (overlay of common memory).

Another effect becomes clear from an examination of Table II. It is extremely unlikely that any particular test method will assure a complete assessment of all errors. Even where a particular method is most suited to detect a particular class of error, it cannot be assumed sufficient. For example, even though a source code editor detected 90 percent of all access and storing errors, 12 errors were found by a combination of six other methods. Therefore, the test planner must consider all test methods available to him and select the set of the methodology that will provide adequate test coverage—we have not yet reached the nirvana of having the one magical tool that will find all errors.

SELECTION OF TEST METHODOLOGY

It has often been suggested that the most cost-effective way to test a program is to load it into its host computer, exercise it black-box fashion, using a myriad of inputs, and determine whether its behavior is reasonable. While this approach seems enticing, it is overly simplistic in that it ignores the complexity existing in even very small programs. Further, black-box testing necessarily means that errors will be found only at the completion of coding. As stated earlier, the most cost-effective way to develop software is to find and fix errors as early as possible. Cost-effective testing therefore requires that some of the testing be conducted before any code has been developed. Such testing can be conceptualized in terms of three types (as shown on Figure 2):

- *Program Concept Analysis:* Achieving satisfaction that the program has been adequately and accurately designed
- *Static Code Analysis:* Analyzing code to detect errors before its execution
- *Code Execution Testing:* Testing code by executing it in successive builds to verify correctness

The process may be likened to building a house. Each level of testing builds upon the foundation and structure established by the previous step until the house is ready for occupancy. As is true of the construction process, each phase has its own set of tools and methods.

Program concept analysis

Of the many test-related functions that can be performed before code generation, one of the more important is to assist the program designer in specifying the standards to be followed in designing and writing the program. Consistent coding techniques greatly simplify the problems of learning code and setting up test tools that can be adapted to specific coding standards, for example, the use of standard subroutine entry-exit conditions. Another precoding function is to investigate design tradeoffs that might facilitate testing without introducing substantial programming inefficiency. An example is designing a program for a real-time application on a variable-instruction-time computer in such a way that "dead time" occurs long enough before an interrupt to guarantee that the program location where the interrupt will occur can always be precisely determined in testing. By doing this it is possible to ease greatly a very difficult test program with an insignificant loss of computation time.

Other standards that can be applied include:

- *Compatible Numbering System:* Each function performed by a section of code should be traceable to a specific, documented requirement.
- *Consistent Use of Specification Language:* A common and consistent nomenclature should be defined that minimizes the possibility of ambiguous interpretations.
- *Flowcharting Standards:* All flowcharts should use common formats and be consistent through different levels of detail.
- *Programming Standards:* All dos and don'ts for the application should be specified, including parameter-passing techniques, annotation conventions, and ways of specifying symbolic names.
- *Program Change Procedures:* Means of making changes should be developed to communicate changes properly both before and after the program is put under configuration management.

A more directly test-oriented function that can be performed during concept analysis is to evaluate the evolving design to find problems. This activity, sometimes called requirements and specification analysis, essentially duplicates the design process in the way it is done but differs in the goals that are being met. This type of analysis is directed to answering questions such as those contained in Table III.

The methods generally used to perform concept analysis testing are:

- *Documentation Research and Analysis:* Verifying that program requirements are genuine and related to real system requirements, that variables and parameters are consistent across all documents, and that all data can be traced to consistent, coherent sources.
- *Algorithm Analysis:* Verifying that the design will work, is accurate, and truly reflects program requirements. Discrete and continuous simulations are often used to check out concepts and algorithms, particularly where very complex models are being designed. Particularly important for real-time programs is performing independent sizing and timing analyses that will detect significant design problems before the design is frozen.

TABLE III—Program Concept Analysis Questions

Requirements Analysis

Are the requirements logical?
Are the applicable physical constraints of the complete system and each subsystem clearly specified?
Are all the performance and design requirements contained within the document? If so, are they correct?
Have the input data and output requirements been identified?
Are the system and all subsystems clearly defined?
Are the human and software interfaces specified?
Are all the requirements stated without ambiguity?
Are the requirements sufficient to realize the system objectives?

Specifications Analysis

Are the program functional flows a true representation of the logical and mathematical operation of the software?
Are the model interactions and interfaces compatible?
Are the equations, algorithms, and data that make up the model properly ordered?
Are the equations, algorithms, models, and modules correct?
Is the data base accurate?
Have the correct assumptions been made in equation/algorithm derivation?
Are the equations and algorithms mechanized in the program flowcharts the same as those developed in the requirements documentation?
Have all the program requirements been correctly translated into the equations, data, algorithms, models, flows, logic, and rationale that make up the program specifications?

Static code analysis

The first test function always performed is to examine the code for error before actually executing it. This is useful because it is very productive (many errors are found) and also because it can be done quickly (no startup time is needed to make the program flow in a computer). A large body of test tools has been developed to aid in analyzing computer code without actually executing it. In some form or another, these tools all replace the analysis that is done by a skilled programmer in examining the code manually. This manual analysis has been replaced by automated analysis for several reasons:

- It is one of the more tedious tasks that can be placed on programmers.
- Human analysts make errors when doing tedious things.
- Computer time is much cheaper than analyst time.
- Tedious processes can be easily and cheaply repeated to check the effects of minor code changes.
- Code too complex to be retained and understood by the human mind can be automatically decomposed to its simplest representation to facilitate analysis.

The tools that are used are numerous and subject to many variations of definitions depending upon their developer. The following describes a few of the static code analysis tools implemented by the author's organization.

Comparators are used to compare the code of one program version to that of another and reveal the differences. The simple one-to-one check provided by use of a comparator is used both to bring out the modifications between an updated program and its baseline version and to demonstrate that all physically different versions of a single program (cards, magnetic tape, and Mylar tape) are identical. Conceptually very simple, these programs are complex. They work very well in setting baseline standards.

Editors are used to analyze source code for coding errors and to extract information that can be used to check relationships between sections of code. Among other functions, the error-detection capability determines whether the code: sets and clears flags properly, uses error-prone instruction sequences, sets up calling sequences properly, modifies instructions, attempts to reference or modify restricted data, uses restricted instructions, or contains inaccessible instructions. The second capability of an editor provides a comprehensive cross-reference listing giving information pertaining to references to program data and to the program/subroutine calling structures. Editors work very well in finding mechanical violations of programming standards. They can also be used to flag coding techniques determined to be risky for the application.

Flowcharters show, in detail, the logical structure of a program, an aspect not readily apparent from the code. Flowcharters can be used to reconstruct the logic flow of both higher-level source language programs and assembly language programs. The flow is determined from the actual operations specified by the executable statements, not from comments. The flowcharts generated are adaptable for comparison with the flowcharts provided in the developer's program documentation.

Logic/Equation Generators automatically reconstruct arithmetic and flowchart assembly language programs. Such a program translates assembly language instructions into a machine-independent microprogramming language and builds the microprogramming statements into a network in which the flow of control is analyzed and arithmetic equations are reconstructed.

Program Structure Analyzers are used to analyze the program paths under input control. They allow different types or sequences of code to be specified (e.g., all subroutine calls, all interrupt-related instructions, all extension register operations) and obtain information such as estimates on timing, paths followed, and entry conditions for specific paths.

Correctness Proofs establish, in a mathematical fashion, that a given program performs a desired function and halts. The proof technique does this by determining the correspondence between a function in its coded form (in FORTRAN, for example) and the same function presented in the pertinent specification in mathematical and English language descriptions. Operational use of this technique has been limited mostly to manual proofs. Considerable work is being done to automate it.[5]

Symbolic Program Executors decompose source code by logically executing it. They provide a capability to express paths in terms of both all necessary conditions to be satisfied in selecting the path and the result of transversing the path. Some symbolic program executors can also be used to produce a structured representation of an unstructed program.[6]

Code execution testing

In the traditional mode of software testing, the program is tested by actually executing the object instructions, generally on the computer it is being built for, sometimes on emulators or simulators. Code execution testing closely follows the test philosophy normally used for complex hardware, namely, to test in pyramid fashion, emphasizing the thorough testing of subassemblies, then successively larger groupings of these subassemblies until the total end item has been tested. Provided that the prior testing has been adequate and, moreover, that it has not been subverted by the incorporation of insufficiently tested changes, the testing done on the top level is essentially reduced to demonstrating that all of the pieces work together properly.

Testing methodology depends to some extent on the development methodology. Traditional developments employ bottom-up testing, in which increasing aggregates of small programs are tested in succession, generally using program drivers. Top-down developments replace program "stubs" with analytical data to simulate the effect of having executed the program element that replaced the stub. The "build" concept requires that stimulus-response patterns be determined to test out constructs of program elements that implement particular stimulus-response paths.[7] The code execution test methodology is applicable to all of these forms of testing. For simplicity, discussion here is limited to the more common bottom-up form.

Unit-level testing is performed at the subroutine level by starting code execution of subroutines as they are verified. Each subroutine is exercised by test drivers that furnish input and parameters, including representative queues. Testing at the subroutine level is performed open-loop and, in general, does not require simulation of the modules or environment with which the subroutine interfaces. Completion of subroutine testing with "live" data should demonstrate that:

- All instructions have been executed at least once
- All error conditions have been tested
- All logic branches have been traced
- All classes of input will be accepted and all outputs can be produced
- Arithmetic results are correct for nominal conditions

Test cases are generated based on an analysis of the program specification. Input data are selected to verify proper handling of the full range of acceptable data and to verify correct action in processing abnormal data. A test driver program typically sets up the input data, calls the module to be tested, and, upon return of control, lists the module output. For routines that are time-critical, the driver is augmented to check timing; the timer can be set either to determine the time to calculate one set of input data or to evaluate average, minimum, or maximum time. Whenever anomalies are suspected, the driver is patched to call upon system routines that will allow step-by-step monitoring of a process.

The unit level interface tests are designed to verify that subroutines can be loaded and executed together in the computer system, and that they properly assume and relinquish computer control. Of particular importance are tests that execute all possible branching conditions between subroutines and modules.

A supplement to actual or simulated code execution is to independently perform each operation intended by a section of code. This procedure consists of following through the calculations of an equation or algorithm. This can be done by hand, but is better done by an automated tool which can swiftly and accurately run many check cases. Using either a listing or a flowchart of a module and the data captured from a specific run, the mathematical calculations or logical processes indicated in the flowcharts (or the listings) are performed and the results compared with the code execution results. The importance of this technique is that subtle coding errors not caught during flowcharting or code analysis procedures are quickly identified.

The most common tools used to test software at module levels are test drivers, simulations, and execution instrumenters. Many examples of this last class of program exist.[8,9] They operate by instrumenting software "probes" within the source code. When the program is subsequently executed, statistics showing execution frequency of paths and code are produced. This gives insight into the program behavior and allows analysts to prepare test cases to more fully exercise the program. A more recent trend in many evaluation efforts is to use hardware monitors to gather these data. Inserting probes into computer back panels is somewhat laborious; hardware monitors allow the same statistics to be collected without modifying the computer program. Further, since they operate in a passive mode, they can be used after the program becomes operational to evaluate it and provide data to optimize it.[10,11]

The sequence of testing individual modules proceeds in order through tests such as the following:

- *Initialization Tests:* The performance of all initialization operations is tested to assure that all indicated initializations are performed and that correct values for all initialized quantities results.
- *Interaction Tests:* All quantities, variables, and system conditions obtained from other modules are examined to determine the sensitivity of the module under test to their possible values or states.
- *Arithmetic Tests:* The precision of arithmetic calculations is checked to discover where insufficient precision is maintained or incorrect arithmetic calculations are performed.
- *Timing Analysis:* The longest and shortest possible execution times for all tests are determined to establish the execution time requirements for the module and to identify potential timing problems.
- *Branch Logic Tests:* The correct branching decision paths for each branch and each closed-loop test case

are checked. The branch decision paths that are not exercised in any of the normal test cases are identified and their correctness demonstrated by special test cases.

Once the modules have completed individual code execution testing, they are integrated into the complete software component package and tested as a group. At this level, the testing is primarily functional: testing the collection of modules to show that the aggregate satisfies the stated problem. For large-scale software or weapon systems in which software is an integral part of the system, this testing very often takes place in a laboratory containing a copy of the computer(s) and enough equipment to simulate the application with considerable fidelity.

TEST STANDARDS

To assure the success of the test effort, the test group itself must be subject to quality standards. All hardware and software used in the test effort should be qualified and controlled.

The test tools can be qualified by certifying they have been previously qualified and not changed; by calibration against actual system data, other operational standards, or other qualified test tools; or by a simplified form of the more elaborate test procedures described earlier. Whatever the qualification procedure selected, it should be formally documented and approved as a part of the test plan/procedures. Satisfaction that formal qualification has been performed should be obtained before the tools are used and should be documented in the final test report. After qualification, the tools should be placed under configuration control using conventional techniques.

Where complex simulation testing is required, there is a natural tendency to modify the test hardware and software to adapt them to the peculiarities of the particular program segment under test. Any such changes cannot be allowed to compromise test integrity, such as by using different versions of test tools to test different program modules. All modifications to the test environment must be first communicated and approved. Where a test bed is to be established, it must be provided with a usable inventory of test support tools. These tools should also be formally qualified, and controlled to provide certainty as to their content.

One of the more critical aspects of testing is ensuring the correspondence of the test results obtained at different times or by different personnel. At a minimum, test results should contain the following information to allow comparison of results:

- Test designation
- Test purpose
- Specification, option, or feature being tested
- Range of parameter being tested
- Method of test
- Inputs required for test
- Output expected from test
- Estimated time for test
- Criteria for satisfactory completion of test
- Identification of test bed configuration

The final test results and supporting documentation should be bound as a single document and labeled with the name of the routine(s) checked, the test completion date, and the names of the programmer(s) and test personnel responsible for reviewing and accepting the results. Results should be accompanied by such flowcharts, diagrams, equations, and verbal descriptions as necessary to identify and clearly describe each test, including what was done, why it was done, and what specifically was demonstrated. A summary, if appropriate, should preface the test results package.

SUMMARY

The success of a software test program is determined at the outset of the development. If testing is accorded the effort due it because of its cost and ultimate importance, it is possible to achieve working software of high quality within cost and schedule. By contrast, letting a test program evolve subject to the caprices of development problems and schedules gives very little confidence that the test program will be successful in all dimensions.

At the outset of the development, the test objectives should be set and agreed to by the project manager, the development group, and the test group. These objectives should have the concurrence of the user. Test requirements should then be developed based upon the objectives, the application, and the anticipated design approach. A plan to use, and acquire as necessary, appropriate test tools based on the requirements and the predicted error set should be prepared. An arsenal of tools and test methodology exists; test planning should meticulously select that which is most appropriate. A test plan that provides appropriate interfaces throughout the development effort should be prepared. Finally, good practices pertaining to tool qualification and configuration control should be followed.

REFERENCES

1. Boehm, B. W., "Information Process/Data Automation Implications in Air Force Command and Control Requirements in 1980 (CCIP-85)," Space and Missile Systems Organization, SAMSO TR 72-122, February 1972.
2. Brooks, F. P., Jr., "The Mythical Man-Month," *Datamation,* December 1974.
3. Smith, R. L., "Estimating Software Resource Requirements," Volume XI of Structured Program Series, Rome Air Development Center, RADC TR 74-300, Vol. XI, January 1975.
4. "Verification and Validation for Terminal Defense Program Software—The Development of a Software Error Theory to Classify and Detect Software Errors," Logicon Report HR-74012, 31 May 1974.
5. Good, D. I., R. L. London, and W. W. Bledsoe, "An Interactive Program Verification System," *International Conference on Reliable Software,* Los Angeles, California, April 21-23, 1975.
6. Ikezawa, M. A. and R. E. Kayfes, "A Structural Calculus for Program Analysis and Testing," Logicon Report CSS-75019, presented at the 9th

Annual Asilomar Conference on Circuits, Systems, and Computers, Pacific Grove, California, November 1975.

7. Chandler, A. R., "Software Verification and Validation for Command and Control Systems," RCA Reprint RE-19-5-23, 12 November 1973.
8. Stucki, L. G., "Automated Generation of Self-Metric Software," *Proceedings of the 1973 IEEE Symposium on Computer Software Reliability,* New York City, April 30-May 2, 1973.
9. Brown, J. R. and R. H. Hoffman, "Evaluating the Effectiveness of Software Verification—Practical Experience with an Automated Tool," *AFIPS Conference Proceedings,* Vol. 41, 1972.
10. "Software Evaluation and Test System (SETS)," Logicon Report R:CDB-75085, 2 March 1976.
11. "Enroute Air Traffic Control Program, Software Optimization Study," FAA Report FAA-RD-75-87, June 1975.

NODAS—The network-oriented data acquisition system for the medical environment

by SHELLEY I. SAFFER, DAVID J. MISHELEVICH, SHIRLEY J. FOX and VICTOR B. SUMMEROUR

University of Texas Health Science Center at Dallas
Dallas, Texas

ABSTRACT

A network-oriented distributed computing system designed for the medical, multi-laboratory environment is described. The development of this network was motivated by the need for a real-time computing system which offers the speed and responsiveness of a dedicated processor and the convenience and cost-effectiveness of resource sharing. The system, a star configured network, utilizes a DECsystem-10 time-sharing system as its host and small memory-only PDP-11 processors as satellites. Program development for the satellite processors can be performed on the host in a higher-level language developed especially for the network. Also present are such network capabilities as down-line loading and remote file manipulation.

INTRODUCTION

The type of computing facility which best serves the real-time computing requirement of the multi-laboratory medical environment must be more flexible than either the smaller in-lab, dedicated computer system or the larger, multi-user, centralized system. The concept of a single centralized processor for multi-laboratory use offers the advantages of resource sharing but encounters problems with throughput and availability. In contrast, the concept of the dedicated "in-lab" computer system offers instant computational availability but may have the disadvantage of uneconomical utilization of resources. However, through a network configuration, combining both the centralized and dedicated processor concepts, a computing system with the advantages of both can be realized. Such a network is now being utilized by the Medical Computing Resources Center at the University of Texas Health Science Center at Dallas in order to augment existing minicomputer capability and in some cases obviate the need for laboratory investigators to obtain their own complete minicomputer systems.

THE MEDICAL LABORATORY ENVIRONMENT

In the medical laboratory environment, real-time applications usually involve relatively high speed acquisition of analog signals. In this context, the term "real-time" involves externally-controlled, time-constrained data which is not readily-reproducible, or involves direct control.[1] However, the varied nature of medical laboratory research presents a wide variety of real-time situations. Some laboratory applications, for example, may merely require the automated acquisition of digital data from a laboratory instrument at a reduced rate (perhaps as slow as one sample every 30 minutes). Laboratories with such low-speed requirements could very well be serviced by a centralized single CPU facility. On the other hand, a laboratory function may require the sampling of a number of analog signals every millisecond for a long period of time. Such rapid A/D conversion may dictate the exclusive use of a dedicated computer performing only one time-critical task.

COMPARISON OF IN-LAB AND CENTRALIZED SYSTEMS

The dedicated in-lab processor has the advantage of giving real-time capability at the instant it is needed. This is an important attribute since time-critical functions are not likely to wait for a multi-user computer system to become available. In many medical laboratories, real-time systems usually consist of small single-user processors dedicated to a particular task. These systems have proven to be a valuable tool and their success has increased the demand for more computing power in the laboratory. As a result, more and more is expected from these small computers which reveals an inherent weakness in the dedicated computer approach. Although a basic processor is relatively inexpensive when purchased, it soon becomes evident that extra memory, fast-access bulk storage, and higher-speed alphanumeric I/O devices are necessary to increase program development capability as well as system productivity. Soon the investment can become sizable and expensive equipment, maintenance, and extra personnel may drain the funds needed for the very research which the computer is supposed to facilitate.

This is not to say that such a progression is inevitable. It is, however, not infrequent. In many cases it is not difficult to reach the point of diminishing returns with such small computer systems. The higher-speed peripherals are necessary for adequate time-utilization and efficiencies. Many small laboratories may not be able to purchase the faster more expensive peripherals such as high-speed line-printers or adequately sized high-speed mass storage devices. Because these peripherals usually facilitate program development, there is the trade-off between increased cost for equipment and increased cost for application software.

On the other hand, a centralized facility has the advantage of resource sharing, thereby lessening the utilization cost of expensive sophisticated equipment to any one user. Qualified personnel can be employed by the central facility and thus be available to the individual laboratories for consultation and other services when needed. Also, maintenance of a central facility can exist within a unified framework under one vendor. However, because of the critical time-constraints which are placed on these applications, one centralized processor may not be able to adequately meet the demand of all real-time needs in a multi-laboratory environment. If many users are competing for high-speed data acquisition capability, some may not receive adequate response from the system. This is of course unacceptable in a real-time situation. A centralized facility could possibly be utilized in a real-time environment only on a demand basis in which a laboratory requests that the highest priority be granted to an upcoming application. However, in the medical environment, the variety of real-time tasks occurring at different times makes scheduling system resources impractical. With many laboratories on-line simultaneously, it is virtually impossible to schedule resources such that each user has access to a specific real-time capability when needed.

THE NETWORK APPROACH

A compromise between the "in-lab" and centralized concepts is the network approach which offers the combined advantages of both of the above alternatives; that is, the advantages of resource sharing of a centralized system and the speed and responsiveness of a dedicated processor. Through a network, or a distributed computing configuration, a low-cost, memory-only processor can be dedicated to the laboratory to perform critical real-time functions, and through its connection to a host, rely on the host processor's peripherals, bulk storage, high-speed line-printers, etc., when needed.

NODAS

NODAS is such a distributed computing system. It was designed to help meet the need for a low-cost and highly flexible real-time computing capability for the laboratory environment. It is configured as an "ICDS" (Indirect, Centralized routing, Dedicated path, Star) system[2] or "star" configuration. It utilizes as its host the time-sharing DECsystem-10 and various models of the DEC PDP-11 as remote satellites, see Figure 1. To the DECsystem-10 host, the remote processor appears to be just an intelligent interactive terminal. Thus as many remote processors can be supported as regular terminals. The remote processor resides in the laboratory and communicates with the host over a "hard-wired" 20-milliamp loop serial asynchronous line operating at 9600 baud. The usual distance from remote processor to host is from 700 to 1000 feet. Further distances can be achieved using special line-drivers.

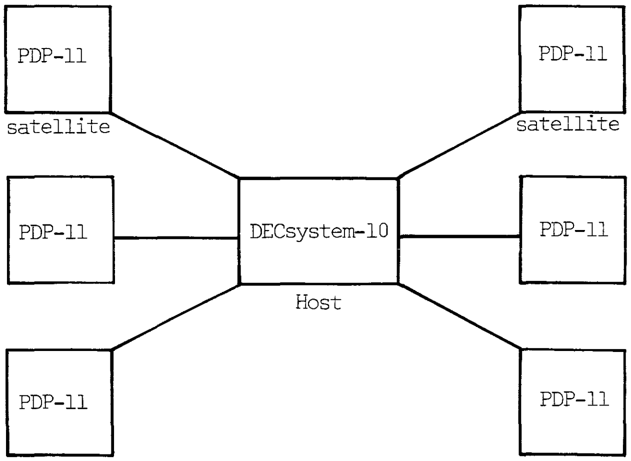


Figure 1—The NODAS distributed computing network; a star configured network with a DECsystem-10 as a host and PDP-11's as satellites

The system was designed to support a minimal amount of hardware dedicated to any specific laboratory. For example, the minimum configuration for any laboratory satellite is a PDP-11 (whether a PDP-11/45 or LSI-11) with 8K of memory, 2 asynchronous serial interfaces and an interactive console terminal. One asynchronous interface is used for the console terminal and the other is used for communication with the host. Of course for most laboratory work, an 8 to 16 channel A/D converter, a 4 channel D/A converter, and a graphics display device should be included. No mass-storage devices are required for the satellite processor because all program development and data storage occur on the host machine. Programs for the satellite are developed on the DECsystem-10 in a normal time-sharing manner using a cross-compiler, NODAL (the Network-Oriented Data Acquisition Language), that was written especially for the network system. Load modules are created on the host and sent down-line to be loaded into the remote satellite. The suggested minimal memory size of 8K words is adequate to support programs compiled with NODAL. The remote monitor takes about 500 Bytes and the NODAL run-time package about 3K words. This leaves about 4.5K words for a user program. The 8K configuration has proven to be adequate for a number of small data acquisition and analysis programs.

NODAS has two monitors, one which runs on the DECsystem-10 in a time-sharing manner, and the other which is resident within the satellite processor. The monitor in the remote processor allows two modes, Transparent Mode and Execution Mode. In Transparent Mode, the remote user has complete access to the time-sharing facilities of the DECsystem-10. In this mode, the remote processor merely acts as a buffer relaying bytes of information from the user to the DECsystem-10 monitor and vice versa. Through Transparent Mode, the user has access to the host to perform data analysis, program development, etc.

To achieve Execution Mode, the user runs a submonitor program on the host. Upon entry into that monitor, the name of a particular load module is entered by the user. That module is then sent down-line under a communications protocol and loaded into the remote processor. If the load is successful, execution of the program begins automatically. At this stage real-time activities can be performed. Data can be collected, averaged, displayed, and processed accordingly. Data, once collected, can also be sent back on a non real-time basis to the host for storage and further analysis. At any time during Execution Mode, the remote user can type "Control-C" signaling the resident monitor to return to Transparent Mode. It is interesting to note that to the inexperienced user in the laboratory, the operation of the DECsystem-10 host is more or less hidden. Thus it may appear that the small memory-only PDP-11 has all the power of a sophisticated general purpose data-processing machine.

As mentioned before, all communications between the host and satellite occur over an asynchronous serial line operating at 9600 baud or less. This baud rate is adequate for most laboratory applications because data transmission back to the host usually takes place in a non time-critical manner. By the time data is sent to the host for storage, the real-time function of collecting, averaging, displaying, and pre-processing has already been accomplished. In many laboratory applications, a great amount of data is collected at high rates and processed in the remote computer creating a final product of relatively few values. Thus data transmission back to the host usually consists of small irregular bursts of data rather than a large contiguous transmission. This is desirable since the transmission of very large data files may be time-consuming and thus interfere with the overall time-sharing response of the host.

Although the communication line operates at 9600 baud, the effective transmission rate (observed during operation of the system) is approximately half that or about 500 bytes/second. This is due in part to the communication protocol overhead, but mainly due to the varying response of the DECsystem-10 time-sharing host. The transmission speed is one limiting factor because some applications may require the rapid collection and immediate storage of data. For these applications, an inexpensive high-speed cassette tape or a small diskette system would be adequate. Actually, an inexpensive form of mass-storage is advisable for certain laboratory functions which must be performed regardless of whether or not the host machine is running. Load modules can be stored on a cassette or diskette and loaded from these devices if the host is not running. Thus a satellite could have enough mass-storage capability to run independently, but not enough to perform the non-time-critical tasks of program development and non-real-time data analysis. The addition of a small inexpensive storage device does not necessarily dilute the advantages of the network configuration. Such an addition could allow the remote processor to be an independent "stand-alone" system with its own operating system. However, the network configuration still has the advantage of allowing the user access to a large time-sharing system with adequate disk space, high-speed alphanumeric input or output devices, and in the case of the DECsystem-10, software utilities such as a powerful text editor, easy-to-use monitor commands, etc.

COMMUNICATIONS PROTOCOL

All communications between the host and remote processors occur within the framework of a communications protocol. The protocol is byte oriented and requires the establishment of synchronization between the host and satellite before the communication of each record begins. Each data record begins with a Start-of-Message control byte followed by a byte count and ends with an End-of-Message control byte followed by a check sum. Each transmission must be properly acknowledged or a request for re-send will be issued.

The protocol is not complex. It was designed for compactness and a reasonable degree of accuracy in a low-noise environment. The protocol establishes the remote processor in a "master" relationship with the host at all times. For example, it is the remote processor that issues synchronization characters when needed and requests that data be sent or that data be received. The host never initiates a request to send or receive data but merely acknowledges the remote processor's requests and acknowledges if a message has been properly sent or received. If an error is detected in a message, it is the satellite that requests to re-send or re-receive. Also, if severe "line noise" should result in the processors' losing synchronization, it is the remote processor's task to "time out" and reestablish synchronization. This master relationship of the remote processor simplifies the protocol and allows an adequate degree of control for network procedures.

This protocol has been used in a test environment for over a year and in two recent laboratory applications with good results. In a number of testing situations, severe noise was imposed on the serial communication lines with the system always recovering without loss of data. The check sum does allow two different one-bit errors in the same relative bit position to go undetected. This has not been a problem thus far. However, future upgraded versions may employ Cyclic Redundancy Checking, CRC-16,[3] which perhaps will enhance the error-detecting capability. It must be noted, however, that increased error-checking capability will result in increased overhead.

NODAL

One of the nice features of NODAS, from a user point of view, is that programs for the remote processor can be easily written in the higher level language NODAL. NODAL, the Network-Oriented Data Acquisition Language, is similar to FORTRAN and was written especially for the network system. A FORTRAN structure was chosen because it is easy to use and perhaps easily recognized by most who have had a little programming experience. This will perhaps encourage laboratory personnel to write their own programs.

NODAL is a cross-compiler (which is written in FORTRAN IV) that runs on the DECsystem-10 in a normal time-sharing manner. Thus programs for the laboratory can be developed outside the laboratory at any time. NODAL has full arithmetic capabilities and has many features of FORTRAN IV. There is a library of callable subroutines that facilitate the implementation of various laboratory and network features. For example, CALL OOPEN and CALL IOPEN which, when executed on the remote processor, direct the host to open a file for remote data storage or retrieval respectively. CALL SEND and CALL RECEIVE will send data from the satellite to the host for storage or direct the host to read a record from its disk and send the data to the remote processor. Thus it is relatively easy for the user to move data over the network.

Another interesting aspect of NODAL is the capability for one load module, executing in the remote processor, to initiate the loading of another load module. The CALL CHAIN command will direct the host to load a new program load module down-line and automatically execute it. Thus a sequence of programs can be executed in the remote processor. Data can be preserved in a "COMMON" area, thus allowing communication between these different load modules.

One of the most flexible features of NODAL is the ability to place PDP-11 assembly language instructions between the higher level language statements. Since the NODAL compiler generates assembly language source, the user written assembly language statement is merely included in the source allowing the user to have complete assembly language capability in manipulating variables defined and used at the higher language level. Some interesting combinations can thus be brought about. For example, assembly language manipulation can direct a vectored interrupt to a routine written entirely in the higher level language. Thus interrupt handlers, device drivers, and other interrupt functions can be written in the higher level language.

There are also a number of user system subroutines that will be added to the NODAL library to facilitate common laboratory computing functions. Such routines as SETCLK to set up a real-time programmable clock, SETBUFF to link a buffer with an A/D interrupt routine, and various display routines for the support of graphics devices will be available to the user in the NODAS Library.

CONFIGURATIONS

There are a number of interesting minicomputer networks which have been implemented.[4-7] As mentioned before, NODAS uses a "star" configuration. Such a "star" arrangement has the advantage of flexibility and modularity with respect to the remote processors. However, with respect to the host, the "star" arrangement is very inflexible when considering failure-effect and failure-reconfiguration.[8] Therefore, with emphasis on reliability, the NODAS user is given the option of one of two hosts, the DECsystem-10 or a PDP-11/45, which is connected to the DECsystem-10 via a DA-28 high speed interprocessor buffer as shown in Figure 2. The PDP-11/45, running the multitasking operating system RSX-11M, has two basic functions. The first is to act as a data concentrator for the remote processors thereby increasing throughput to the DECsystem-10 within the network. The second is to act as an independent host if the DECsystem-10 is not running. The PDP-11/45 can perform all of the host functions except program development.

In many respects, from the remote user's viewpoint, there is no difference in being directly connected to the DECsystem-10 or being directly connected to the PDP-11/45. The PDP-11/45 host can achieve Transparent Mode and allow remote users access to the DECsystem-10 timesharing monitor the same as if that user were directly connected to the DECsystem-10. However, one major difference does exist between the two hosts. When a remote processor is using the PDP-11/45 as a host, files opened for output by the remote processor will be written on the PDP-11/45's disk. Therefore the data file must be sent from the PDP-11/45 over to the proper user's area on the DECsystem-10 in order to be available to the user on a time-sharing basis.

EXAMPLE APPLICATION

One NODAS application involves the real-time collection of EKG data from the EKG Exercise Laboratory in the Division of Cardiology of The University of Texas Health Science Center at Dallas. The NODAS satellite processor utilized in this application is an 8K PDP-11/20 with 2 asynchronous serial interfaces, an 8 channel A/D converter and a 4 channel D/A. This remote processor samples a 3-lead EKG and Phonocardiogram every 5 milliseconds. A Trigger signal is also present (which detects the QRS

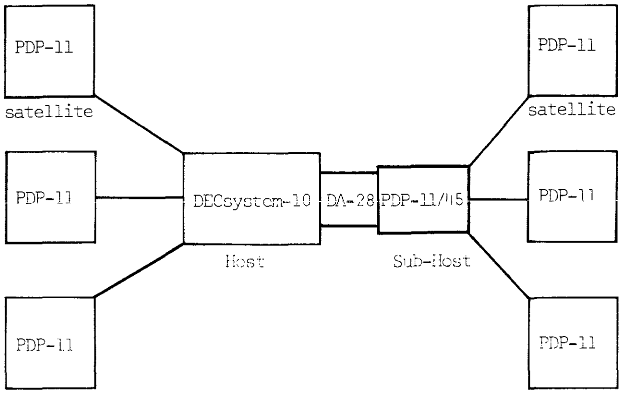


Figure 2—Alternative configuration for the NODAS Network with PDP-11/45 connected to the DECsystem-10 via DA-28 interprocessor buffer

complex). Instant graphics feedback is produced with a 4 channel D/A converter driving a Tektronix 611 Storage Scope. During the exercise test, data is sampled for a series of 25-second rest and exercise sessions using different exercise loads. As each waveform is collected, it is examined for proper length and compared, using a correlation routine, with the running averaged waveform. The current waveform is accepted only if the correlation factor is within specified limits. Thus, erroneous waveforms caused by transient noise can be excluded from the final averaged signal. After each 25-second exercise segment, the three EKG waveforms are displayed on the storage scope along with a numeric value representing the heart rate. If acceptable to the operator, the remote processor sends the averaged data to the host for storage and resumes to collect the next 25-second segment. This type of user interaction is typical in the medical laboratory environment and reinforces the need for a responsive computing facility. When the exercise session is finished, the data file is closed on the host and the satellite returns to Transparent Mode allowing the operator access to the DECsystem-10 time-sharing monitor. At this point, the technician can run "non-real-time" data analysis programs on the DECsystem-10 in a regular time-sharing fashion.

CONCLUSION

The NODAS system exemplifies the use of a distributed computing configuration in establishing a real-time, multi-laboratory computing facility for the medical environment. The system is relatively new and is being used at The University of Texas Health Science Center at Dallas. As the use continues, the system will constantly be evaluated and improved. A useful expansion would be the support of the more common micro-processors. The LSI-11 is already supported since its instruction set is equivalent to the PDP-11/40. However, the support for a variety of microprocessors would increase the utility and value of NODAS. Another area for improvement is in the computer-to-computer communications. Different communication protocols could be evaluated in the attempt to improve transmission efficiency.

As mentioned previously, the greatest disadvantage experienced thus far has been the network's sensitivity to hardware failures of the host. However, the addition of inexpensive cassette or diskette storage will help to lessen the dependence on the host during critical real-time activities.

The network approach is a valid method for implementing real-time computing capability in a multi-laboratory environment, especially an environment involved in medical research and clinical activities. The distributed-computing approach demonstrates the realization of a laboratory computing facility which offers the speed and responsiveness of a dedicated system while at the same time offering the advantages of resource sharing.

REFERENCES

1. Saffer, S. I. and D. J. Mishelevich, "A Definition of Real-Time Computing," *Comm. ACM (Forum)* 18,9 September 1975, pp. 544–555.
2. Anderson, G. A. and E. D. Jensen, "Computer Interconnection Structures: Taxonomy, Characteristics, and Examples," *ACM Comp. Surv.* 7,4, December 1975, pp. 197–213.
3. Boudeau, P. E. and R. F. Steen, "Cyclic Redundancy Checking by Program, *AFIPS Proceedings*, Vol. 39, 1971, pp. 9–15.
4. Ashenhurst, R. L. and R. H. Vonderohe, "A Hierarchical Network," *Datamation*, 21,2, February 1975, pp. 40-44.
5. Farber, D. J., "A Ring Network," *Datamation*, 21,2, February 1975, pp. 44-46.
6. Wulf, W. and R. Levin, "A Local Network," *Datamation*, 21,2, February 1975, pp. 47-50.
7. Fraser, A. G., "A Virtual Channel Network," *Datamation*, 21,2, February 1975, pp. 51-56.
8. Anderson, G. A. and E. D. Jensen, "Computer Interconnection Structures: Taxonomy, Characteristics, and Examples," *ACM Comp. Surv.* 7,4, December 1975, p. 206.

A system for priming a clinical knowledge base

by RANDAL L. WALSER and BRUCE H. McCORMICK
University of Illinois
Chicago, Illinois

ABSTRACT

Priming refers to the stocking of a knowledge base with inference rules derived from expert physicians. One approach to priming is demonstrated by KAMM, a program that accepts rules from an expert at a CRT, and integrates them into the systemic memory of a medical consultation system called MEDICO. Decomposition of inference rules, and storage in a relational database, provides for great flexibility. Easy reorganization of the knowledge base is facilitated by RAIN, a Relational Algebraic INterpreter which is available both to the designer and to KAMM. Verification of systems of inference rules, i.e., inference nets, is performed while the rules are still in decomposed form in the relational database. Systemic memory is generated subsequently by KAMM, from the verified network implicit in the relational database. A hashed index on an encyclopedia of propositions helps speed MEDICO's access to groups of inference rules during consultation sessions.

INTRODUCTION

Perspective

Recently, several large knowledge-based systems have been developed for providing nonspecialist physicians with advice about disease management.[11-13,17,22] These systems construct diagnoses and treatment plans by manipulating symbolic models of patients and situations, according to general rules acquired from expert physicians.

While knowledge-based systems hold great promise for improving the quality of clinical decision making in circumstances in which an expert physician is unavailable, their construction presents the builder of health care systems with an enormous task. It has been estimated that a clinically useful consultation system may need from 2,000 to 10,000 inference rules, each comparable to a statement in a high level programming language. Acquiring such large numbers of rules, and insuring that they fit together coherently and consistently, a process we call *knowledge priming*, is a major bottleneck in the development of practical systems. In the present paper we discuss our approach to knowledge priming, and describe a system for knowledge acquisition and maintenance, called KAMM.

Background

KAMM is being used to help develop MEDICO, a computer-based consultant for giving advice about the management of chorioretinal diseases in Ophthalmology. MEDICO is basically a production system,[5,18,19] patterned after that in References 11, 12, 17 and 9. Figure 1 shows MEDICO's three major divisions, one of which is KAMM (the Knowledge Acquisition and Maintenance Module). In addition, there is a rule interpreter, which applies rules at consultation time, and a knowledge base, which holds rules and models of situations. Short term memory is the "immediate" knowledge base, containing state description (propositional) models[7] of current clinical events. Long term memory has two major parts: episodic memory, with facts about individual patients and events encountered in the past; and systemic memory, with general clinical knowledge. Our present focus is systemic memory, and the manner in which it is generated by KAMM.

STORAGE OF AN INFERENCE NET IN A MINICOMPUTER SYSTEM

Inference rules, which are composed of propositions, often fit together logically to form an "inference net." In this section we consider the structure of inference nets, and illustrate how they may be organized in secondary storage, on disk, to promote efficient access.

Propositions

We represent propositions in the traditional way, as in the predicate calculus. For example, the English assertion "The exudate is yellow" is written

$$\text{color of (exudate, yellow).} \qquad (1)$$

For us, a proposition is an interpreted formula; that is, the

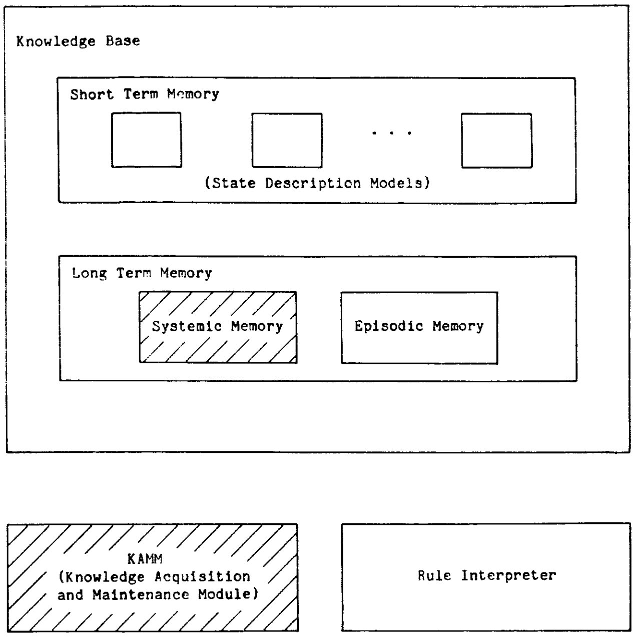


Figure 1—Overview of MEDICO. The concern of the present paper is the manner in which KAMM generates systemic memory.

predicate is the label of a relation $A_1 \times A_2 \times \ldots \times A_n$ on sets of knowledge elements. The argument list is an ordered n-tuple $(a_1, a_2, \ldots, a_n)$, such that $a_k \epsilon A_k$. We say the first argument is the topic of the proposition. For example, in (1), the predicate "color of" is a relation on {clinical objects}×{colors}. The arguments "exudate" and "yellow" are knowledge elements, instances of the categories "clinical objects" and "colors," respectively. The topic of (1) is "exudate."

Inference rules

Of the propositions comprising an inference rule, one is the *hypothesis*, while the others are the *supports* for the hypothesis. An inference rule is depicted graphically as in Figure 2a, in which the nodes are propositions. The hypothesis H is a possible explanation, or diagnosis, for the conditions modelled by the propositions $e_k \epsilon E$, the set of supports. This representation is similar to the one used in Reference 6, except that we use a *bar* to represent a rule, in an allusion to Petri nets.* An inference rule dictates that all conditions represented by the set E must be operative before the condition H may be inferred. The bar in Figure 2a indicates that, in reasoning, a transition to H is permissible only after the occurrence of the conjunction $e_1 \wedge e_2 \wedge \ldots \wedge e_n$. In the case in which a rule carries a single support, the bar may be omitted (Figure 2b).

An inference rule has an assigned strength (Σ in Figure 2), supplied by a knowledge domain expert. Strength is a subjective estimate of the extent to which the hypothesis is valid, given the set of supports as evidence. It is instructive to think of strength as the explanatory power of a hypothesis, given that the supports are valid. There is currently much controversy over means for using strengths, and transferring uncertainty from supports to hypotheses. We skirt the issue here, since our emphasis is on overall organization of systemic memory, and not on details of its use. However, the methods in Reference 6 are noteworthy.

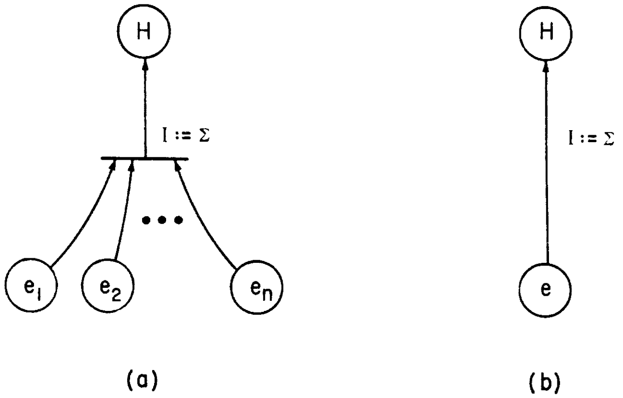


Figure 2—(a) Graphical representation of an inference rule I with assigned strength Σ. The bar indicates that the proposition H, the hypothesis of the rule, is triggered by a conjunction of propositions, the $e_k<E$, the set of supports. The rule dictates that all conditions represented by the set E must be operative before the condition H may be inferred. (b) The bar may be omitted for a rule having a single support.

Inference nets, and their disassembly

Sometimes a proposition occurring as a hypothesis in one rule appears as a support in another rule. A system of rules effectively meshed together in this way, by common propositions, is called an *inference net*.[14] An example, with three hypotheses, is shown in Figure 3. The propositions H_9 and H_5 are hypotheses that do not support any other hypotheses. The proposition e_2, on the other hand, is a hypothesis in rule 70, a support in rule 36.

The evaluation of a hypothesis during a consultation session may require that many, perhaps all, supporting propositions be evaluated. In minicomputer implementations* this means that all rules relevant to the evaluation of a particular hypothesis must be brought from disk into core storage. If rules with a common hypothesis are stored in physically separate locations, substantial time may be consumed in positioning the disk access mechanism before hypothesis evaluation can get under way. In MEDICO, in order to reduce disk access time, rules in systemic memory are organized around common hypotheses, and are stored in contiguous locations on disk. Then, when a hypothesis is evaluated, minimal searching is required to bring all rele-

* Petri nets have been useful for modelling "patterns of activity" in concurrent systems.[10] It may be worth pursuing the possible uses of Petri nets in the present context. Hack[8] has provided formal footing, and Murata[13] has offered some computer implementable methods.

* MEDICO is implemented on a DEC PDP-11/40 computer with 160k bytes of core storage. Programming is in the C language, supported by Bell Telephone Laboratories UNIX operating system.

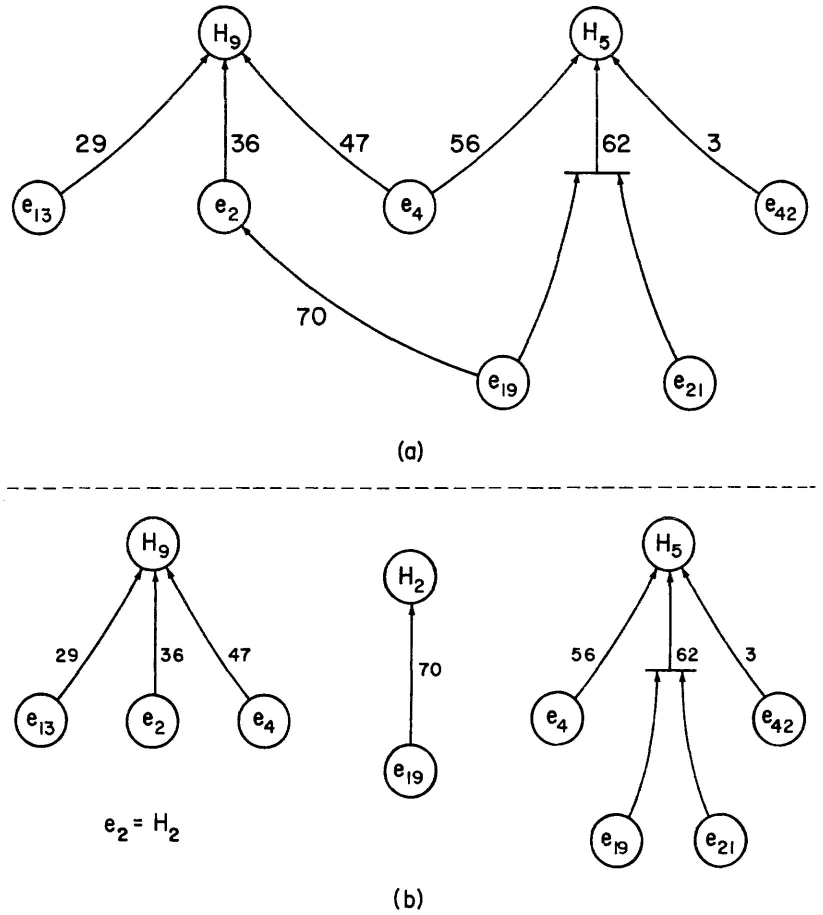


Figure 3—(a) Example inference net with rules labelled with integers (rule strengths not indicated). Notice that e_2 is the single support in rule 36, but the hypothesis in rule 70. (b) Disassembly of the inference net in Fig. 3a into groups oriented around hypotheses.

vant rules into core. Figure 3b is an example, showing the appropriate reorganization of the net in Figure 3a.

USE OF A RELATIONAL DATABASE

Rules are acquired during priming sessions involving an expert physician, who interacts with KAMM at a CRT. The input acceptable to KAMM may look very rule-like, or, for convenience, may be formatted in a style that accents the embeddings natural in English assertions.*

It is important to notice that rules input by an expert do not find their way directly to systemic memory. Rather, incoming rules are parsed by KAMM and are placed in decomposed form (i.e., 3rd normal form[4]) in a relational database. Later, after maintenance operations directed at insuring the integrity and consistency of the inference net, KAMM generates systemic memory automatically from the decomposed rules.

KAMM does this through an interface to a Relational Algebraic INterpreter (RAIN).[2] The chief advantage of this approach is its flexibility. Systemic memory is subject to continual revision, especially during the initial priming period, which may go on for several years. Using RAIN, it is easy to examine, update, and reorganize groups of rules, all with little reprogramming effort.

Structure of the relational database

The database has four major relational files. Each is defined in RAIN with a statement of the form "define efile R(d_1, d_2, . . . , d_n)," where "efile" specifies that the relation is elementary,[2] R is an arbitrary relation name, and the d_k are the names of the relation's domains (i.e., attributes, or descriptors). In particular, the four pertinent files are defined as follows:

define efile supports (prop#, rule#)
define efile hypotheses (rule#, prop#)
define efile propositions (predicate, argl, argk, arg code)
define efile likelihoods (rule#, reverse, forward)

The structure of the database is depicted in the information graph[3] in Figure 4. The dashed, labelled ovals repre-

* For a more detailed discussion see Reference 21; the input syntax is listed explicitly in Reference 20.

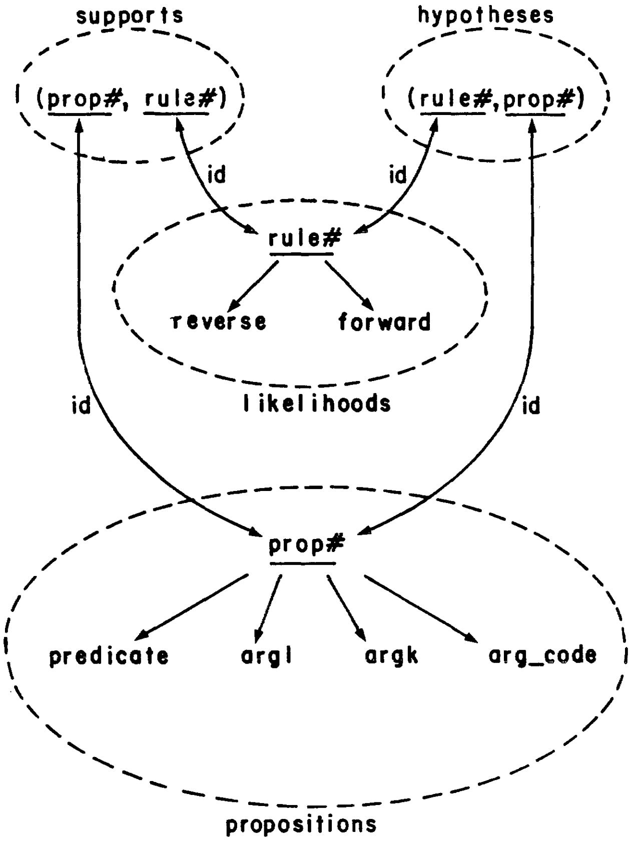


Figure 4—Information graph [3] showing the organization of the relational database in which MEDICO's inference net is stored.

sent 3rd normal form[4,6] relational files, with the files' attributes written inside the ovals. The unlabelled unidirectional arrows within the "likelihoods" and "propositions" files indicate functional dependence. For example, the arrow from "prop#" to "predicate" in the "propositions" file indicates that "predicate" is functionally dependent on "prop#." This means simply that if we know the number of a proposition (which is assigned by KAMM, and is unique), then we know the predicate of the proposition. The attributes of a relational file in 3rd normal form are functionally dependent on the *key* of the file (shown underlined in Figure 4), which may be an attribute, or a set of attributes. The arrows labelled "id" link attributes which are identical across files (these are not defined in the database, but are part of its semantics).

Following the id links and using relational algebra, it is possible to move logically between files, creating new relations, and grouping the components of rules in virtually any way desired. RAIN's "sort" operation is especially useful for perusing the contents of the database. It is an easy matter, for example, to get an alphabetical list of all topics; one simply does a "projection" on the attribute "argl" in the "propositions" file, then "sort"'s the resulting relation (which is effectively a list). One especially important set of lists indicates which rules have common hypotheses. Such a list, for an hypothesis labelled (numbered) H, is generated by doing a "restriction" on the file "hypotheses" such that "prop#"=H.

The reason for distinguishing forward and reverse likelihoods in the "likelihoods" file is that corresponding to every rule of the form A→B with likelihood l_{ab}, there is a dual B→A, with likelihood l_{ba}. The A−B association constitutes two rules because, in general, $l_{ab} \neq l_{ba}$. Rather than enter the dual of every rule explicitly in the database, it is vastly more economical to note, for each rule, that it has likelihood l while its dual has likelihood l*. The *forward* attribute in the "likelihoods" file refers to the likelihood of the hypothesis of the rule R which is explicitly stored in the database; the *reverse* attribute applies to the hypothesis of R's dual (which is implicit).

All propositions, of any degree (any number of arguments), are stored in the "propositions" file. Inspection of Figure 4 reveals that the "propositions" file provides for only two arguments in each tuple (of the file). This is appropriate for most propositions in practice, which are dyadic. The "arg_code" attribute is provided in order to accommodate propositions of arbitrary degree: propositions of degree greater than two "spill over" into additional tuples. Argument order is signalled by the value of "arg_code." If "arg_code"=2, then "argk" is the value of the second argument; if "arg_code"=3, then "argk" is the value of the third argument; and so on. The values of "predicate" and "argl" are identical in all tuples having the same "prop#." For example, two tuples are needed to store the triadic proposition "between (neck,head, shoulders)." If this proposition has the label, say, 38, the two tuples in which it is stored in "props" would appear as follows:

(38,between,neck,head,2)
(38,between,neck,shoulders,3)

Notice that nothing in either tuple, individually, indicates that "between" is a trinary relation. This information is uncovered automatically when RAIN is used to carry out a "restriction" on "props" such that "prop#"=38. The resulting relation contains the two tuples carrying the three arguments associated with the relation "between."

As a simple example of a rule's representation in the relational database, consider the following rule (in which the strength code 6 has the meaning "often"):

identify of(mass,melanoma):^{6}:
site of(mass,choroid) &
height of(mass,very high) &
shape of(mass,mushroom-like)

An interpretation is "A mass is often a melanoma, if it occurs in the choroid, is very high, and looks like a mushroom." Suppose the rule is labelled arbitrarily with the number 98; the hypothesis—"identity of(mass, melanoma)"—is labelled 22; and the supports are labelled 30, 31, and 32. Then the tuple (98,22) would appear in the "hypotheses" file, and the three tuples (30,98), (31,98), and (32,98) would appear in the "supports" file. The predicates and arguments of the four propositions involved in the rule would appear explicitly in four corresponding tuples in the "propositions" file. Finally, the strength of the rule would appear as the value of the "forward" attribute in a tuple in the "likelihoods" file; if the strength of rule 98's dual is defined, it would appear as the "reverse" attribute in the same tuple.

Parsing and bundling of incoming rules

The CRT screen used during knowledge priming is divided by a horizontal line into two sections, one for the expert, the other for KAMM. These sections may be thought of as "windows" through which the expert and KAMM communicate. The windows are dynamic, expanding and contracting with a bias toward providing the expert with as much space to type as possible. The top window is KAMM's, and is typically small, taking up just one or two lines, or just enough to accommodate the brief messages which KAMM puts out to the expert. The expert types rules in the bottom window, using an editor, incorporated in KAMM, which is very much like the RAND "windowing editor."[16] During these sessions the CRT is set to "raw" input mode, meaning that characters are transmitted to the computer—and to KAMM—as they are typed. The expert may type anywhere in his window, with KAMM "watching."* When the expert finishes typing the rule, he submits it to KAMM by signalling with a control key.

KAMM parses the input rule and, if it is syntactically well-formed "bundles" its components into RAIN insertion commands. It is possible for KAMM to initiate these commands directly, through a C language subroutine that provides an interface to RAIN.[1] Currently, KAMM does not communicate with RAIN during an acquisition session because the RAIN interface, while effective, is presently too slow for interactive use. Instead, the RAIN command bundles are written out to a temporary file on disk, where they remain until the acquisition session is over. Then a KAMM subprogram called the "inserter" passes the commands one by one to RAIN.

Integrity

The present inserter is the precursor of a potentially large program that will operate on incoming rules, and the relational database at large, maintaining data integrity and checking for inconsistencies. The relational database is a "relational network" of sorts, in which MEDICO's inference net is implicit. We consider it to be a natural structure on which to perform consistency checking. Using RAIN operations, for instance, we can follow inference chains through the network, and, as logical anomalies are uncovered, evoke appropriate modifications (which may involve, for example, setting flags, removing inconsistent rules to temporary files for future inspection by an expert, or striking rules from the database entirely). Using the C/RAIN interface, we plan to write programs to execute many of these tasks with minimum user guidance. Such programs will be the focus of much future attention in KAMM's development.

In the current implementation, the inserter performs a simple maintenance service by checking to insure that propositions and rules are not already stored in the relational database. Providing they are not, the inserter assigns them a unique identifying number (the information graph in Figure 4 illustrates the link-making role of these numbers). It is extremely important that these numbers be associated uniquely with the appropriate data objects; otherwise the functional dependencies around which the relational database is organized will deteriorate, and the means for recomposing rules will be useless.

If the inserter finds that an incoming data object is already stored, it of course does not insert the object. If the object is part of a larger, unstored data structure, then the object's previously assigned number is used to represent the object in that structure. Suppose, for example, that an incoming rule (which the inserter has confirmed is new) has two supporting propositions A and B. Suppose that the proposition A is already stored, and was previously assigned the number, say, 108. Suppose further that the proposition B is indeed new, and that the inserter will assign it a unique number, say, 942. The inserter accordingly inserts proposition 942, but not 108, and then enters two tuples into the "supports" file, defined over {rule#}×{prop#}. If the new rule is assigned the number, say, 416, the two new "supports" tuples will be (416,108) and (416,942).

SYSTEMIC MEMORY

Task modules

Earlier, we said that inference rules have the form E→H, where E is a set of propositions that support the hypothesis H. More generally, inference rules are production rules, having the form S→A, where S is a situation and A is an action. Consider that an inference is "made" when something is done. In MEDICO, an inference is made when an hypothesis (a proposition) is confirmed (by some procedure) and the confirmed proposition is added to a list of propositions in a state model (of a clinical situation). The consequence of a rule in MEDICO is invariably an action, perhaps an internal operation like the making of an inference, or perhaps something more public, like the making of a remark, via a CRT. In general, a production rule means "when this situation is recognized, do such and such." By grouping rules around common hypotheses, i.e., common actions, we assemble, for easy reference, all situations

* In a proposed future improvement, KAMM will do more than just watch: as the expert types, the multiprocessing capability supported by the Unix operating system and the C language, in which MEDICO is implemented, will allow KAMM to refer to systemic memory during lulls in typing. If incoming words or propositions are unknown, or present consistency problems, KAMM will interrupt the expert to pose questions, or else prepare questions to be asked when the expert indicates he is ready for them.

which mean "perform this task." We call such groups *task modules*.

The encyclopedia

The encyclopedia is a large file in secondary storage, containing all propositions known to MEDICO. We consider a proposition's topic to be its first argument. "Site of(hemorrhage,retina)" is a proposition on the topic "hemorrhage," for instance. An entry in the encyclopedia consists basically of a topic, followed by a list of propositions on that topic. In the encyclopedia under the topic "hemorrhage" we might find the propositions "site of(hemorrhage, retina)," "color of(hemorrhage,dark red)," and "composed of(hemorrhage,blood)," among many others.

Recall that a proposition by our definition is incomplete without an accompanying specification of context. The required specification is provided in the encyclopedia by a pointer, called a *context pointer* from each proposition P to the task module whose task is: "infer proposition P." The sets of supports in this task module constitute models for the contexts entailed by P (or which are merely associated with P). Also accompanying each proposition in the encyclopedia is a list pointing to all task modules in which the proposition appears as a support. Besides indicating what contexts a proposition may be part of, this list, called the *trigger list*, tells which hypotheses the proposition triggers.

CONSTRUCTION OF SYSTEMIC MEMORY

As indicated earlier, systemic memory is not assembled directly after input of rules by an expert. Rules are stored first in decomposed form in a relational database. Our plan is for KAMM to put systemic memory together automatically, using RAIN through the C/RAIN interface. Ideally, this should occur only after the inference net which is implicit in the relational database has been determined to be free of inconsistencies. Currently, the mechanics of the assembly process have been worked out and several relevant KAMM subprograms are in working order. Sophisticated verification procedures, however, are not yet available.

Assembly of systemic memory begins with generation of the task modules. After producing a checklist for all rules by doing a "projection" over "rule#" in the "hypotheses" file, KAMM uses "restriction's on the hypotheses" file to find groups of rules with common hypotheses. As each group is found, it is formatted and appended to a disk file that is available to MEDICO during consultation sessions. The context pointers and trigger lists are also determined at this time, and stored in temporary relational files.* Then the encyclopedia is produced. First a list of all topics is generated, and then all propositions with the same topic are grouped in one location in the encyclopedia disk file. The context pointer and trigger list of each proposition is inserted as the proposition is laid down in its (encyclopedia) location on disk.

ACCESS OF SYSTEMIC MEMORY AT CONSULTATION TIME

During consultation, the encyclopedia serves as an index that helps speed access to task modules. When a user inputs a proposition, MEDICO's parser extracts the proposition's topic, and then looks it up in the encyclopedia. If MEDICO cannot find the topic in the encyclopedia, or is unable to infer its meaning, it asks the user to rephrase the input. If, however, MEDICO does find the topic in the encyclopedia, it then searches the associated list of propositions for the incoming proposition. Again, if it cannot find the proposition, MEDICO attempts to deduce the proposition's meaning, and if it cannot, asks for a rephrasing. If the incoming proposition is in fact in the encyclopedia, the proposition's trigger list is then used to access all relevant task modules.

The proposition recognition process is aided by the use of a hashed encyclopedia index, which is generated by KAMM upon construction of the encyclopedia (Figure 5). We will not discuss our hashing methods here, since details are available in a technical report.[20] The upshot, however, is that hashing enables MEDICO to recognize the topics of incoming propositions virtually immediately, regardless of the size of systemic memory. It takes one seek (disk access) to determine if an incoming topic is known, and

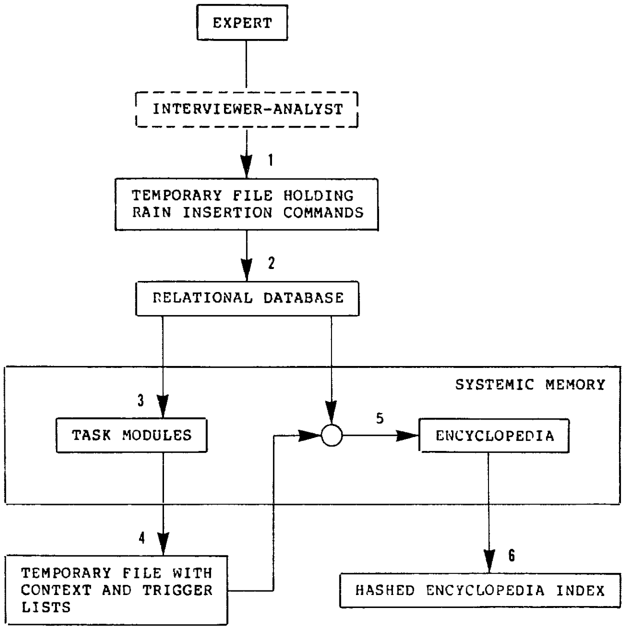

EXPERT
INTERVIEWER-ANALYST
1
TEMPORARY FILE HOLDING RAIN INSERTION COMMANDS
2
RELATIONAL DATABASE
SYSTEMIC MEMORY
3
TASK MODULES
5
ENCYCLOPEDIA
4
TEMPORARY FILE WITH CONTEXT AND TRIGGER LISTS
6
HASHED ENCYCLOPEDIA INDEX

Figure 5—Overview of major steps in the generation of systemic memory by KAMM. An expert may interact with KAMM directly at a CRT, or may communicate knowledge to an interviewer-analyst, who relays the knowledge to KAMM. The numbers on the arrows indicate the order in which the various structures are generated.

* Context pointers and members of the trigger list are actually file addresses relative to the base of the task modules file.

then from three to four seeks to locate any associated task module. We believe this provides for reasonably efficient and sensible "movement" through an inference net, considering that each task module may hold a large number of inference rules, all pertinent to an immediate purpose.

DISCUSSION

Reliable methods for acquiring and verifying large bodies of computer-resident knowledge are not available at present. Yet priming a knowledge base with accurate, relevant knowledge is essential to the success of practical systems for giving clinical advice. The knowledge acquisition and maintenance module (KAMM) described briefly in this paper is an attempt to deal with the priming bottleneck. Figure 5 provides an overview of the major operations performed by KAMM during the priming of MEDICO's systemic memory. We consider the storage of a knowledge base in decomposed form, in a separate relational database, to be an important step. Using relational algebra as a basic tool, the designer can rearrange the elements of a knowledge base until he achieves an organization that suits his purposes.

Our goal is to elaborate KAMM into a collection of design tools, useful generally for constructing, verifying, and validating knowledge structures. In the present paper we described KAMM's role in generating a particular knowledge structure, suitable to our application. While KAMM does not yet entail a general methodology for knowledge engineering, it does provide a useful basis for continued development. Plans for future work include (1) development of means for generating alternative knowledge structures from a relational database, and (2) greater emphasis on techniques for verification and validation of inference nets. Chorioretinal diseases will continue to serve as the example knowledge domain.

ACKNOWLEDGMENTS

KAMM has benefited from contributions by W. H. Cheng, M. T. O'Brien, A. C. Petersen, S. K. Chang, and M. H. Goldbaum.

REFERENCES

1. Chang, S. K., "RAIN Manual," MISL Report M.D.C. 1.3.4, Department of Information Engineering, University of Illinois at Chicago Circle, 1976.
2. Chang, S. K., M. O'Brien, J. Read, R. Borovec, W. H. Cheng, and J. S. Ke, "Design Considerations of a Database System in a Clinical Network Environment," *Proc. National Computer Conference, New York*, 1976, pp. 277-286.
3. Chang, S. K. and W. H. Cheng, "A Database Skeleton and its Application to Logical Database Synthesis," MISL Report M.D.C. 1.1.17, Department of Information Engineering, University of Illinois at Chicago Circle, 1976.
4. Codd, E. F., "Recent Investigations in Relational Data Base Systems," *ACM-SIGMOD Workshop on Management of Data,* San Jose, California, 1975.
5. Davis, R. and J. King, "An Overview of Production Systems," Memo AIM-271, Standford Artificial Intelligence Laboratory, 1975.
6. Duda, R. O., P. E. Hart, and N. J. Nilsson, "Subjective Bayesian Methods for Rule-Based Inference Systems," Tech. Note 124, Artificial Intelligence Center, Stanford Research Institute, 1976.
7. Fikes, R. E., "Deductive Retrieval Mechanisms for State Description Models," *Proc. Fourth Int. Joint Conf. on Artificial Intelligence,* 1975, pp. 99-106.
8. Hack, M., "Petri Net Languages," Computation Structures Group Memo 124, Project MAC, Massachusetts Institute of Technology, 1975.
9. Hart, P. E., "Progress on a Computer Based Consultant," *Proc. Fourth Int. Joint Conf. on Artificial Intelligence,* 1975, pp. 831-841.
10. Holt, A. W., *Final Report of the Information System Theory Project,* Technical Report RADC-TR-68-305, Rome Air Development Center, Griffiss Air Force Base, New York, 1968.
11. Michalski, R. S., "On the Selection of Representative Samples from Large Relational Tables for Inductive Inference," MISL Report M.D.C. 1.1.9, Department of Information Engineering, University of Illinois at Chicago Circle, 1975.
12. Michalski, R. S., "Problems of Designing an Inferential Medical Consulting System," Department of Computer Science, University of Illinois at Urbana-Champaign, 1974.
13. Murata, T., "State Equation, Controllability, and Maximal Matchings of Petri Nets (revised)," MISL Report M.D.C. 1.1.10, Department of Information Engineering, University of Illinois at Chicago Circle, Submitted for publication, 1976.
14. Nilsson, N. J., "Some Examples of AI Mechanisms for Goal Seeking, Planning, and Reasoning," Tech. Note 130, Artificial Intelligence Center, Stanford Research Institute, 1976.
15. Pople, H. E. Jr., J. D. Myers, and R. A. Miller, "DIALOG: A Model of Diagnostic Logic for Internal Medicine," *Proc. Fourth Int. Joint Conf. on Artificial Intelligence,* 1975, pp. 848-855.
16. RAND Corp., "Manual for the RAND Editor," RAND Corp., Santa Monica, California, 1975.
17. Shortliffe, E. H., "MYCIN: A Rule-Based Computer Program for Advising Physicians Regarding Antimicrobial Therapy Selection," MEMO AIM-251, Stanford Artificial Intelligence Laboratory, 1974.
18. Vere, S. A., "Relational Production Systems," to appear in *Artificial Intelligence*, 1976.
19. Vere, S. A., "Composition of Relational Productions for Plans and Programs," Department of Information Engineering, University of Illinois at Chicago Circle, Submitted for publication, 1976.
20. Walser, R. L., "Technical Notes on the Organization and Access of Systemic Memory in MEDICO," MISL report M.D.C. 1.2.3., Department of Information Engineering, University of Illinois at Chicago Circle, 1976.
21. Walser, R. L. and B. H. McCormick, "Organization of Clinical Knowledge in MEDICO," to appear in *Proc. Third Illinois Conference on Medical Information Systems,* November 4-6, 1976.
22. Weiss, S. M., "A System for Model-Based Computer-Aided Diagnosis and Therapy," Ph.D. Thesis, Department of Computer Sciences, Rutgers University, 1974.

A proposed study to access the impact of microprocessors on health care delivery

by WILLIAM HYMAN and WILLIAM LIVELY

Texas A&M University
College Station, Texas

ABSTRACT

Rapid development in microprocessor technology points the way to significant impacts on a number of industries. One such industry is that of health care delivery in which the micro-miniaturization and almost zero cost of the microprocessor itself will lead to increasingly widespread applications.

The assessment of the impact of this new technology is of paramount importance for component and medical device manufacturers in delineating new product directions and for the prioritization of research funding.

It is necessary to make a thorough survey of current and projected microprocessor devices and applications through literature searches, research and manufacturer surveys, a special conference and a workshop. The results of these efforts would be incorporated into a report containing the background information and recommendations for further research and development. Failure to collect and disseminate technological information of this type as widely as possible retards the ultimate technological advances that may be derived. Enlightened technological application forecasting and direction emphasis become a key to effective and efficient future developments in health care delivery.

INTRODUCTION

The development of microprocessors has stimulated a number of new industries and produced significant impact on various old ones. It can be expected that these impacts will accelerate for a number of years to come due to increasing sophistication of available components and rapidly decreasing prices. One such industry is that of health care delivery in which the microprocessor will be significant for widespread technological innovation and reduced costs.

In order for developments in this area to be efficiently channeled, a thorough investigation of present and expected research and implementation of microprocessor usage within the health care delivery field is necessary.

The results of this study would be useful in prioritizing future government research support and of paramount importance in the coordination of interests among health care professionals, medical device manufacturers, and the computer component industry. It could also serve as the background of a technology assessment study of this area, which is a natural follow-on project.

STATUS OF MICROPROCESSOR TECHNOLOGY

In 1963 Digital Equipment Corp (DEC) produced the first minicomputer. This was a move to bring computing to a larger audience of users who could not afford the large computing machines available before 1963. With the advent of the microprocessor in 1969 by INTEL (4004), computing has been potentially brought to an even larger audience of users. LSI (Large Scale Integration) techniques have culminated in the development of a postage stamp size computer processor.[1]

The reduction in size and cost of computing hardware has resulted due to constant efforts of the semiconductor industry to increase the number of functions per semiconductor device, and this figure is doubling every year. The term Medium Scale Integration (MSI) refers to chips containing 100-1000 gates while over 1000 gates/chip is generally referred to as Large Scale Integration (LSI). Interestingly, the cost of producing a chip is a function of its size and not the amount of logic on the chip. Thus, increased integration results in lower cost per function. An accompanying advantage is increased reliability through fewer inter-device connections.[2]

The currently available microprocessors can be grouped into three classes based on functional partitioning: (1) the multi-chip family, consisting of compatible CPU, memory, and I/O devices; (2) the single-chip CPU, designed for standard product memory and I/O devices; and (3) the multi-chip, microprogrammable CPU, designed for standard product memory and I/O devices. Types (1) and (2) can be classified as "conventional" types of microprocessors with their architectures closely paralleling "conventional" minicomputers and consequently falling into this application area. Type (3) is classified as a bit-slice type of architecture which frequently comes packaged in 4 bit-slices. Bit-slice architectures become a unique entity with microprocessors and offer the greatest potential for flexibility and innovative processor designs.[3]

Past experimentation with parallel processors such as pipeline, array, and multiple processor systems has clearly demonstrated that performance is certainly not a linear function of the number of processors. In fact, a point of diminishing returns is very rapidly reached after which the addition of a processor no longer results in any significant performance improvements. Under these circumstances, it is the incremental cost/performance ratio which becomes important. The low (and rapidly dropping) cost of microprocessors should yield a significant reduction in this incremental cost/performance ratio. It is now possible to talk of systems consisting of hundreds of processors even if the overall utilization is low. Microprocessors lend themselves well to the design of modular and flexible systems. Again, the inherently high reliability of LSI components is another factor favoring the use of microprocessors.

Distinguishing limits of the microprocessor, as compared with those of the minicomputer, are shorter word lengths (typically 4 and 8 bits), usually slower speed, limited addressing modes, fewer internal registers, and less sophisticated interrupt capabilities. Programming is more difficult for microprocessors, since manufacturers' software, particularly in high level languages, is very limited, but trends indicate that the burgeoning microprocessor industry will overcome these limitations within a few years.[4]

The microprocessor is currently being used in two broad classes of applications.[5-7] The microprocessor provides a low cost, very dense hardware logic substitute where the complexity of the application makes the design of traditional discrete logic systems difficult. On the other hand, the more powerful microprocessors can serve as self-contained computers that perform tasks commonly assigned to minicomputers. The future will bring arrays of microprocessors that will compete with large scale computing machines.

Mass production and technological advances have decreased the cost of microprocessors from about $250 for an INTEL 8008 in 1969 to about $25 today. Within a few years these processors will have almost a zero cost. The cost of computing then reverts to the devices for inputting and outputting information to the processor and the accompanying application development.

The aspects of an almost zero cost for the microprocessor and its miniaturized size create tremendous potentials for widespread application and use. Any application that requires control or computational capabilities such as manufacturing, automobile dynamics, medicine, patient rehabilitation, prosthetic devices or telephone systems is a potential candidate for microprocessors.[8]

POTENTIAL HEALTH CARE APPLICATIONS

One industry which can be expected to be among those which are greatly affected by microprocessor technology is the multifaceted "health care" industry. This industry is not limited simply to hospital services. It includes planning and delivery of all facets of health care including both large and small hospitals as well as clinic services, group medical practices, individual medical practices, and non-physician services such as rehabilitation or pharmacy. The industry also extends to equipment and devices owned and used by patients themselves and to the selection, monitoring and maintenance of such equipment. The recent offering of short courses on microprocessor design in medicine and biology[9] as well as computer applications in general[10] are evidence of the continued growth in these areas. Increased use of computer systems in the health care delivery arena will require the employment of technologically trained people to purchase, use, and maintain microprocessor based equipment. Recent federal legislation concerning safety and efficacy of medical devices will of necessity include consideration of microprocessor components.

We can hypothesize on the application areas in health care and medical science for this technology. These areas can be divided into two groups. The first is one in which presently available digital computer capability could be utilized, but such use is limited primarily due to hardware and software interface problems and cost factors. A second group would consist of those application areas in which individuality, size, and mobility considerations are important as well as costs of implementation.

The first group includes any form of information processing such as basic hospital data, including both administrative (e.g., patient records, billing, scheduling) and clinical (e.g., patient charts) information. Computer based laboratory equipment and the analysis of laboratory results also fall into this group. Applications of this type include both automation of techniques for speed and accuracy as well as methodologies which are uniquely suited to computer capabilities such as complex diagnostic services and forms of pattern recognition. These application areas generally must rely on access to a large computer facility either through direct access or telephone hook-ups. This approach provides substantial computer capability but carries the problems associated with being dependent on an external processing system. The literature on large scale computer use is extensive. Such systems also can bind the user to fixed information formats and lack of instant access. Until recently, dedicated minicomputers were the only alternative to direct tie-in with a large central computer. These systems provide individual control of computing style and capability but carry a sometimes prohibitive price tag. The microcomputer will serve to provide dedicated capability in these areas at reasonable cost. Reports of such applications have begun to appear in the literature.[11,12]

The second group is perhaps the more exciting one in that it extends computer technology into new problem and application areas. An extension of information processing and clinical analysis capabilities to small group or individual medical practitioners offers the possibility of significantly improving the abilities of this segment of the health care industry. Such items as diagnostic screening, EEG and EKG analysis, drug interaction screens and administrative problems could be implemented using on site, individually tailored programs and equipment. This would eliminate problems of access and cost involved in using central computer capability for this type of service. It would also eliminate any psychological disadvantage associated with the use of "remote" programs and services.

The ability to stay "on-line" through the use of dedicated microprocessing suggests the further extension of the development of "smart" machines in the laboratory and in the realm of medical devices.[13] Many sophisticated clinical devices such as heart-lung machines, intensive and critical care instrumentation, dialysis machines, respirators, intra-aortic balloon and similar equipment could be significantly improved by automated monitoring and warning systems and/or direct control via microprocessing.

Extending our thinking somewhat further, it is likely that artificial totally internal prosthetic organs, such as an artificial heart or kidney, could also be monitored, controlled and functionally tested with internal microprocessing capability. Monitoring or control subsequent to non-prosthetic surgical intervention such as in vascular or orthopedic surgery can also be envisioned. This would provide post-surgical information on the efficacy of a procedure and its progress. This type of information is in general very difficult to obtain at the present time. It is for these patient carried systems that ultra-small size and reasonable cost substantially separates these future applications from large scale computer capability.

All of the application areas previously discussed are within the expected growth of microcomputer technology and are in no way science fiction. It would be of great value to expand on and amplify these application areas through the proposed study. This information should include assessment of what is currently available, in what areas is research currently under way, what is the source of the financial support, and what is the time frame for widely available implementation. Finally, an educated guess on what the next group of applications in the future is likely to be with projected costs is necessary. This information would form the background for a technology assessment study in which the interplay of needs, resources and related factors with purely technological capabilities would be determined. The information sought here would also serve as a guideline for future funding expectations and perhaps as a stimulus for accelerating new and worthwhile developments in microprocessor design, control, and application.

METHODOLOGY

An appropriate methodology for this investigation would consist of several phases.

The first phase would be a routine literature review and the subsequent generation of a comprehensive annotated bibliography on (1) microprocessor devices that are available and (2) medically related uses of microprocessors. Both manual and computer based searching methods for this material would be undertaken and an initial report could be generated within three months. This would of course be followed by periodic updates in order to keep abreast of the very latest literature. This document would serve as a reference work of progress to date in implementing this technology.

A second phase would consist of a survey of manufacturers and distributors of microprocessors and medical device manufacturers in order to ascertain the current efforts relating to microprocessor utilization in the commercial aspects of health care delivery. Major manufacturers of microprocessors such as INTEL, MOTOROLA, FAIRCHILD, etc., have application groups who could participate in this survey, as do major medical device manufacturers such as General Electric, Honeywell, Hewlett-Packard, etc. Reviews of bio-medical product literature could serve as another source of input to the survey. Several conferences are held yearly which display microprocessors and medical devices, and a canvas of this information would provide additional input to this survey. An example of one such conference, is the MINI/MICRO Computer Conference and Exposition, October 1976, held in San Francisco. Over 100 companies displayed their products at this exposition. Culmination of this phase would result in a complete as possible listing of current microprocessor applications in commercial aspects of health care delivery.

A third phase would focus on the identification of current microprocessor based medical and health care research projects. This information is available in part through the computer based files on existing government research contracts which is available on-line through many libraries. Another useful source for this type of information is the Smithsonian Science Information Exchange. Current activities could also be identified through a review of papers being presented at all relevant computer and medical science meetings (e.g., AIIE, AEMB, AAMI, IEEE).

The primary activity in this phase should be a conference on Microprocessor Applications in Medicine and Health Care Delivery. A Call for Papers would be widely distributed in the early stages of this study. In order to encourage wide participation in the conference, abstracts could be accepted and organized into a Proceedings whether or not they are to be presented. Microprocessor and medical device manufacturers would also be invited to participate in an equipment display area.

The final phase of this type of study would begin immediately after the conference. This phase of the project would draw on the previous identification of those who are active in this subject area. A select group of individuals associated with current or recently completed microprocessor application projects as well as manufacturer representatives would be invited to participate in a workshop session on the future of microprocessors in health care delivery. This knowledgeable group would have the benefit of the previous documentation on the current state of the art and would help formulate a final overview and projection. The results of this overview would be prepared as a report for distribution following the workshop.

SUMMARY

The rational and efficient development of microprocessor applications in health care delivery depends on a number of factors including realistic assessment of application needs, government and private funding directed to these needs, and coordination between computer component manufacturers, medical device manufacturers, medical device researchers, and medical "consumers." This orderly development cannot occur in the absence of a thorough review of where we are now and a projection based on past and current efforts.

REFERENCES

1. Williman, A. O., and H. J. Jelinek, "Introduction to LSI Microprocessor Developments," *Computer,* Vol. 9, No. 6, June 1976.
2. Laliotis, T. A., "Microprocessors: Present and Future," *Computer,* Vol. 7, No. 7, July 1974.
3. Reyling, G., "Considerations in Choosing a Microprogrammable Bit-Slice Architecture," *Computer,* Vol. 7, No. 7, July 1974.
4. "Computers . . . by the Millions, for the Millions," *COMPCON '76 Digest of Papers,* IEEE Computer Society, September 1976.
5. "Tutorial: Designing with Microprocessors," *IEEE Computer Society,* September, 1976.
6. Nelson, D. R., "Microprocessor Applications," *Computer,* Vol. 7, No. 8, August 1974.
7. Symposium on trends and Applications 1976: Micro and Mini Systems, *IEEE Computer Society,* May, 1976.
8. "Minicomputer and Microprocessors: A Tutorial," *IEEE Computer Society,* 1976.
9. "Fundamentals of Microprocessor System Design in Medicine and Biology," Alliance for Engineering in Medicine and Biology, Boston, November 1976.
10. "Practical Computer Applications in Medicine-Theory and Practice," Society for Advanced Medical Systems, Boston, November 1976.
11. Westlake, G., "Microprocessors, Programmable Calculators and Minicomputers in the Clinical Laboratory," in Enland, D. Ed., *Computers in Laboratory Medicine,* Academic Press, 1975.
12. Enlander, D., "Microcomputer Preprocessing in the Clinical Laboratory," in Enlander, D. Ed., *Computers in Laboratory Medicine,* Academic Press, 1975.
13. Smith, M. B. and J. L. Braidwood, "The Use of Micro-minicomputers in Clinical Chemistry: On Line Operation with the SMA 12-60," *Clinical Biochemistry, 8,* 320, 1975.

Natural language knowledge processing

by CHRISTINE A. MONTGOMERY
Operating Systems, Inc.
Woodland Hills, California

ABSTRACT

This position paper treats some advanced concepts which present some short and longer range solutions to the formidable knowledge processing problems:

- Short term—As data bases increase in size, the data management problems associated with maintaining inverted files become more complex, and in cases where the data base is extremely volatile, the software machine can only be driven by a large dedicated mainframe. For most knowledge processing applications this represents an impractical solution to the problem of large scale natural language data management. A more practical solution is to replicate the software machine for text searching and retrieval in hardware. This approach has been adopted in recent hardware developments, based on parallel and associative technology.
- Longer term—Despite the formidable problems involved in simulating the understanding processes of a knowledge worker scanning a text and distilling its content into information of interest to him, automating the generation of data bases which are susceptible to quantitative processing from nonquantitative natural language text is ultimately the only feasible means of exploiting knowledge in the form of natural language. The automated understanding of text is a more complex undertaking than the automatic analysis of natural language queries since a text-based—rather than a sentence-based—grammar is required.

In addition, what is called an introspective data base will be required which assumes a high level of inferential capability, the ability to represent, analyze and synthesize knowledge from many disparate sources, the ability to create, modify, retain and compare patterns of various types and a capability for making probabilistic judgments based on information which may be incomplete and is often erroneous.

INTRODUCTION AND SUMMARY

A data base is a repository of knowledge which in some sense constitutes a model (or models) of a real world state of affairs. The contents of the data base are utilized by knowledge workers attempting to carry out particular decision-oriented tasks.

This paper treats advanced concepts which present some short term and longer range solutions to the following formidable knowledge processing problems:

(1) What happens if the body of knowledge required by the human decision maker is in the form of natural language?
(2) Where do data bases come from?
(3) How does a knowledge worker know what to do next?

The three following sections of this paper address these issues: (1) treats a near term approach to the problem of natural language data management based on associative hardware technology; (2) discusses a longer range approach to the same problem in the context of question (2) above, and (3) focuses on assistance to the knowledge worker involving an active information system built around an "introspective" data base.

NATURAL LANGUAGE DATA MANAGEMENT: A NEAR TERM SOLUTION

In a previous paper discussing corporate data bases,[1] the chairman raised the thorny issue of dealing with non-quantitative corporate data, e.g., the mood of the stockholders, employee morale, legal constraints, etc. Such data generally appear in the form of natural language text, since that is the way humans typically approach non-quantitative data. To put the matter succinctly, if we can't count it or write a formula for it, all we can do is talk or write about it, which we do—indefatigably. This results in an enormous and ever increasing volume of accumulated human knowledge in the form of natural language. These data must be made available to knowledge workers, who are problem solvers and decision makers at all levels.

For the past decade, the accessibility of natural language data to knowledge workers has been achieved via key word and phrase retrieval from text stored on random access devices. The usual retrieval mechanism is an inverted file of all content words in the natural language data base, with associated document pointers. As the data base increases in size, the data management problems associated with maintaining the inverted file become more complex, and in cases where the data base is extremely volatile, the software machine for managing the natural language data can only be driven by a large dedicated mainframe. For most knowledge processing applications, this represents an impractical solution to the problem of large scale natural language data management.

A more practical solution for the natural language data management problem is to replicate the software machine for text searching and retrieval in hardware. This has been the approach adopted in a recent hardware development based on parallel and associative technology, where associative is interpreted as content addressable. In the course of this development, Operating Systems, Incorporated has built and is currently field testing a hardware prototype version of an associative/parallel character matching device which is the critical component of a parallel processor for natural language data management.

The logic and parallel matching circuitry module—called the Associative Crosspoint Processor (AXP)*—is capable of passing 8K bytes of key matching memory with a 1 megabyte data stream for an effective search rate of 8 billion matches per second. The key memory accommodates 50 natural language queries, assuming 25 words each, or any combination of query terms up to 8K bytes. As shown in Figure 1, the AXP intercepts a data stream from a conventional disk controller (at rates of 0 to 1,000,000 bytes/second—the AXP is self-clocking), performs parallel key matching, and sends the results to a cpu.

An AXP, together with a disk system, cpu, and appropriate software, constitutes an Associative File Processor (AFP).** An AFP system is thus capable of searching a sequential, indexed, or randomly ordered, unstructured natural language data base at disk readout speed.

The AFP system configuration which is currently in field test in a customer environment consists of an 8192 byte AXP, a PDP-11/45 cpu, and a Bunker-Ramo 1535 disk controller and 1536 disk with a capacity of 180 million bytes.

The associated software runs under the RSX-11D operating system and includes modules for communicating with a Sperry-Univac 1652 display terminal, query translation and resolution, and other data management modules.

The AXP development appears to provide a cost effective concept for natural language data base management and thus constitutes an efficient, near term solution for delivering non-quantitative data in natural language form to workers engaged in knowledge processing.

Although the AXP approach is in itself an advanced concept, it does not materially improve the situation described in Reference 1 as "the lack of ability to deal precisely with non-quantitative data." The compelling challenge "to bring more and more of such data into the realm of the quantifiable" is then added by the authors. The following section focuses on a longer range approach to natural language data management which satisfies both these goals: that is, the approach described will improve our ability to deal precisely with non-quantitative textual data by analyzing such data into elements that can undergo quantitative processing.

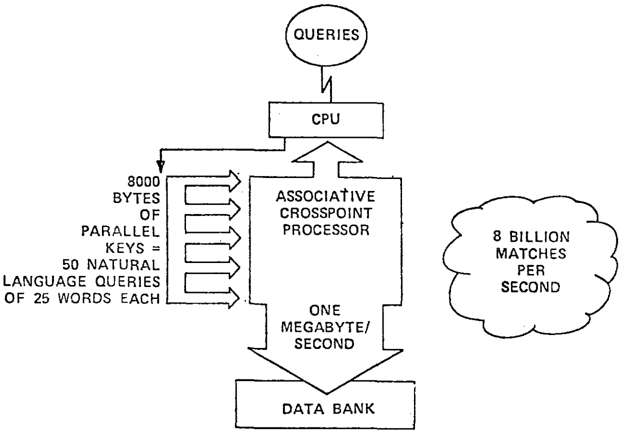

QUERIES
CPU
8000 BYTES OF PARALLEL KEYS = 50 NATURAL LANGUAGE QUERIES OF 25 WORDS EACH
ASSOCIATIVE CROSSPOINT PROCESSOR
ONE MEGABYTE/ SECOND
8 BILLION MATCHES PER SECOND
DATA BANK

Figure 1—Associative crosspoint processor (AXP)

WHERE DATA BASES COME FROM AND HOW NON-QUANTITATIVE DATA CAN BE QUANTIFIED

A great deal of the computer and information science literature is devoted to discussions of data bases, where a data base is defined as a file of records in a particular format containing data elements of specified types. The literature covers data base management, data base administration, data definitions, and so forth; but articles on data base generation are conspicuously absent. To state the issue more graphically, there is a vast amount of literature on the care and feeding of data bases, but none which addresses the more fundamental questions of where data bases come from. In fact, as is well-known, data bases are generated by laborious manual procedures which no one really wants to discuss. Although formatted files of data elements offer considerable advantages for analytic manipulation of data in knowledge processing applications, little progress can be made in developing and utilizing data bases unless the generation process can be automated.

Of particular interest is the automatic creation of data elements from natural language text, since this effectively achieves the transformation of non-quantitative textual data into discrete data elements and relations which can be input to quantitative algorithms. This process involves an in-depth syntactic and semantic analysis of the natural language text in terms of some model of knowledge representation for a particular universe of discourse in order to synthesize meaningful information records. An automated "understanding" of the input text is necessarily implied.

The automated understanding of text is a more complex undertaking than the automated analysis of natural language

* "Associative Crosspoint Processor" and "AXP" are trademarks of Operating Systems, Incorporated, Woodland Hills, California.

** "Associative File Processor" and "AFP" are trademarks of Operating Systems, Incorporated, Woodland Hills, California.

queries, since a text-based—rather than a sentence-based—grammar is required. Referential and anaphoric elements (e.g., articles, pronouns, appositives, reference by synonym) operate within a larger discourse context and consequently, are more difficult to unravel. Moreover, a formalism for knowledge representation such as Minsky's "frames," Schank's "scripts," Reiger's "Commonsense Algorithms"† is necessary to provide a basis for the computer to infer information implicit in the text in order to reduce its content into a set of discrete information records.

Despite the formidable problems involved in simulating the understanding processes of a knowledge worker scanning a text and distilling its content into information items of interest to him, automating the generation of data bases which are susceptible to quantitative processing from non-quantitative natural language text is ultimately the only feasible means of exploiting knowledge in the form of natural language. As an example of such a development, for the last few years, the author and some colleagues have been pursuing the goal of automated data base generation from the natural language text of reports describing events of various types.[3]‡ The objective is to synthesize a data base of event records which can then be statistically manipulated to define event patterns, associated patterns, and changes in such patterns, providing a basis for event prediction. The basic structure for knowledge representation is an event template, which is a relational structure linking the predicate that defines the event or event sequence (e.g., "overfly") with the objects that can form its arguments ("aircraft," "ship"), attributes of the objects of significance to the event ("bomber"), attributes of the event (location, time, and other dynamic attributes), and links to other events (temporal, causal). A text can thus be analyzed into a sequence of one or more event templates, which also form the basic structure of the event records generated for the data base by the synthetic processor.

HOW A KNOWLEDGE WORKER KNOWS WHAT TO DO NEXT—Partnership with an active information system built around an introspective data base.

Now, assuming that such an event data base has been composed and that quantitative algorithms have operated on the data, what have we really done to improve the lot of the knowledge worker who needs this information? The answer is "not much," if the knowledge worker must laboriously seek out the information he needs through a tedious series of data base accesses, involving many false starts and unproductive lines of inquiry, in the course of which the user may become confused and forget his original objective.

Clearly, a serious deficiency of current information systems is that the model on which they are based is essentially passive: it is up to the user to recognize an information need and to seek out the required information via user-initiated communication with some information system. Thus, the user is the active element—the processes of information analysis and synthesis are user dependent and completely external to the system. The burden of information flow and control is on the user, who is forced to define a bottom-up query strategy involving many low level questions to answer a high level question such as assessing the feasibility of acquiring a corporation.

With the volume and complexity of available information on the increase, this burden is rapidly becoming intolerable to the user, whose abilities to assimilate and integrate information are essentially saturated.

What is required is the development of an active information model: that is, one in which the system—rather than the user—is the active agent, assuming the burden of information flow and control according to prestored information goals and algorithms.

The processes of assimilating and evaluating new data, as well as the self-organizing processes built around these, provide a basis for generating hypotheses about future changes in the state-of-affairs represented by the data base. For an event-oriented data base, such hypotheses constitute predictions about future events.

In essence, these on-going analytical and synthetic processes provide the data base with an awareness of its contents; the self-organizing capability also provides a higher level of self-awareness, i.e., of its procedures. It has a self-knowledge such that it can be described as introspecting about its contents to produce new information about the state-of-affairs represented in its contents. The data base's knowledge about its own procedures for operating on data gives it the capability to manifest goal-oriented behavior in planning how to apply these procedures.

Thus, in addition to the already complex dimensions of a data base shown as vectors in Figure 2, we introduce the more significant dimension of "introspectability"—or degree to which a data base is capable of exhibiting introspective, initiative assuming, goal-directed behavior.

An introspective data base is capable of initiating transactions with a user and guiding the user through a complex

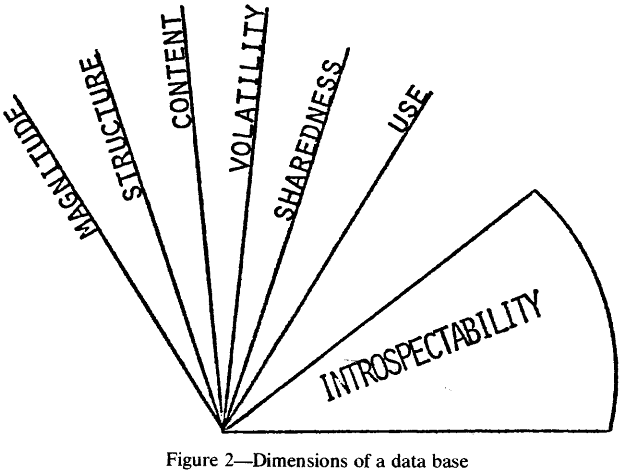


Figure 2—Dimensions of a data base

† These and other approaches to knowledge representation are discussed in [2].

‡ This work was sponsored by the Air Force System Command's Rome Air Development Center, Griffiss AFB, New York.

task-oriented interaction, based on knowledge of its own contents, acquired through introspective algorithms which permit it to:

- monitor newly acquired data to detect changes in the real world system or state-of-affairs it models;
- reorganize its contents to reflect such changes;
- purge irrelevant or obsolete materials;
- validate data in several dimensions;
- request new data to resolve conflicting information or to complete fragmentary data;
- correlate multisource data reporting on the same event;
- assign credibility ratings to incoming data and update credibility ratings of stored data;
- summarize the contents of its files to provide a concise report on the current state-of-affairs;
- acquire additional models of state-of-affairs, objects, and events in order to accommodate differing user world views;
- generate hypotheses about probable changes in the state-of-affairs;
- for each hypothetical change, suggest alternative courses of action and probable consequences of such action;
- present information to the user so as to maximize his ability to assimilate and comprehend.

In addition to the usual data base management and user interfacing functions of storing, retrieving, updating, purging, statistics and report generation, editing, querying, and so forth, the functions of an introspective data base listed above assume a high level of inferential capability, the ability to represent, analyze, and synthesize knowledge from many disparate sources, the ability to create, modify, retain, and compare patterns of various types, and a capability for making probabilistic predictive judgments based on information which may be incomplete and is often erroneous.

Obviously, for a long time to come, successive approximations to this ideal will involve an enlightened partnership between human and machine; however, it is important to keep such a model in mind as an ideal to approximate to. The model thus serves to direct the evolutionary steps in hardware, software, data management, artificial intelligence and other technologies which will be required to progressively offload such functions from the human onto the active information system. As these developments are realized, the knowledge worker's objective and intuitive knowledge processing abilities will be incrementally enhanced by the automated system, allowing him to more fully exploit his analytical and judgmental potential.

REFERENCES

1. Altshuler, G. and B. Plagman, "User/System Interface Within the Context of an Integrated Corporate Data Base," *Proceedings of the NCC*, 1974, pp. 27-33.
2. Silva, G. and C. A. Montgomery, "Knowledge Representation for Automated Understanding of Natural Language Discourse," To appear in *Computers and the Humanities*.
3. Kuhns, J. L., C. A. Montgomery and D. K. Whelchel, "ERGO: A System for Event Record Generation and Organization," RADC-TR-75-51, 1975.

The intelligence cycle—A differentiated perspective on information processing

by PETER G. W. KEEN
Stanford University
Stanford, California

ABSTRACT

This brief position paper presents a framework for mapping computer-based information aids onto the mental activities involved in the full problem-solving process. It argues that these activities are best described in terms of operators—the verbs and commands that the individual uses in a particular stage of the Intelligence Cycle.

The cycle begins with Discovery, the recognition of some signal requiring response. Discovery filters data into information and mainly involves operators that attenuate or amplify data: "alert," "keep track of" (amplification) and "summarize," "report averages" (attenuation). Few computer tools support amplification. The second stage, Interpretation is one where the machine generally outperforms the human mind, especially in inference and statistical analysis. Examples of operators for this stage are "compare," "review" and "suggest." The final stage, Analysis, is strongly supported by management science, especially through optimization models; typical operators are "test the impact of" and "evaluate."

None of our current tools supports the full Intelligence cycle. The position paper suggests that the development of a science of information-processing must both identify the cognitive operators underlying the activities within the cycle and match the technical building blocks to them.

INTRODUCTION

Developments in information-processing are mainly driven by technology; new tools generate new uses. As relational data base or pattern recognition methodologies move from the laboratory to commercial availability, we will extend our ability to help Knowledge Workers make more effective and more efficient use of their information resources. This process is fragmented and relies on serendipity; it also obscures the broader concern:

What is the objective of the science of information-processing?
What are the activities its tools support and augment?

The aim of this presentation is to step back and map the technology into its context—the *cycle* of Intelligence, a process of Discovery, Interpretation and Analysis[1] which begins from an initial awareness of a stimulus or problem and ends with a decision. By clarifying the operations within this cycle, we can both assess the techniques we now have and define the areas of most need and payoff. It is this analysis that should drive the technology and determine the tools we require.

The scheme presented here is a paradigm, a conceptual framework for organizing current knowledge, rather than a formal theory. It derives mainly from cognitive psychology, the science of human information-processing.[2] Its emphasis is descriptive; before we can prescribe tools for improving an activity, we must first describe its dynamics and context. In general, our technical focus has stressed prescription at the expense of this understanding.[3]

The paradigm clarifies some traditional distinctions in discussions of human response to and use of information:

structured	versus	unstructured
data	versus	information
qualitative	versus	quantitative
problem-finding	versus	problem-solving[4]

The presentation is influenced by Miller's[5] definition of a psychology for man-machine systems; as here, he similarly focuses on the *activity* of information-processing, in terms of operators—verbs such as "show me," "scan" and "summarize." Tools can best be understood in relation to the operators they support.[6]

THE CYCLE OF INTELLIGENCE

Most existing tools assist a Knowledge Worker who already has a purpose: a researcher scanning a database, an analyst reviewing a situation or a manager requesting summary reports. The Intelligence cycle begins well before this stage and includes Discovery—identifying and defining the problem and purpose. At an extreme, the Knowledge Worker sits daydreaming while the world around him randomly throws in his direction signal and noise. The cycle begins only when he is alerted to some signal.

Sometimes, the Discovery process is passive; he may not be actively scanning his environment. He may thus overlook "relevant" data.

Many commentators have emphasized that *data* becomes *information* by being filtered through some mental model that gives it meaning and relevance. The mind outperforms most computer-based tools in this process, which relies on alertness, pattern-creation and pattern-recognition. Of course, human limitations on memory, attention span and capacity for assimilating large masses of data make machine support invaluable in many instances.

Discovery filters data into information.[7] In many ways it is the key stage in the cycle, in that it involves problem-finding. Once alerted to a signal, we can generally respond to it if only through some barely adequate rule of thumb or standard operating procedure, but we may easily *overlook* a potentially critical signal. If we can scan more alertly or provide better filters, we reduce the risk of doing so, but the effort needed may exceed our resources. It is thus desirable to automate part of the filter. *What are the criteria for doing so*? The technical focus does not in itself define any criterion.

Stafford Beer,[8] the British cybernetician, describes the filter in terms of *attenuation* and *amplification*. Discovery reduces chaos to singularity. Huge volumes of data are attenuated through summary, selection and aggregation. Key signals are amplified and drawn to our attention. In general we have many tools for attenuation—for reducing the load on our limited attention and capacity; most reporting systems organize mass data into "meaningful" summary. Beer gives an example of amplification in the CYBERSTRIDE system he helped build for the Chilean government.[9] A forecasting algorithm, based on Bayesian decision theory, traps the stream of data on ongoing economic activity and tests if new signals imply a change in current trends and consequently a need to revise the mental model: if they do not *the data is ignored* and the Knowledge Worker not burdened with it. If the input implies a shift, the signal is amplified and the worker alerted to it.

The operators relevant to attenuation include "alert," "keep track of" and "locate any discrepancy." These are clearly different from "summarize" and "report averages," operators relating to attenuation. Information tools for Discovery thus need to be *differentiated*. We have far more tools for attenuation than for amplification; since the latter is central to problem-finding, this is clearly an area of great potential payoff in the selective development of new tools.

Attenuated information needs interpretation while amplification leads to Search, to a response to the signal and a more active effort at Discovery (see Figure 1). In either instance, only now is the worker aware of purpose and ready to select information and analytic aids. The second stage in the cycle, Interpretation, is largely inferential. The operators—the commands the worker gives to his tools—include "suggest," "review," "compare" and "deduce the (often statistical) meaning of." It is fairly easy to map techniques into this activity. Interactive Decision Support Systems such as Gerrity's portfolio management system are explicitly designed in terms of such operators as "SCAN" and "HISTO" (provide histograms).

While man outperforms machine in most aspects of Discovery, he is much less effective in Interpretation.[10] Tversky and Kahneman[11] emphasize the frequency with which individuals make simple statistical errors. Edwards[12] similarly points up our inability to make effective use of information we already have (to update probability estimates, for example). Tools such as MYCIN,[13] which aids medical diagnosticians in the process of inference, exploit the machine's comparitive advantage in Interpretation.

The final stage of the cycle is Analysis, the assessment of interpreted information; this usually results in some conscious decision. Discovery is mainly perceptual and therefore hard to observe or make explicit but Analysis is generally conscious, methodological and sequential.[14] It is concerned more with the use of information than information itself. Its operators include "evaluate," "compare these alternatives" and "test the impact of"—and of course "what if." The tools of management science—optimization and simulation models—obviously support these. It is not clear where man outperforms machine or vice versa. Many problem-solvers prefer to rely on their own intuitive methods although in structured situations they will rely on formal models.

Because Analysis is conscious and sequential, it is often constrained by time and computational effort. In many cases, we simplify the problem to the point where it is feasible for us to handle its demands, even if this involves misrepresentation—and sometimes peversion.[15] In many cases, by automating the operators it involves, we encourage the individual to make *more* comparisons, to enlarge his "bounded rationality;"[16] a frequent benefit cited for interactive computer systems is simply that they allow a user to test out more alternatives. More is not necessarily the same as better; such support may thus improve *efficiency* but not *effectiveness*.[17] In developing tools for Analysis, we must consider which operators they support and what "improvement" means (and is worth). Automative mechanical aspects of Analysis (comparisons, summarization) looses the bounds of rationality and *potentially* releases time for more attention to issues of effectiveness.

TOWARDS A DIFFERENTIATED PERSPECTIVE

The paradigm presented here can be expanded to give a fairly rigorous summary of both human information-processing activities and computer-based aids. The latter are the building blocks for a system to support the full cycle of Intelligence. If we focus on the operators needed by the Knowledge Worker, we will define those blocks in a far more differentiated way than we do now. The issue is not, simply for example, between qualitative and quantitative data or structured and unstructured problems. In the Discovery stage, attenuation generally involves numeric, and amplification judgmental, information. Our tools should reflect this.

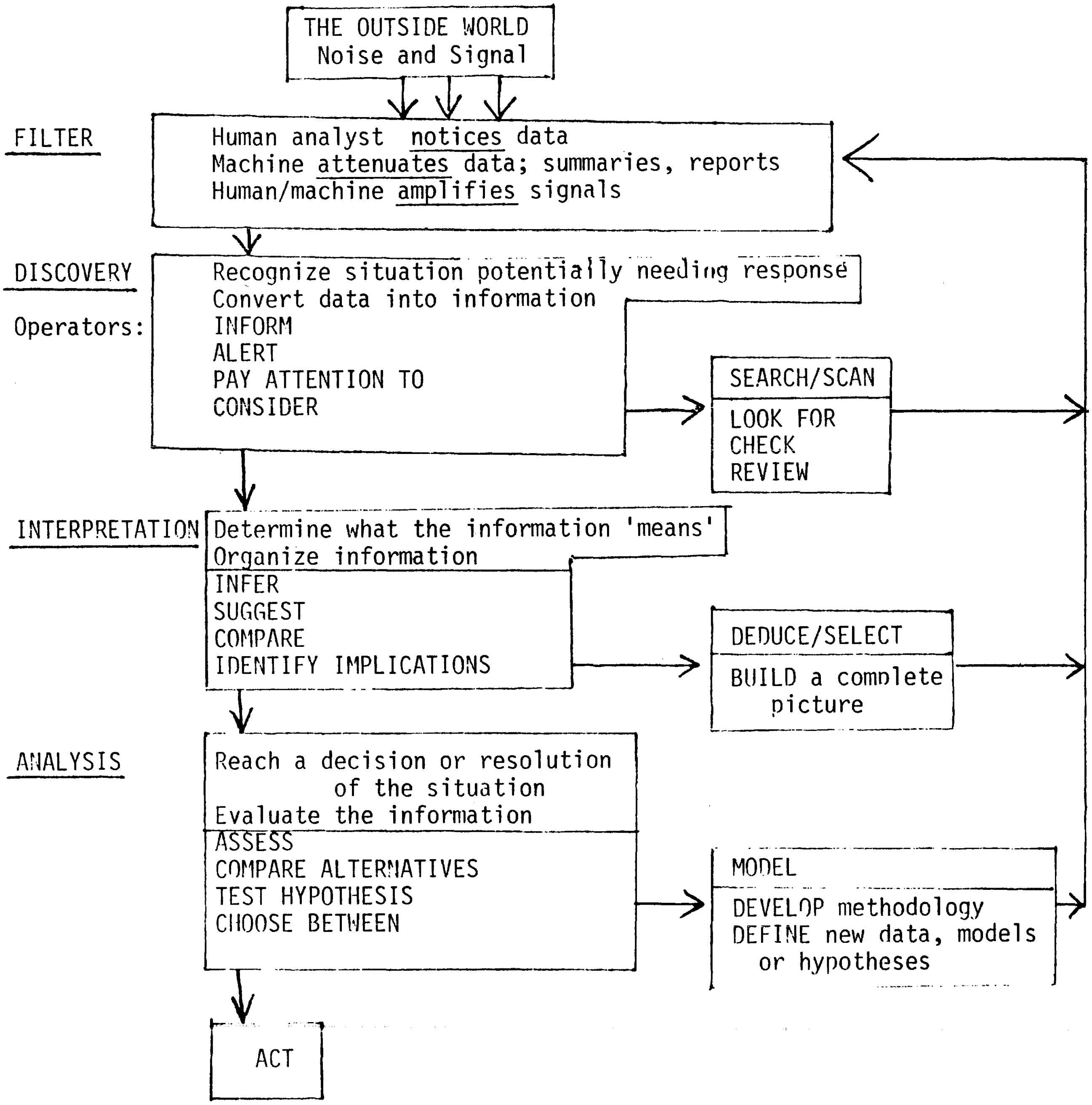


Figure 1—The cycle of intelligence

The paradigm implies selective development of new techniques, matched to specific operators, rather than imposition of the methods on the overall cycle. Any catalog of existing tools will show an abundance of aids for Interpretation and Analysis and near absence of support for Discovery. Any detailed examination of the operators involved in Discovery will, as a corollary, suggest that retrieval methods suitable for scanning and directed search are of little value for amplification and alerting the Knowledge Worker.

There is not space in this position paper to explore the paradigm in detail. The following assertions summarize its intent:

1. None of our tools can support the full Intelligence cycle, nor should we assume that they are other than building blocks;
2. In developing information aids, we must look at the activities involved in the cycle and draw on descriptive

and psychological models;
3. Tools are best defined in relation to operators;
4. We do not yet have a *science* of information processing; even if this paradigm is inaccurate or incomplete we need such frameworks that force us to develop a clearer sense of what our efforts should aim towards and what our techniques really are.

REFERENCES

1. cf Simon's model of the decision process: Intelligence, Design & Choice: Simon, H. A., *Elements of a Theory of Human Problem Solving,* Psychological Review Vol. 65, No. 3, May 1958.
2. More recently, the term "cognitive sciences" has been used by researchers to define a fusion between developmental psychology, Artificial Intelligence & information-processing theories.
3. Keen, P. G. W., and M. S. Scott Morton, *Decision Support Systems: An Organizational Perspective,* Addison-Wesley (in press), Chapter 3, Decision Making: Description versus Prescription.
4. This distinction is discussed in Leavitt, H. A., *Beyond the Analytic Manager,* California Management Review, Spring-Summer, 1975.
5. Miller, R. B., *Psychology for a Man-Machine Problem-Solving System,* Technical Report TR00 1246, IBM Data Systems Division, 1965.
6. Gerrity, T. P., *Design of Man-Machine Decision Systems: An Application to Portfolio Management,* Sloan Management Review, Winter 1971, for an example of a formal design strategy based on this approach.
7. Druzhinin, V. V., and D. S. Kontorov, *Concept, Algorithm and Decision,* Soviet Military Thought No. 6, U.S. Government Printing Office, 1972.
8. Beer, S., *Platform for Change,* Wiley, 1975.
9. Beer, S., "Fanfare for Effective Freedom," in Reference 8, pp. 423-451.
10. Dawes, R. M., "Objective Optimization under Multiple Subjective Functions," in J. L. Cochrane and M. Zeleny, eds., *Multiple Criteria Decision Making,* University of South Carolina Press, Columbia, South Carolina, 1973, 9-17.
11. Tversky, A., and D. Kahneman, *Judgment Under Uncertainty: Heuristics and Biases,* Science, Vol. 185, September 1974.
12. Edwards' work is summarized in Beach's comprehensive survey: Beach, B. H., *Expert Judgment About Uncertainty: Bayesian Decision Making in Realistic Settings,* Organizational Behavior and Human Performance, Vol. 14, pp. 10-59, 1975.
13. Shortliffe, E. H., S. G. Axline, B. G. Buchanan, and S. N. Cohen, "Design Considerations for a Program to Provide Consultations in Clinical Therapeutics," *Proceedings of the 13th San Diego Biomedical Symposium,* February, 1974.
14. However, there are significant differences between individuals' cognitive strategies and styles of problem-solving. See McKenney, J. L., and P. G. W. Keen, *How Managers' Minds Work,* Harvard Business Review, May 1974.
15. Taylor, R. N., *Psychological Determinants of Bounded Rationality and Implications for Decision Making Strategies,* Decision Sciences, Vol. 6, No, S, pp. 409-429.
16. Simon, H. A., *Administrative Behavior,* (2nd Edition), McMillan, 1957.
17. See Reference 3, Chapter 1 for a discussion of the relevance of this distinction for the design of computer systems.

Plans for a program in medical information science

by ALLAN H. LEVY and THOMAS T. CHEN
The University of Illinois
Urbana-Champaign, Illinois

ABSTRACT

Although the "information revolution" has pervaded nearly every other aspect of modern industrial life, the health care system—replete with an overdose of information—lags behind in management and control. Although there are isolated examples of the use of information technology in the health delivery system, these are fragmented and usually limited to isolated processes, rather than to an integrated information system.

This underutilization is largely the result of an isolation of the health professional and the physician from any substantive knowledge of computing. "Computer ignorance" has led to overexpectation and consequent disappointment and failure. We propose that formal structured training of health professionals at the postgraduate level will provide the knowledge base that will lead to intelligent planning and practical implementation. The details of a graduate training program, with a strong embedded research component, are presented.

INTRODUCTION

Although the nation is in the third decade of the "information revolution," clinical medicine and the health care delivery system are only now beginning to realize the substantial impact of computer technology and information science. Developments in medically-related computing date back scarcely ten years. In health care delivery, the scope of applications has substantially broadened: early efforts were limited to hospital accounting; the present spectrum encompasses applications ranging from health information systems to laboratory automation to computer consultation.[1] In addition, the digital computer has become a research tool of high utility to both the basic medical scientist and to the clinical researcher.

Computer technology has made practical the accumulation of large data bases; their actual utilization, however, has been limited in both patient support and in health planning functions. It is ironic that some of the heaviest users of large health data bases have been insurance carriers concerned with economic risk factors. Parenthetically, this reminds us that substantial ethical and legal issues relating to individual rights of privacy must be taken into account when considering the management of health data bases.[2]

Minicomputers are now out-pacing monolithic large computers in terms of both variety and versatility. Costs of processors have decreased almost exponentially. Minicomputers are now widely used in medicine, for patient monitoring, for control of laboratory instruments, and for a wide variety of other applications. The effective amalgamation of large central computers with minicomputers into integrated information networks is a developmental area of high potential.[3]

Educational and training activities related to medical computing lag far behind.[4] The responsibilities for biomedical computing have been assumed in an unstructured manner by most American medical schools. In some, departments of physiology have subsumed the task and have placed primary emphasis on research applications involving hardware development. In a number of medical schools, special computer research resources have been established under the Biotechnology Resources Program of the National Institutes of Health; these function as service centers, as well as local foci for computer-related research. The attention of such centers to medical education has been relatively minor, primarily because of the statuatory limitations to research inherent in the program.

Consequently, it is fair to state that although computer applications in medicine are increasing both in variety and extent, there has not been a commensurate increase in developing programs in medical information processing science. There is the real danger that unless more programs are developed, health professionals will become less capable of adequately specifying sensible requirements for medical information processing, will be less able to make effective use of computer technology, and will be less capable of evaluating the impact of the computer technology to which they are exposed. Furthermore, if such uncoupling between health professionals and computer scientists should continue, new developments will inevitably grow less relevant to the real needs of the health care delivery system.

It is therefore postulated that effective educational programs in medical information science are a necessary component of a health science educational institution. We propose that such a program should be designed and effectively implemented.

PROGRAM DEVELOPMENT

The establishment of the School of Clinical Medicine at Urbana-Champaign campus of the University of Illinois is projected for July, 1977 with the first class of students entering in Fall, 1978. A proposal to establish a Center for Medical Information Science in the School of Clinical Medicine [Appendix A], Urbana-Champaign has been submitted to the planning officers of the School of Clinical Medicine. It will be a basic instructional unit within the School of Clinical Medicine. Since July 1975, ongoing programs have been coordinated, new research projects have been initiated, and the instructional goals and learning objectives in medical information science for professional students have been developed. An application for a physician training grant in medical computer science has been submitted to the National Library of Medicine and has been approved.

Instruction

The goals and learning objectives of instruction in medical information science have been defined as:

Goals

- to enhance the understanding of computer science and its relations to clinical medicine, basic health sciences and the health care delivery system.
- to encourage career development in medical informatics.

Learning Objectives

To provide physician-in-training:

- the basis for understanding in cybernetics, mechanisms of human intelligence, and the clinical decision processes.
- the means for acquiring, structuring, analyzing, and displaying data to enhance its usefulness in the health care process.
- skills in the appropriate important applications of computers in clinical consultation; medical record maintenance; pattern recognition in ECG, EEG, and other physiological recordings; utility of computer assisted tomography; and patient monitoring.
- experiences in the usefulness of simulation and modelling, both as a learning tool and as a means for enhanced decision making.
- skills in the use of problem oriented and computer based medical records through the development of a school ↔ clinical affiliate system.
- opportunities for complementary experiences via computer-assisted and managed instruction.

Program Description

- Undergraduate Medical Education

The instructional efforts will be directed to the needs of the medical students in the School of Clinical Medicine as listed in the learning objectives above. It will also be coordinated with the activities at the School of Basic Medical Sciences, Urbana-Champaign (SBMS-UC).

The curriculum of SBMS-UC is self-paced and is problem oriented. Computer based instructional materials and a sophisticated diagnostic examination system through the PLATO IV terminals are an integral part of the present basic science experience. Thus, students here are familiar with computers and terminals. We will capitalize on such familiarity: the use of computer by medical students as an integral part of their regular medical education, in our opinion, will be the only effective method for providing an operational understanding of medical information science.

During their clinical years, students will be using a computer based and problem oriented medical record system which is currently being developed here. A few lectures will be given in medical information science; but more important, the use of terminals in actually creating and inspecting medical records will provide that element of familiarity which can be obtained in no other way. Participation of developmental and research activities in this field will be encouraged on the same basis as in other clinical areas. Education in medical information science will be on the same level as biological and physiological components of medicine.

- M.D. Master in Computer Science Training (This program has been approved by the National Library of Medicine. Funding will begin in fiscal 78.)

The purpose of the program is to provide an effective medicine computer science interface for four physicians per year who have completed or partially completed their residency training in a conventional specialty. They will be enrolled here in an individualized but intensive two year program in computer science. Upon completion of the program, the trainees will receive the degree of Master of Computer Science from the Department of Computer Science, University of Illinois and may pursue the doctoral program if they so desire.

It is anticipated that demand and opportunities for graduates from this program should be and will remain high for the foreseeable future. Most projections indicate that the health field will become one of the major, and perhaps the major user of computer technology after the developmental threshold of minimally acceptable systems has been reached. As physician graduates of a program of this type are needed to reach this threshold, their own work should contribute to the future rapid expansion of demand for similarly qualified personnel.

Research

The training experiences of our students will be conditioned strongly by the ongoing medical computing research activities of our faculty. Our research activities can be summarized in two areas:

1. PLATO-Based Health Science Network Activities

Since 1972, SBMS-UC of the University of Illinois College of Medicine has been extensively committed to the development of basic medical sciences CAI materials. During the last three years, under contract support from the Bureau of Health Manpower, a group of faculty has been developing basic sciences lesson materials on the PLATO IV system.[5] The target of this project is 300 hours of CAI lessons in basic medical sciences. There are about 200 lessons currently available.

Since the College of Medicine of the University of Illinois consists of six schools in four cities (Chicago, Peoria, Rockford, and Urbana-Champaign), it was natural to think of a small medical CAI network for the schools. Our work has been directed toward coordinating system development and use by the several campuses.

In late 1973, we recognized that the College of Medicine PLATO IV health science activities should encompass institutions outside the University, in order to facilitate more multi-institutional participation in lesson development and delivery, and to get a wider experience with student needs. We consider it desirable to explore the feasibility of expending these CAI activities to a "Health Science Computer Network" which includes computer management of instruction (CMI), as well as CAI and a medical information system (MIS).[6]

The four campuses of the College of Medicine are the nidus of this presently existing *ad hoc* health sciences network. The University of Southern California, the University of Oklahoma, the Southern Illinois University, the University of Tennessee, and the University of Maryland are among the participants. The Regional Health Resource Center, the Champaign County Blood Bank, the Mercy Hospital Pathological Laboratory, three private physicians' offices, and the University of Illinois McKinley Health Service are also local users in the Urbana-Champaign area. A terminal at the Lister Hill National Center for Biomedical Communication is presently in use for their staff's observation of the activity. Eighty terminals are currently in operation, within and without the University of Illinois College of Medicine.

2. A Depository Health Science Computer Network

One of the objectives of our research in the PLATO-based activities is to specify the design for the creation of a dedicated regional medical information system suitable for linkage to networks of small individual units. The current research in developing a depository health science computer network is an effort directed toward that goal.

A depository network consists of two types of nodes: the depository node and the local node. The depository node is an information center into which the local nodes put and through which they inquire and access information. Each local node provides independent computational power and support, administratively and physically, the man-machine interface to its users.

The successful implementation of a depository computer network requires the presence of both local and depository nodes. The experience gained in the past indicates the practicality of a small to medium size computer supporting PLATO-like system for such activities. We have proceeded to implement and operate a PLATO-like system on a minicomputer.[7] System development efforts are now directed toward the refinement of the existing system, the development of a multiple terminal, multiple user, PLATO-like system. CAI, CMI and MIS programs developed on the PLATO IV system are being transferred to the small system. Such development is important to the application of computer technology to health education and health care delivery because:

- in the framework of such a system, health professional education, patient education, consultation, monitoring, and referral programs can be effectively integrated and utilized.
- the distributed computer network makes it feasible to share courseware developed at different institutions on a locally administered computer system.
- via such a system, it is feasible to develop a comprehensive, incremental, and detailed health and clinical data base. Such a data base is essential to resolving problems in other areas of computer application to health care, particularly in those domains involving probabilistic decision and prediction algorithms, and those involving validation of criteria of quality of care—decisions that are now largely made on an intuitive basis.

SUMMARY

Plans for a program in medical information science have been developed at the Urbana-Champaign campus of the University of Illinois College of Medicine. Curricula for undergraduate and postgraduate medical students are being planned. Several research projects in medical computing are well under way.

Like many other institutions, our University is facing reduction in federal funding for many programs. The relevance of health care technology to a better health care delivery system is under close examination by various governmental agencies. With the tight budget situation, every institution is now experiencing greater cooperation, and a coordination among all institutions and members of the health computing profession is a necessity for the future progress of medical computing systems. A greater emphasis should be placed on the direct education of senior medical and other health professional manpower: the present abyss between potential and realization may well be largely due to the expectation by health professionals without the accompanying knowledge that would enable them to effectively implement. We believe that structured training will remedy this and provide a more realistic basis for a national cooperative effort.

APPENDIX—GOALS AND OBJECTIVES OF A PROPOSED CENTER FOR MEDICAL INFORMATION SCIENCE

GOALS

- To advance the quality of health care delivery by increasing understanding of the effective uses of computer science and information processing in clinical medicine; and
- To advance basic knowledge in medicine, both clinical and experimental, through research in medically-related information science.

OBJECTIVES

Instructional

- Develop an educational program in the principles and use of information science and computer science as related to clinical medicine and health care delivery;
- Define the specific learning goals and objectives in information science for medical students (including students in basic science and in the clinical program), as well as students in allied health science programs:

    i. Identify objectives and goals common to all health science students.
    ii. Define specific objectives for individual health science careers (including allied health) relevant to the needs of the particular disciplines.

- Develop teaching and instructional programs to fulfill the defined objectives. The instructional programs are designed to be integrated into the overall curriculum and learning goals of the students. They encompass:

    i. Basic principles of computer science related to medicine.
    ii. Information processing technology related to data analysis and medical records.
    iii. Systems design related to health care delivery in community health maintenance, ambulatory care systems, and hospital information systems.

- Participate in curriculum development and the definition of learning goals and objectives in information science related to medicine for University undergraduate students (particularly those majoring in the Department of Computer Science).
- Participate in instruction to such undergraduate students.
- Participate with the Department of Computer Science in the design of a curriculum for graduate instruction in medically-related computer science for those students majoring in Computer Science who wish to specialize in health areas.
- Provide opportunities for advanced training (fellowships) for physicians and others on the doctoral level in medical information science.
- Serve as overall coordinator of the PLATO Medical CAI project within the College of Medicine.

    i. Assist in the definition of goals and objectives.
    ii. Assist in preparation of instructional material.
    iii. Develop tools for the evaluation of the utility of computer-assisted instruction in various settings.

Research

- Engage in original research and development in medical information science.
- Broaden scientific knowledge by the delineation and identification of important interdisciplinary problems.

    i. Define those areas where important health-related problems may be solved only by the application of skills from medical and other University disciplines.
    ii. Cooperate in the structuring of such interdisciplinary research and participate in its execution.

- Provide opportunities for research and research training for medical students, graduate students and undergraduates.
- Advance scientific knowledge by contribution to the scientific literature.

Service

- Professional

    i. Provide professional and technical support to the affiliated hospitals in the areas of medical computing.

        - Assist in the development and operation of medical data bases shared by the University and affiliated hospitals.
        - Assist in the creation of specialized information modules and information systems utilized jointly by the University and affiliated hospitals.

    ii. Provide a supportive educational resource in the field of medical computing and information systems for health professionals throughout the State.

iii. Make available consultative support to health professionals throughout the State.

iv. Provide special support services for health-related information processing for health professionals and State and local planners.

- Assist in the design and maintenance of health resources and health manpower inventories.
- Develop software for specialized health data management needs.

v. Provide assistance, consultation, and support to health professionals and other departments in the University involved in:

- developing systems of data collection and classification.
- developing models of clinical management.
- developing computer-based quality assessment and assurance systems.

vi. Assist other faculty and departments by providing a computer and information technology support base for categorical discipline research problems.

- Public Service

Assist in the wider understanding of the impact of technology and systems organization on the quality of health care delivery.

REFERENCES

1. Collen, M. F., "Hospital Computer Systems," Wiley, Johnson, and Sons, Inc., 1974.
2. Lindberg, D., "Special Aspects of Medical Computer Records with Respect to Data Privacy," *Proceedings, Second Illinois Conference on Medical Information System,* SBMS, University of Illinois, 1976.
3. Chen, T. T., B. T. Williams, and A. H. Levy, "A Depository Health Computer Network," 8th Annual Conference, Society for Advanced Medical Systems, 1976.
4. Anderson, J. and J. M. Forsythe, Editors, *World Conference and First Medical Informatics,* Medinfo 1974, North Holland.
5. Bitzer, D. and D. Skaperdas, "The Design of an Economically Viable Large-Scale Computer-Based Education System," Report x-5, CERL, University of Illinois, 1971.
6. Chen, T. T., "An Overview of a Health Science Computer Network," *Digests of Papers,* CompCon Fall 75, IEEE Catalog No. 75CH0988-6C, 1975.
7. Chen, T. T., A. B. Baskin, and D. Jones, "A Local Node of a Health Depository Computer Network," to be published.

The health care computer user—"Where will we find the integrators?"

by ROGER H. SHANNON and MARION J. BALL
Temple University
Philadelphia, Pennsylvania

ABSTRACT

This paper assumes that the primary objective of the medical care system is patient welfare. To best accomplish this objective the medical specialties must communicate and be well coordinated. The organization and distribution of information, which is the domain of the medical information scientist, is of central importance. In practice, medical information scientists often influence reorganization of human institutions, and thereby become change agents. They are commonly consulted about problems crossing disciplinary lines and see trends that allow them to predict and guide future developments. Many information scientists also simultaneously fill a role in some other specialty so that they exert influence both from inside and from outside the medical practice structure. The medical information scientists described are in an excellent position to be professional integrators. Good integration like any art is predicated on appropriate attitudes and has basic skills that can be taught. This paper suggests that formal preparation to integrate the activities of the medical practice environment is a desirable adjunct to the traditional preparation of students of medical information science.

WHERE WILL WE FIND THE INTEGRATORS?

The traditional academic hierarchy has served medicine well as a framework for dividing tasks in a knowledge-rich environment. However, specialty has tended to produce autonomy if not mutual isolation of one discipline from the other. Since patients continue to function as integrated units and patient care continues to be medicine's prime objective, there exists a constant counter-demand to integrate the diverse activities of the specialty oriented health care team. But where do the integrators come from?

To answer the question, one must make some presumptions about integration. Integration can perhaps best be attacked as a boundary problem. It therefore focuses on communications—both technical and human. "Communication" has been defined by Mortensen and Sereno as "a process by which senders and receivers of messages interact in given social contexts."[1] It is fairly simple to see embedded in this definition the concept of input- a link- and output. But this still leaves unresolved the issues of data compatibility, data perception (information), and social context with all the mystery of its deep structure. Going one step farther, one may presume that the problems of data are solvable, at least at a technical level. The issue of social context remains.

It is a position of this opinion paper that to exclude consideration of social context from the process of integration makes the process not only worthless but often dangerous. Good integration is still, in the last analysis, an art, and the artists are still emerging randomly and by accident. It would seem to be precarious for the stewards of complex systems to rely on accident to produce an essential skill.

Fortunately, the principles underlying these artists' expertise are being subjected to increasing scrutiny and an abundant literature is appearing addressed to "communication theory," "social organization," "change dynamics" and a host of other related subjects. The tools exist for us to recognize aptitude and train artists of integration. These people will then be competent generalists as well as specialists.

Since a major concern of Medical or Health Information Science is systems, and since systems imply integration, the question which must be confronted is "Should the skills of integration be included in the basic curriculum of the Medical Information Scientist?"

A tight academic viewpoint might well generate a negative answer, contending that the medical information scientist should be limited to the technical aspects of interface design. This view would be predicated on the assumption that other specialists exist to deal with associated problems. In addition, such an academician could rightly assert that the field is already overburdened with content and that the addition of more material could only act to degrade the quality of current medical information science programs.

This view of any discipline, perceived from within, rightly emphasizes quality and tends to keep the specialty manageable. It also pragmatically recognizes that it is commonplace to deal with both information overload and increasing social complexity by repeated division of responsibility. It fosters the ability to maintain personal and academic order without which there would be a rapid erosion of effectiveness.

Unfortunately, the process of progressive specialization, as beneficial as it may be to integrity *within* the specialty, defaults on the need to preserve integrity *among* the specialties. The larger social or academic blocks which spawned the specialties are left untended, and the network of boundaries which permeates the interstices among the specialties is attacked late and with a total absence of coordination. In its default, specialization can be both socially and academically destructive.

This destructive element has not gone unnoticed. The public response is seen in consumerism, centralization of control, the explosion of the planning industry and, indeed, even among esoteric academics in the quest for a general system theory. These responses bear testimony to the fact that the ultimate purpose of specialties resides in combined rather than isolated effect. They are testimonies of the value of integration and of the need for competent generalism.

It may then be acceptable to say the skills of integration have value to the medical information scientist who leaves training to be a fulltime integrator, but what value will such skills have for the other graduates?

Sias in "An Analysis of the Job Market for Biomedical Computer Scientists"[2] has suggested that training program graduates will be absorbed by three major categories of medical computer applications, (1) business data processing, (2) database management, and (3) automated medical instrumentation. In his conclusion, Sias states that, "It is likely that biomedical computer scientists will be matched most appropriately to positions with computer systems supporting large medical data bases that will be needed to establish a nationwide comprehensive health-care system." Regarding development of this field he said, "Computers have been found useful in the ambulatory care setting so it is likely that this rapidly growing segment of medicine will require significant support while the larger hospitals will over a period of time introduce computer automation first in such areas as clinical laboratories and later in medical records and hospital-wide information and communications systems." A listing of graduates of training programs supported by the National Library of Medicine, while still too few in number to be a reliable trend setter, tends to support the Sias prediction and suggests a fourth, perhaps self limited, market in university faculties.[3]

Business data processing appears to find its manpower in the general pool of data processers or through training new personnel on the job. Consequently, detailed preparation for that market need not be a major concern in training medical information scientists. Medical instrumentation is heavily oriented to engineering and is an appropriate field for the technically oriented computer scientist. The true medical information scientist, as his title implies, is the individual who may be expected to find his way into the world of data bases and medical systems.

It appears generally recognized that the popularity of large computers shown in the 1960's is being overshadowed by a rush to smaller computers in this decade. The pressure exerted by the maxis to agree and share can once again be circumvented by those who would rather go it alone. This is resulting in numerous applications that are counterparts of the laboratory systems to which Sias refers above. Most of these specialty oriented, standalone systems, although offering some service of process control, at a minimum develop some raw data for human consumption. Many systems include small verbal records even when their primary purpose is to produce data in other forms. Some examples in addition to the now familiar laboratory systems, are physiologic monitoring systems which display analog signals and derived numeric data but frequently keep patient mini-records as well; nuclear imaging employs computers to manipulate digitized pictures and to derive associated data, but patient records and registries are a frequent accompaniment. The computerized tomographic (CT) scanner probably outstages all the other special applications at the present time, and it too carries verbal data with the expectation that there will be more in the future.

Primarily verbal stand-alone systems are also proliferating. Specialty registries, patient records banks, reporting systems and scheduling applications (many of them structured) are a few examples. These merge smoothly into ambulatory care and the chain of development suggested by Sias where burgeoning hospital information and medical records systems are bearing out his predictions. Just over the horizon, one can see the combination of these communications oriented systems with more effective decision systems including the promising work in artificial intelligence. At some time, integrated with communications, the data base will have come into its own, and the stand-alone systems of today will have become prologue.

It is inconceivable that medical information scientists can be fully effective designers, administrators or implementors during the socially critical years comprising their professional lives without understanding and specific skill in the art of integration. It is also inconceivable that medical information scientists will always be in positions where all of the desired support expertise will be available. In fact, if past experience is any indication, they will often be the lone "vox clamantis in deserto"—the voice crying in the wilderness, and their main resource will be themselves.

If the current popularity of stand-alone systems can be accepted as only a phase of the trend projected in the preceding paragraphs, one would then have to reply with "yes" to the question of "Should the skills of integration be included in the basic curriculum of the Medical Information Scientist?"

If the medical information scientist were entering a society with a mature demand, and his only role were to be filling that demand in a pre-defined manner, then the value of adding generalism to his expert base might again be questioned. But he cannot expect the luxury of such a predictable professional life. He will live, instead, in a world of transition and turmoil, and his professional activities will be punctuated by surprise.

There are four roles which many medical information scientists will play, that may further accent the need for skills of integration. These are the roles of (1) change agent, (2) internal medical and health care consultant, (3) multi or at least dual specialist and (4) futurist.

Change agent

One becomes a change agent whenever he is charged with the responsibility of introducing new technology or for revising organization. By their nature, computer systems, small or large, are innovations, and the medical information scientist will be viewed as their champion. It makes no difference whether the computer scientist acts as application consultant, systems analyst, administrator or advisor; he will be the harbinger of change and instability. How he acts will affect the course of change, perhaps even to the extent of success or failure. Basic principles of psychology and management of change have been well explicated[4,5] and their teaching can be well accommodated by any degree granting educational program.

Internal consultant

By virtue of his association with new methods and because of the cross disciplinary nature of his specialty, the medical information scientist is often consulted when other members of his institution are actively considering innovation. These other members may be themselves change agents or in search of solutions to specific problems or merely curious. In addition the computer scientist may expect to make contributions to in-house educational programs. The quality of contribution made by the medical information scientist will be influenced by his understanding of the boundary conditions in his institution, including interrelation of medical expertise, the impact of financial constraints and policy, and internal politics. To truly integrate, he will have to understand overall medical objectives.

Dual specialty-niche

During at least the next few years of the current transitional period, the medical information scientist may have to continue to operate in two different specialties simultaneously. This practice is perhaps most clearly delineated in the domain of the M.D., Ph.D. Any one who has personally investigated the job market has recurrently encountered situations where expertise in medical information science is recruited with no appropriate slot. Instead the physician is offered a clinical position and salary with the expectation that he will divide his time. Cursory reflection will reveal that much current research is conducted by people fitting this description. On the other side of the coin, academia is beginning to have people with advanced degrees in information science apply for training as M.D.'s or for residencies in medical specialties. Some of these candidates are already well established, recognized professionals in their original discipline. Others, according to the reports to NLM, are recent graduates of the NLM funded training programs.[3] In at least the near future, one can expect to see many more niches provided by established disciplines than by formal positions for medical information scientists. Those filling dual roles will be living interdisciplinary lives which will be more comfortable and of greater value if they are generalists in possession of skills of integration.

The futurist

Finally, the medical information scientist will have to be to some extent a futurist. The health care science is changing at an astounding rate, both clinically and politically. Shifting power is redistributing the machinery of goal and policy setting. By the time an innovation is established, it is on the way out. The people who are most effective in such mercurial times are those who are reacting, not to what is, but to what will be. The success of these people is the proof of their ability to predict. Good futurism is still largely intuitive, but it is yielding to examination. Techniques of forecasting and simulation have become common tools. There is a literature on the sociology of complex organizations. And "long range planning," although contraversial, has become a household phrase.

One well known futurist, Robert Theobald,[6] has posed the challenging question, "Are we more interested in coming up with sophisticated answers to obsolete questions than defining the new questions that are developing as the world continues to change?" The call for papers for the Conference led off with the still young question, "How should health computer facility directors and staff be trained to interact effectively with actual or potential health users?" As a partial answer, this paper has suggested that the early efforts of a minority of programs to include training in the skills of integration and to foster the attitude of generalism be recognized and supported.

In conclusion one should again ask, "Where will we find the integrators?" The answer—probably always from a variety of sources, but Medical Information Science should be a major contributor.

REFERENCES

1. Mortensen, C. D., and K. K. Sereno, *Foundations of Communication Theory,* Harper and Row, New York, 1970, p. 5.
2. Sias, F. R., Jr., "An Analysis of the Job Market for Biomedical Computer Scientists," 14th Annual Southeastern ACM Conference, April 22-24, 1976.
3. National Library of Medicine Report of Training Program Directors, May 11-12, 1976.
4. Judson, A. S., *A Manager's Guide to Making Changes,* John Wiley and Sons, Inc., New York, 1966.
5. Shannon, R. H., and M. J. Ball, "The Role of the Medical Information Scientist as a Change Agent in Future Health Care Delivery," *Journal of Clinical Computing,* VI (1): 11-13, 1976.
6. Theobald, R., *Beyond Despair,* New Republic Book Company, District of Columbia, 1976.

NAA—An approach to analyzing backpanel crosstalk

by J. S. HEBHARDT, C. F. GROVES and R. BARDAS
Sperry Univac: Philadelphia Systems Development (PSD)
Blue Bell, Pennsylvania

INTRODUCTION

Commensurate with the trend towards higher packaging density in wirewrap backpanels is the problem of crosstalk between the wires which distribute signals. The Noise Analysis Approach (NAA) described in this paper is an attempt to develop a general technique to assess the status of a new machine or a new packaging approach with regards to crosstalk noise in the backpanel. Although generally applicable, the technique was first applied to the Sperry Univac 90/30 computer system. This system is microprogrammed, and is the first user of a TTL packaging technique of high backpanel wirewrap density called Modular Performance Packaging-1 (MPP-1).

Picture a computer backpanel with several thousand wires each carrying signals that switch in response to an operating program being executed by the machine. The spacing between wires is not uniform, nor are the wires at a uniform height above a ground plane. The complexity of this picture and the lack of uniformity make a theoretical prediction of crosstalk virtually impossible. Backpanel crosstalk is a pervasive problem affecting almost every signal to some extent. Hypothetically, if the number of signals were small, "worst case" crosstalk measurements could be made on each signal in the backpanel. These signals could then be ranked from low (those with the least crosstalk) to high (those with the most crosstalk). The large number of signals in a computer backpanel, however, and the difficulty of defining worst case conditions in an operating machine environment make this approach impractical.

Another alternative to theoretical prediction is to use an empirical/statistical approach—define the parameters critical to the creation of crosstalk; derive an estimating equation relating these parameters; apply the equation to signals and their associated backpanel wires; and, rank signals based on the value computed using the estimating equation. This ranking is an estimate (based on the parameters defined) of the "worst case" ranking mentioned previously. To use the ranking, signals with high estimated crosstalk (e.g., top 200) can be measured on the test floor in an actual machine. If no crosstalk problems are found in these cases, a confidence is achieved concerning the crosstalk susceptibility of the remaining signals. If problems are found, the ranking data is useful in pinpointing logic areas that are most susceptible and should be re-wired. This, in brief, is the framework of the Noise Analysis Approach.

Requirements

To proceed with the approach outlined, the following basic elements are required: (1) a physical description of backpanel wire routing, containing information concerning adjacency between wires; (2) a means of linking crosstalk estimation to the actual signal switching activity which exists during program execution; (3) a crosstalk estimating equation; and, (4) an "analyzer" to combine the above and produce a ranking. Data of the type required in item 1 has been provided by PSD Design Automation for past systems through the Length of Adjacency Process (LOAP), which describes backpanel adjacency and routing for wires associated with specific signals selected by the user. The existing PSD Logic Simulation Process was used to provide the data required by item 2. The 90/30 System had undergone extensive logic simulation during its development and accurate models were available. The crosstalk estimating equation required by item 3 was not available, and had to be developed by the PSD Hardware Applications Section. The equation was synthesized from data obained from crosstalk measurements made on wires threaded through a wired 90/30 backpanel. An "analyzer" called the Noise Analysis Program (NAP) is the fourth and final element required in the Noise Analysis Approach outlined above. The function of NAP is to compute a crosstalk estimate by applying the estimating equation to each set of wires associated with a signal. The estimate represents the susceptibility of that signal to crosstalk, and is used as the basis for signal ranking.

NAA BASIC ELEMENTS

This section describes in more detail each of the four basic NAA elements mentioned previously: (1) Physical Backpanel Description; (2) Logic Simulation; (3) Crosstalk Estimating Equation; and (4) Analyzer (NAP).

Physical backpanel description

90/30 Backpanel hardware

The 90/30 wired backpanel assembly (motherboard or module) consists of two rows of connectors (51 connectors/row) assembled on a drilled pc board (12"×14.5") with voltage artwork on the connector side and ground artwork on the opposite or wire wrap side. Six such modules contain the 90/30 Processor logic. Connector pins protrude through the pc board and are used for wire wrap connections, and the area between these pins form channels for the routing of backboard wires. Figure 1 shows an enlarged portion of a module and illustrates ground artwork, pin spacing and channels. Three levels of wire wrap are allowed per pin, with level 1 being closest to the ground artwork and level 3 being at the top of the pin at a maximum distance from the ground artwork. The wire used for wrapping is a mylene insulated, 30 gauge wire with an O.D. of .0175 inches. A logic card consists of either Series 74 or TTL III Integrated Circuits on a 1.45"×5.25" printed circuit board which mates with a motherboard connector. Where possible, printed circuit card artwork is routed such that inputs and outputs from each gate are brought out to the 96 contact fingers of the card, allowing all logic interconnections to be made via wire wrap on the connector pins. All of the wires or wire segments (links) associated with a specific logic signal or source are defined as a net. Due to the three level per pin limitation mentioned earlier, a net may have up to three branches with each branch having several wires or segments. A three branch net is shown in

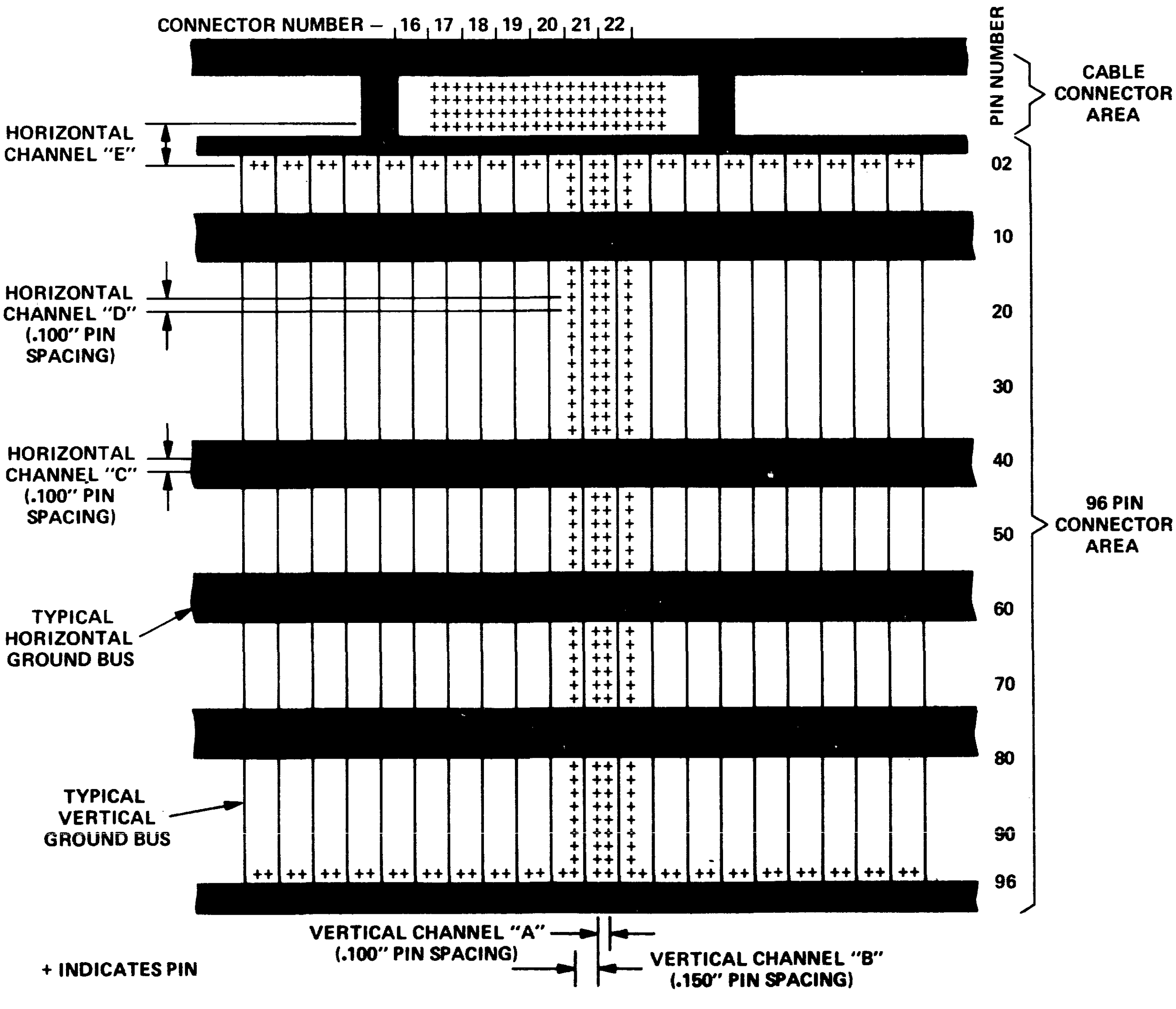

CONNECTOR NUMBER – 16 17 18 19 20 21 22
PIN NUMBER
CABLE CONNECTOR AREA
HORIZONTAL CHANNEL "E"
02
10
HORIZONTAL CHANNEL "D" (.100" PIN SPACING)
20
30
HORIZONTAL CHANNEL "C" (.100" PIN SPACING)
40
50
96 PIN CONNECTOR AREA
TYPICAL HORIZONTAL GROUND BUS
60
70
80
TYPICAL VERTICAL GROUND BUS
90
96
VERTICAL CHANNEL "A" (.100" PIN SPACING)
VERTICAL CHANNEL "B" (.150" PIN SPACING)
+ INDICATES PIN

Figure 1 – Section of MPP-1 backpanel viewed from wire wrap side

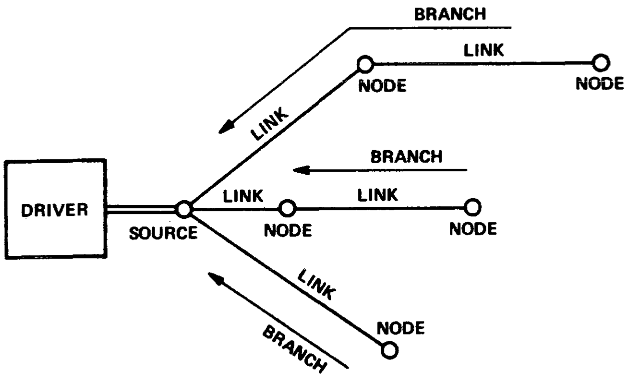


Figure 2—Three branch net

Figure 2. The connection technique illustrated is commonly called "chaining."

Wires as victims/culprits

In general terms, crosstalk is the coupling of an unwanted signal from an active signal carrying line, or "culprit," to a nearby line termed a "victim." For the NAA, only wires adjacent to the victim in the victim channel were considered to be "nearby." If crosstalk is of sufficient magnitude and duration, it will affect gates being driven by the victim line, causing them to switch unintentionally. Potentially, crosstalk can be induced on any of the backpanel wires which comprise a branch and can affect any of the loads tied to that branch. For reasons given in a later section, it is unlikely that crosstalk induced on one branch of a victim will affect loads attached to another branch of that victim. Consequently, each branch was considered a potential victim for noise analysis. The victim example shown in Figure 3 is treated as two separate cases; one having adjacencies with culprits C2, C3 and C4, and another case having adjacencies with culprits C1 and C5. Neither C6 nor C7 would be considered as having any effect on either victim branch.

Before attempting to estimate the crosstalk on a specific branch assumed to be a victim, knowledge of adjacent wires or potential culprits is essential.

LOAP

The Length of Adjacency Process mentioned previously provides complete routing information for any branch selected by the user. It also supplies a list of components which drive wires that lie adjacent to the victim in the victim channel and indicates that the length of each adjacency in inches. These components are the "potential culprits" for the selected victim. To assure that only manageable amounts of data are generated, LOAP information is controlled by two cut-off parameters. One parameter sets a minimum adjacency limit for components in the list. If set at 3″ for example, a component will not be listed as a potential culprit unless it drives a wire that lies adjacent to the victim for 3 or more inches. The second parameter sets a minimum limit for the total adjacency of all the components listed. If this limit is not met, the entire branch is eliminated from the edit. If set at 100″ for example, only "cases" (branches and their potential culprits) would be listed that had component adjacencies which totaled 100 or more inches. A sample page from the LOAP Edit is shown in Figure 4. The minimum adjacency cut-off was set at 3″ for the NAA on the basis of empirical measurements. The total cut-off was set at 135″ based on a sample taken from LOAP data. The 90/30 Processor contains an estimated 10,000 branches, 2000 of which were included in the sample. It was found that total adjacencies for these branches went from five inches or less, to greater than 1200 inches. Figure 5 shows a cumulative percentage chart representing approximately 1500 branches which remained after the 3″ minimum cut-off was imposed on the 2000 branches sampled. Applied to the entire number of branches in the Processor, this indicates that 7500 branches would remain with the three inch cut-off imposed. From the graph, it can be seen that 20 percent of the branches have total adjacencies greater than 135″. Applying this fact to the 7500 remaining Processor branches implies that 1500 cases would have to be analyzed. With the resources available, it was decided to accept the 135″ total cut-off knowing that as time allowed, it could be relaxed to pick up additional cases.

Once the LOAP list of potential culprits associated with a specific victim is available, logic simulation is used to determine which of these components actually switch in a machine environment.

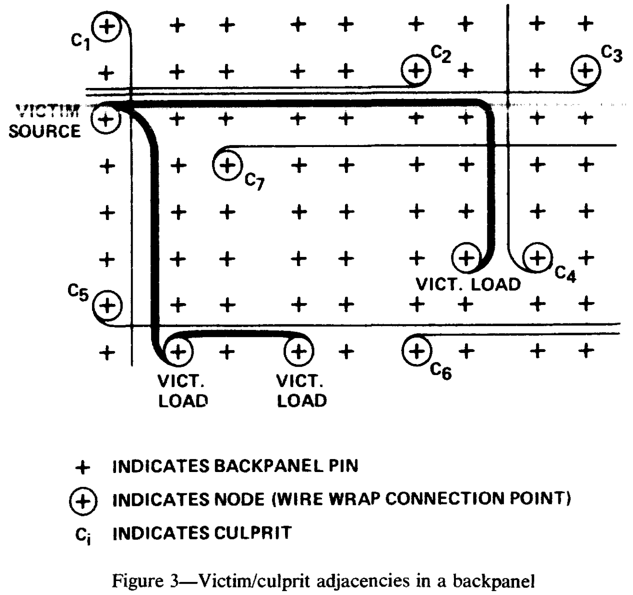
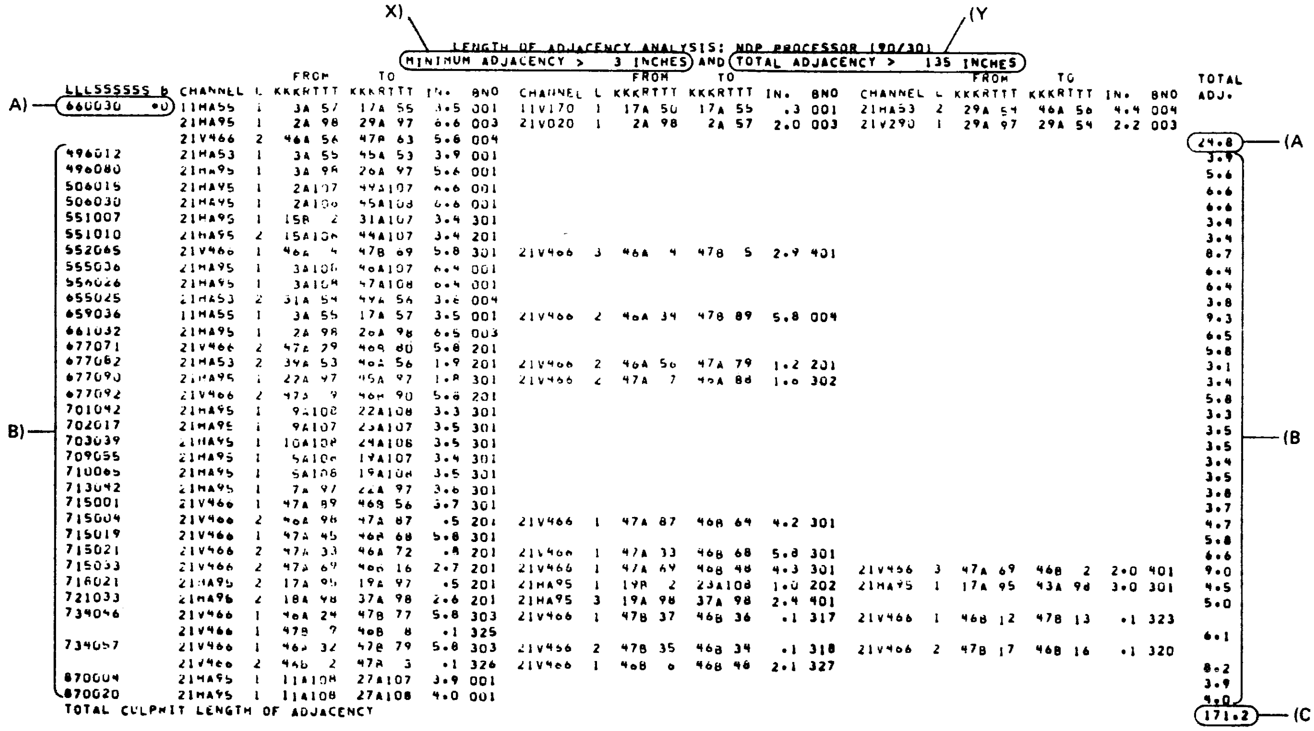


Figure 3—Victim/culprit adjacencies in a backpanel



```
                                      LENGTH OF ADJACENCY ANALYSIS: NDP PROCESSOR (90/30)
                                    MINIMUM ADJACENCY >   3 INCHES AND TOTAL ADJACENCY >   135 INCHES
                        FROM    TO                               FROM    TO                                 FROM    TO                      TOTAL
LLLSSSSSS B CHANNEL L KKKRTTT KKKRTTT IN.  BNO  CHANNEL L KKKRTTT KKKRTTT IN.  BNO  CHANNEL L KKKRTTT KKKRTTT IN.  BNO  ADJ.
660030   .0 11HA55  1   3A 57  17A 55  3.5 001  11V170  1  17A 50  17A 55   .3 001  21HA53  2  29A 54  46A 56  4.4 004
            21HA95  1   2A 98  29A 97  6.6 003  21V020  1   2A 98   2A 57  2.0 003  21V290  1  29A 97  29A 54  2.2 003
            21V466  2  46A 56  47B 63  5.6 004                                                                           24.8
496012      21HA53  1   3A 55  45A 53  3.9 001                                                                            3.9
496080      21HA95  1   3A 98  26A 97  5.6 001                                                                            5.6
506015      21HA95  1   2A107  49A107  6.6 001                                                                            6.6
506030      21HA95  1   2A106  45A108  6.6 001                                                                            6.6
551007      21HA95  1  15B  2  31A107  3.4 301                                                                            3.4
551010      21HA95  2  15A106  44A107  3.4 201                                                                            3.4
552065      21V466  1  46A  4  47B 69  5.8 301  21V466  3  46A  4  47B  5  2.9 401                                        8.7
555036      21HA95  1   3A106  46A107  6.4 001                                                                            6.4
556026      21HA95  1   3A108  47A108  6.4 001                                                                            6.4
655025      21HA53  2  31A 54  49A 56  3.8 004                                                                            3.8
659036      11HA55  1   3A 55  17A 57  3.5 001  21V466  2  46A 34  47B 89  5.8 004                                        9.3
661032      21HA95  1   2A 98  26A 98  6.5 003                                                                            6.5
677071      21V466  2  47A 79  46B 80  5.8 201                                                                            5.8
677062      21HA53  2  39A 53  46A 56  1.9 201  21V466  2  46A 56  47A 79  1.2 201                                        3.1
677090      21HA95  1  22A 97  45A 97  1.8 301  21V466  2  47A  7  46A 88  1.6 302                                        3.4
677092      21V466  2  47A  9  46B 90  5.8 201                                                                            5.8
701042      21HA95  1   9A102  22A108  3.3 301                                                                            3.3
702017      21HA95  1   9A107  23A107  3.5 301                                                                            3.5
703039      21HA95  1  10A108  24A108  3.5 301                                                                            3.5
709055      21HA95  1   5A106  19A107  3.4 301                                                                            3.4
710065      21HA95  1   5A108  19A108  3.5 301                                                                            3.5
713042      21HA95  1   7A 97  22A 97  3.6 301                                                                            3.8
715001      21V466  1  47A 89  46B 56  3.7 301                                                                            3.7
715004      21V466  2  46A 96  47A 87   .5 201  21V466  1  47A 87  46B 64  4.2 301                                        4.7
715019      21V466  1  47A 45  46B 68  5.8 301                                                                            5.8
715021      21V466  2  47A 33  46A 72   .8 201  21V466  1  47A 33  46B 68  5.8 301                                        6.6
715033      21V466  2  47A 69  46B 16  2.7 201  21V466  1  47A 69  46B 48  4.3 301  21V456  3  47A 69  46B  2  2.0 401  9.0
716021      21HA95  2  17A 95  19A 97   .5 201  21HA95  1  19B  2  23A108  1.0 202  21HA95  1  17A 95  43A 98  3.0 301  4.5
721033      21HA96  2  18A 98  37A 98  2.6 201  21HA95  3  19A 98  37A 98  2.4 401                                        5.0
734046      21V466  1  46A 24  47B 77  5.8 303  21V466  1  47B 37  46B 36   .1 317  21V466  1  46B 12  47B 13   .1 323
            21V466  1  47B  7  46B  8   .1 325                                                                            6.1
734057      21V466  1  46A 32  47B 79  5.8 303  21V456  2  47B 35  46B 34   .1 318  21V466  2  47B 17  46B 16   .1 320
            21V466  2  44B  2  47A  3   .1 326  21V466  1  46B  6  46B 46  2.1 327                                        8.2
870004      21HA95  1  11A108  27A107  3.9 001                                                                            3.9
870020      21HA95  1  11A108  27A108  4.0 001                                                                            4.0
TOTAL CULPRIT LENGTH OF ADJACENCY                                                                                       171.2
```

REFERENCE	DESCRIPTION
A	VICTIM IDENTIFICATION AND BRANCH NUMBER, VICTIM ROUTING DATA AND VICTIM WIRE LENGTH IN INCHES.
B	SOURCE IDENTIFICATION AND ROUTING DATA FOR POTENTIAL CULPRITS OF VICTIM. EACH POTENTIAL CULPRIT LIES ADJACENT TO THE VICTIM FOR THE LENGTH SHOWN.
C	TOTAL ADJACENCY OF ALL POTENTIAL CULPRITS.
X	MINIMUM ADJACENCY CUT-OFF WHICH EXCLUDES FROM THE POTENTIAL CULPRIT LIST ANY POTENTIAL CULPRIT THAT IS NOT ADJACENT TO THE VICTIM FOR AT LEAST "X" INCHES. IN THE SAMPLE X=3".
Y	MINIMUM TOTAL ADJACENCY CUT-OFF WHICH EXCLUDES ANY VICTIM FROM THE LOAP EDIT THAT DOES NOT HAVE A TOTAL POTENTIAL CULPRIT ADJACENCY OF AT LEAST "Y" CULPRIT INCHES. IN THE SAMPLE, Y=135".

Figure 4—Sample page from Loap edit

Logic simulation

Model description

The PSD logic simulator exercises a model derived from transcription data stored in a Master Data Bank (MDB). An MDB exists for each of the major functional blocks of the 90/30 System (Processor, Integrated Peripheral Channel, etc.). The parallel nature of data path logic is expected to create the best environment for crosstalk. Since the Processor contains a significant amount of data path logic, it was selected as the main target for NAA application. The approach, however, is applicable to the logic comprising each of the remaining functional blocks.

Storage capability (including main storage, control storage and general registers) for the processor model is provided through the logic simulator via the normal processor logic interfaces. The model storage areas have a capacity equal to that of the actual machine with the exception of main storage which is limited to 512 bytes. The contents of these storage areas is input to the model prior to simulation.

The logic simulator provides timing pulses to the model at intervals selected by the user. The 90/30 System uses a

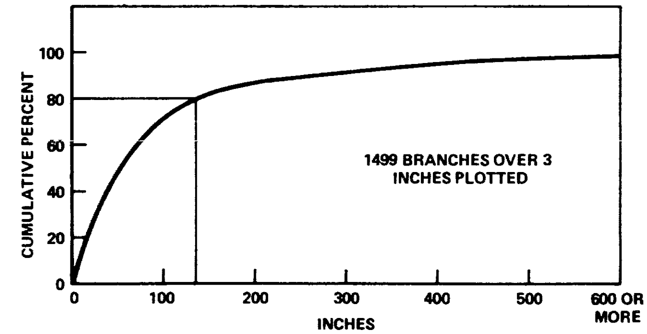


Figure 5—Cumulative distribution of adjacencies for 90/30 processor sample of branches

four phase clock which was simulated by using five intervals or passes. The relationship between these intervals and the 90/30 clock timing is shown in Figure 6. The five passes (four clocks) constitute a cycle.

To exercise a model, initial conditions for storage areas must be specified along with input levels for critical system signals such as System Clear, Run etc.. The model control storage is initialized with the micro-code written to execute the full 90/30 instruction set. Initial main storage conditions are merely the series of macro instructions (program) and operands which are to be simulated. Taken together, these conditions constitute a simulation "run" and the model will respond, cycle for cycle, as an actual machine executing the same program.

The Logic Simulation System allows the selection, sampling, and recording of the activity of any set of logic elements during a simulation run. This set of elements is input to the simulator via the PR File. The run output data, termed "display," contains the value (1 or 0) of each of the selected elements (or logic gates) at each pass during the run. Display data is output onto a tape called the DP File from which a printout can also be obtained. Initial main storage conditions and register contents are specified by the user on punched cards and then transferred to the PR File for input to the logic simulator. These cards can also be read into a 90/30 Processor via a card loader thus providing the machine with the same initial conditions as the simulator.

Simulation of LOAP components

LOAP produces a list of components or logic elements adjacent to a specific victim branch under the two cut-off parameters mentioned previously. With this list as an input to the simulator (via the PR File as elements to be displayed) it is possible to determine at each pass of a program being simulated which of the potential culprits actually switch. The link that enables LOAP Files to be entered onto a PR File for Simulation is a program called DISSEL. A flow diagram illustrating the parts of the NAA described thus far is shown in Figure 7.

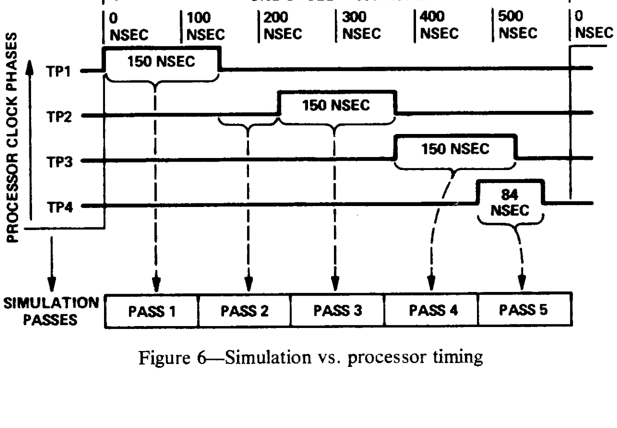


Figure 6—Simulation vs. processor timing

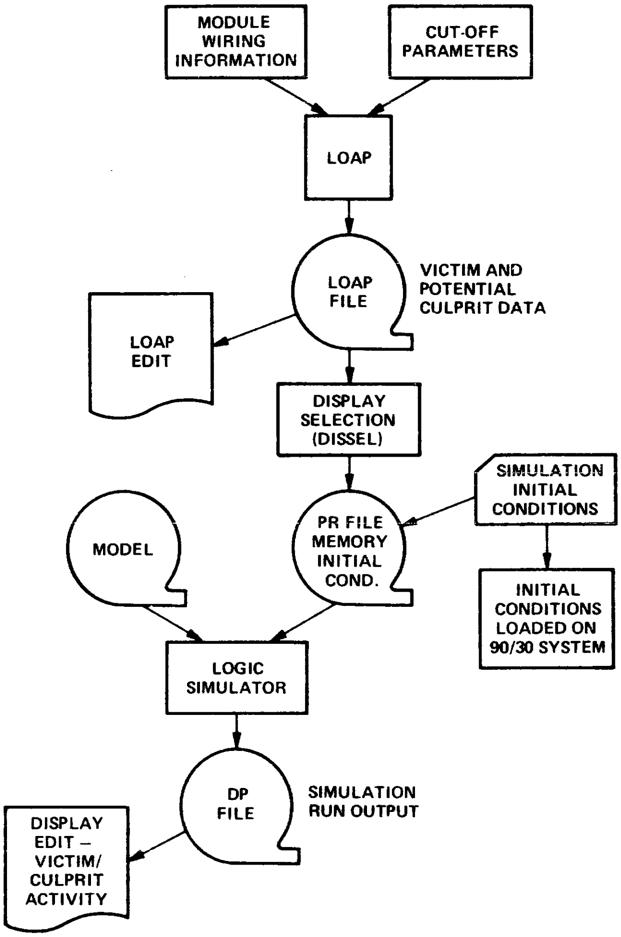

MODULE WIRING INFORMATION
CUT-OFF PARAMETERS
LOAP
LOAP FILE
VICTIM AND POTENTIAL CULPRIT DATA
LOAP EDIT
DISPLAY SELECTION (DISSEL)
SIMULATION INITIAL CONDITIONS
MODEL
PR FILE MEMORY INITIAL COND.
INITIAL CONDITIONS LOADED ON 90/30 SYSTEM
LOGIC SIMULATOR
DP FILE
SIMULATION RUN OUTPUT
DISPLAY EDIT – VICTIM/ CULPRIT ACTIVITY

Figure 7—Loap and simulation elements of NAA

Simulation run

Ideally, the program or simulation run used to identify culprits affecting a branch at each clock phase or simulation pass should be developed to cause as many operand dependent logic elements to switch together as possible, while exercising the entire macro and micro instruction set. A program with this ideal capability was not available for the 90/30 Processor. What was available, however, was a test program developed to check the accuracy of the Processor model following updates. The test program executes 96 percent of the 90/30 macro instruction set and 90 percent of the micro instruction set. When simulated, the program provides the best available sample of actual signal switching activity for crosstalk considerations.

So far, it has been shown how victim/culprit, adjacency and routing data can be obtained from LOAP and how switching activity can be obtained via logic simulation. The next step is to combine these parameters into an estimate of crosstalk using an empirically derived estimating equation.

Crosstalk estimating equation

Test details

As part of the NAA effort, the PSD Hardware Applications Section was requested to make crosstalk measurements on test wires threaded through a wired 90/30 module and to use the data to derive an estimating equation expressed in terms of relevant variables controlled during the tests. The following parameters were controlled during the tests:

1. Adjacency between each culprit and the victim (6, 12 and 18 inches).
2. Number of culprits (1, 2, 4, and 8).
3. The number of culprit loads (1, 2 and 3).
4. The number of victim loads (1, 2 and 3).
5. The victim wire wrap level (levels 1 and 2).
6. The victim channel type (channels identified in Figure 1 as b, c and d were tested).

While not all combinations of the above were tested, approximately 120 crosstalk measurements were made and approximately 90 of these were used as data points in the derivation of the equation. Half of the card complement of the module was in place during the tests, and power was applied to the module to simulate actual system impedance levels. TTL III gates were used as drivers and receivers due to the faster switching times of these circuits as compared to Series 74. For both victims and culprits, quad, 2-input NOR gates were used as drivers while hex-inverter gates were used for receivers. For cases involving more than one culprit, the maximum time span from the first driver switching to the last switching was 1 nsec. Culprit lines were caused to switch through both positive and negative transitions while a measurement was made of the maximum peak positive noise on the victim line. For all tests, the victim line was held in the low state where these gates have the smallest noise margin and are therefore most susceptible to crosstalk noise. Because the low impedance at the victim driver output tends to reduce noise at this point, measurements were taken only at the receive end of the victim line. The low driver impedance also tends to eliminate interaction between branches so that noise induced on one branch does not affect loads attached to another branch driven by the same source.

Equation derivation

Before obtaining an empirical relationship between measured noise and the parameters controlled, the data was normalized to isolate the dependence of crosstalk on each of the parameters. The normalized data was then graphed, and curves were fitted to the data points. The fitted functions were then concatenated to produce a final equation with a multiplying constant selected such that approximately 80 percent of the data points fell on or below the estimating equation. The equation thus tends to predict higher levels of crosstalk than would actually be found, a bias that was chosen throughout the derivation and in later applications. This bias is reflected in the use of TTL III drivers for initial measurements, for example, knowing that the results would later be applied to Series 74 gates which would, because of their slower switching time, produce crosstalk lower than the predicted value. The final derived equation is shown in Figure 8. It should be noted that the six basic terms (or functions) in the equation were chosen from a basis of fit and have no theoretical interpretation.

Application

Each of the six functional terms will be briefly reviewed to indicate how a computation would be made. The first of the six functional terms indicates the relationship between crosstalk and the number of culprits that switched (N). By comparing successive passes of simulation data for a specific case (branch and its associated potential culprits) the number of culprits can be determined and this becomes the value used for "N" in the crosstalk equation. The second term shows the influence of culprit loading on crosstalk. The AC load value at a node (see Figure 2) represents the amount of transient loading presented by that node to the source component during switching. If the number of AC loads driven by each culprit is known, then Lc is the average load for all these culprits (i.e., the sum of individual culprit loading divided by N). Loading data is available for each of the potential culprits from LOAP. The third term indicates the contribution of victim loading to crosstalk and also shows that it is dependent on culprit loading. Empirical measurements were made with both victim and culprit loads clustered at the end of the respective lines. It was found that distributing culprit loads did not significantly alter the crosstalk on the victim and therefore the culprit loading was merely averaged. On the victim line, however, distributing the loads changed the crosstalk, which measured greatest at the victim node with the smallest load. On the basis of these results, it was decided to use the AC load found at the smallest loaded victim node for the value of Lv. This value is also obtained from LOAP data. The fourth term of the estimating equation indicates

$$CT = \underbrace{169}_{\substack{\text{MULTIPLYING}\\ \text{CONSTANT}}} \underbrace{\left[4.5\,(1-e^{-N/4})\right]}_{\substack{1\\ \text{CULPRIT}\\ \text{TERM}}} \underbrace{\left[.8+.2L_c\right]}_{\substack{2\\ \text{CULPRIT}\\ \text{LOAD}\\ \text{TERM}}} \underbrace{\left[.9+.1\left(\frac{L_c}{L_v}\right)\right]}_{\substack{3\\ \text{VICTIM}\\ \text{LOAD}\\ \text{TERM}}} \underbrace{\left[.6+.4C_h\right]}_{\substack{4\\ \text{CHANNEL}\\ \text{TERM}}}$$

$$\underbrace{\left[W(1.3-.3C_h)\right]}_{\substack{5\\ \text{WRAP LEVEL}\\ \text{TERM}}} \underbrace{\left[1.19\,(1-e^{-A/6})\right]}_{\substack{6\\ \text{ADJACENCY}\\ \text{TERM}}}$$

Figure 8—Crosstalk estimating equation

the relationship between victim channel type and crosstalk. A complete description of victim routing is available from LOAP and must be translated into the basic channel types illustrated in Figure 1. Channel types A through E were assigned values of 2.5, 2, 1, 3 and 1 respectively. For a victim routed in more than one channel, a weighted channel type is determined based on the portion of total branch length that lies in each of the defined channel types. This weighted value is used for Ch in the equation. The fifth functional term shows the effect of wire wrap level on crosstalk and shows that it is dependent on channel type. For single wire branches, the value used to represent wrap level (W) in the equation is the wire wrap level itself (i.e., 1, 2, or 3). For most branches, however, a weighted level must be calculated based on the portion of the branch at each level. This is the value of W used in the estimating equation and is obtained from the victim routing description on the LOAP output tape. The final term in the equation indicates the relationship between adjacency and crosstalk. From the LOAP tape, the adjacency between each culprit and the victim branch can be obtained. The value of "A" in the equation is the average of these adjacencies (i.e., sum of individual adjacencies divided by N).

Three elements of the NAA have been described thus far; LOAP, logic simulation, and the estimating equation. The final element is the Noise Analysis Program (NAP) which uses the above three elements to assign one value of crosstalk to each branch and then ranks the branches accordingly.

Noise analysis program

The NAP considers one case at a time from the LOAP tape. It computes a crosstalk estimate for that case for successive passes of simulation data starting at the beginning of the simulation run. A computation is only made where simulation display data indicates that the victim is low. It was felt necessary to separately maintain the maximum value of calculated crosstalk for each of the five simulation passes. While the crosstalk at a given pass may be high enough to affect logic gates attached to the victim branch, the output of these gates may not be sampled until another pass where there is very little crosstalk. It is the function of the cognizant logic design personnel to establish which pass or passes are noise critical for a specific victim branch. If, for example, a comparison between pass one and pass two of a given cycle shows that five potential culprits have switched, a computation would be made based on these five culprits and the result would be interpreted as a pass two calculation. Each time a new computation is made, it is compared with the previous value for that pass and only the largest is kept. After the entire simulation has been exhausted, pass by pass, the maximum values (one for each pass) are assigned to the branch being analyzed along with the simulation cycle and pass where each maximum was computed.

VICTIM/BRANCH – 600001/2

DATA DESCRIPTION	PASS 1	PASS 2	PASS 3	PASS 4	PASS 5
CROSSTALK ESTIMATE (MV.)	2250.00	500.00	1600.00	2650.00	2300.00
SIMULATION CYCLE	819	29	163	1162	720

Figure 9—NAP results—Single case

Figure 9 shows a sample of the final results for a single branch. Note that the values of predicted crosstalk are much higher than would normally be expected due to the "predict high" bias mentioned earlier. A table is available from the Simulation Process to translate the cycle specified into the macro- and micro-instruction being executed at that cycle. The crosstalk shown is the maximum value calculated for the specific branch and was computed based on switching activity which occurred at the simulation cycle and pass indicated.

After evaluating one branch, the NAP proceeds with each of the remaining branches in a similar manner. When this is complete, the branches are ranked as shown in Figure 10. Five rankings are shown, one for each pass, and each victim branch will appear only once within a ranking for a given pass. A fixed number of branches can be listed for each pass (e.g., top 200 as shown) or only those branches which surpass a minimum crosstalk level can be listed (e.g., all branches over 2000.00 millivolts.) Also, a single average ranking can be produced in which branches are ranked according to an average crosstalk value computed from the 5 separate pass values of each branch. Still another single listing can be produced which contains the same information as in Figure 10, but is sequenced according to cycle numbers. This listing would be helpful when trying to validate estimates with actual measurements.

NAA RESULTS

When applied to the entire 90/30 Processor file using the cut-off parameters, the NAP produced a total of 1186 victim

INDEX	DATA DESCRIPTION	PASS 1	PASS 2	PASS 3	PASS 4	PASS 5
1	VICTIM/BR. ESTIMATE CYCLE	714001/2 700.00 214	900001/3 0000.00 516	622007/0 300.00 216	492001/3 2000.00 100	829050/3 1800.00 350
2	VICTIM/BR. ESTIMATE CYCLE	500010/0 1500.00 743	832001/0 350.00 1196	500010/0 800.00 19	600001/2 2200.00 67	900001/2 2100.00 290
3	VICTIM/BR. ESTIMATE CYCLE	829050/3 1650.00 10	714001/2 400.00 1243	900001/2 1000.00 65	900001/3 2300.00 85	600001/2 2300.00 720
	• • •					
199	VICTIM/BR. ESTIMATE CYCLE	429017/0 2100.00 512	722010/3 700.00 29	900001/3 1500.00 699	722010/3 2700.00 860	714001/2 2650.00 35
200	VICTIM/BR. ESTIMATE CYCLE	600001/2 2250.00 819	580060/0 800.00 819	600001/2 1600.00 163	714001/2 2800.00 600	492001/3 3100.00 1000

Figure 10—Sample NAP ranking

branches. The distribution of these victims for each pass is shown in Figure 11. Before actually measuring the crosstalk of a sample of these victims in the machine, a means of assuring accurate measurements is required.

Provision for system measurements

As mentioned earlier, the initial main storage/register contents simulated for the NAA can be loaded into a 90/30 System via a card loader. The machine response to the program (initial conditions) matches, cycle for cycle, the results of the simulation. The cycle where the maximum crosstalk was predicted for each victim is available from NAA results. To make crosstalk measurements on the victim branch, a triggering unit was designed and built to provide a flexible means of triggering a scope at or before the specified cycle. The unit is initialized by a spare bit set into a micro-word accessed prior to the measurement cycle. Switches allow the user to trigger a scope by entering the number of cycles between the micro-word containing the spare bit and the measurement cycle. Provisions have been made to eliminate multiple triggering which would occur if the micro-word containing the spare bit were accessed several times during the program. Switches allow the user to select any one of these accesses to initialize the triggering unit and blank out the spare bits from the remaining accesses. With this unit, the user can obtain a reliable scope sync at any program cycle and observe culprit switching and crosstalk.

NAA validation

Since the 1.95 volt and above class still contained a substantial number of victim cases (416) with high predicted

CROSSTALK CLASS INTERVAL (VOLTS)	PASS 1 # OF VICTIMS	PASS 2 # OF VICTIMS	PASS 3 # OF VICTIMS	PASS 4 # OF VICTIMS	PASS 5 # OF VICTIMS	TOTALS	CUMULA-TIVE
3.4 - 3.5	1	0	2	0	0	3	3
3.2 - 3.3	2	0	4	0	0	6	9
3.0 - 3.1	5	2	2	0	1	10	19
2.8 - 2.9	11	4	7	0	2	24	43
2.6 - 2.7	19	12	11	0	4	46	89
2.4 - 2.5	15	5	18	2	7	47	136
2.2 - 2.3	47	11	26	6	18	108	244
2.0 - 2.1	66	16	49	9	32	172	416
1.8 - 1.9	85	20	61	8	38	212	628
1.6 - 1.7	120	30	100	18	69	337	965
1.4 - 1.5	112	55	82	37	63	349	1314
1.2 - 1.3	103	94	84	83	87	451	1765
1.0 - 1.1	105	90	101	105	86	487	2252
0.8 - 0.9	74	136	126	144	98	578	2830
0.6 - 0.7	77	133	82	150	130	572	3402
0.4 - 0.5	54	163	99	145	137	598	4000
0.2 - 0.3	74	131	87	152	139	583	4583
0.0 - 0.1	17	47	28	89	45	226	4809
TOTALS	987	949	969	948	956	4809	4809

Figure 11—Distribution of predicted crosstalk

crosstalk, an attempt was made to measure a significant number of these cases. A total of 131 crosstalk measurements were made under the conditions specified through NAP. The distribution of measured values is shown in Figure 12. Cases with measured crosstalk below 400 millivolts were not investigated because they fell within the guaranteed d-c noise margin of the logic family. Although only 400 millivolts is guaranteed, a TTL gate typically exhibits a d-c noise margin in excess of 1.0 volt. Typical a-c noise immunity for low-nanosecond region pulse widths (10-15 nsec) is even higher. For these reasons, cases exhibiting crosstalk below 700 millivolts were not investigated. Each of the remaining 37 cases were investigated to determine whether victim receivers were sensitive to the noise during the pass in which the noise occurred. Results indicated that in all cases the noise was present during a clock interval in which the victim receivers were not susceptible. Consequently, no re-wiring of these cases was required.

ACCURACY OF NAA ESTIMATES

Several factors limit the accuracy of the NAA estimates, and to that extent must be considered when applying and interpreting the results. First, the estimating equation was derived from a limited number of points taken from measurements made under controlled conditions. Its application to backpanel branches involved weighing certain parameters and, in some cases, using the equation slightly beyond the range for which it was developed. It is unreasonable to expect the equation to provide accurate predictions in such cases. Second, in the backpanel, all culprits were considered equally close to the victim, even though the number of wires (potential culprits) in most channels makes this a physical impossibility. This consideration tends to bias the predicted results high compared with actual measurements. A more severe problem may be caused from a victim located along the side of a channel (adjacent to the backpanel pins). The possibility exists that wires in the adjacent channel (separated from the victim by .025″ square pins) could induce noise on the victim. It was felt that the effect of these wires would be secondary in nature, and no provisions were made to include them in the current version of the NAA. It would, however, be possible to include the effect of these wires in the analysis. A third limitation to the accuracy of the NAA estimates is the fact that all levels of logic in a given logic net were assumed (via the logic simulation data) to switch together within a clock phase, a condition which reflects the method used for logic simulation. In the actual machine, each gate is a net switches one circuit delay after the previous gate. With the wide range of circuit propagation times for the basic logic gates (between 3nsec. and 15nsec.) used in the 90/30 System, it was felt more desirable to accept the high bias, which results from assuming zero delay between gates, as an estimate of worst case switching conditions. Finally, the estimates are limited by the fact that the contribution of cable crosstalk was not considered in the ranking of branches which drive cables. There are several methods of including the effect of cabling on these branches, but the time/resource schedule of the NAA prohibited implementation. Estimates for branches which drive cables are made only based on the adjacencies, loading, etc., found in the source and destination motherboards.

Measured Crosstalk Class (mV)	Number of Occurrences	Cumulative	%
0- 199	19	19	14
200- 399	41	60	46
400- 599	27	87	66
600- 799	15	102	78
800- 999	13	115	88
1000-1199	7	122	93
1200-1399	6	128	98
1400-1599	2	130	99
1600-1799	0	130	99
1800-1999	1	131	100
2000-2199	0	131	100
2200-2399	0	131	100
TOTALS	131		100

Figure 12—Distribution of measured crosstalk

For all of the reasons described in this section, the estimates or values assigned to each branch by NAA are not accurate estimates of absolute crosstalk magnitude. When choices were available, the predictions were always biased high. The values are, however, valid relative to one another, and can be used to produce ranking mentioned earlier.

CONCLUSION

NAA is a unique approach combining adjacency data, switching coincidence data and crosstalk measurement data into a single crosstalk analysis tool. Useful in locating potential problem areas during the development cycle, NAA can also be used to evaluate the effects of test changes intended to remedy crosstalk problems. By providing a ranking of backpanel branches based on their estimated susceptibility to crosstalk, NAA directs the user to specific backpanel branches where high crosstalk is expected and where crosstalk measurements should be made. For each branch, it specifies the point in the operating program (cycle and clock phase) where the worst crosstalk condition is expected. With modification, NAA can be used for various packaging techniques. While originally developed for wire wrap backpanels, the approach should provide more accurate results when applied to packaging techniques with controlled conductor spacing.

ACKNOWLEDGMENTS

For contributions to the NAA effort, the authors wish to thank the following individuals:

E. J. Crossen
C. J. Hammond
M. A. Hebhardt
H. A. Nidecker
A. B. Pataki
A. T. Wissink

True liquid cooling of computers

by E. A. WILSON

Honeywell Information Systems, Inc.
Phoenix, Arizona

ABSTRACT

True Liquid Cooling means removing the heat from the heat source with a liquid instead of just pre-chilling air with cold water. One of the major advantages of True Liquid Cooling over pre-chilling air is that the heat may be rejected from the internal liquid into either chilled water or computer room air, at the option of the customer.

The basic problem, which must be solved in order to achieve such a cooling system, is to get the liquid to and from the heat source without getting everything else wet. This paper describes how this is accomplished for both multichip integrated circuit packages and high wattage power supplies. Also, described is the implementation of a True Liquid Cooling system for a large central processor.

INTRODUCTION

The heat rejection problem in Honeywell's new top of the line Model 66/85 has been handled with TLC (true liquid cooling). This cooling system qualifies for the status of true liquid cooling because the liquid is taken to the heat sources (integrated circuits and power supplies) instead of the liquid being used to pre-chill air which is then used to cool the heat sources (quasi-liquid cooling or QLC). In this paper, the full cooling system from the novel SLIC (silent liquid integral cooler) to the PERU (pump, exchanger, reservoir unit) and COOL (cools off our liquid) will be presented.

Before addressing the system as a whole, it is worthwhile to show the inherent advantage of liquid (in this case, water) over air as a cooling fluid. The Honeywell micropackaging technology not only has made TLC practical but has also made it necessary. The micropackage shown in Figure 1 has a pattern which may accommodate 76 chips. Different chip placement patterns are used depending on the mix of SSI, MSI, and LSI chips. The number of chips mounted in an 80 mm by 80 mm micropackage is equivalent to half of a 300 mm by 300 mm printed circuit board with the result of increasing the planar power density by approximately a factor of 10 (since more high power chips are used in the micropackage than on the board). With this increase in planar power density, came the problem of how to cool the micropackage coupled with the goals to have a quiet system and to not force low packaging density in the mainframe cabinets by using up space with large blowers and air ducts.

The solution of "more air" would violate the second of the aforementioned goals and the solution of many small, local fans or blowers would violate the first goal. The solution required a fundamental change; the coolant itself. The simplified analysis below shows why such a change provides the easiest solution.

The coefficient of convection (h) of a fluid flowing over an object is obtainable from the dimensionless grouping of the Nusselt, Reynolds, and Prandtl numbers:

$$Nu=(Re)^n(Pr)^m$$
$$hL/k=(VL\rho/\mu)^n(c\mu/k)^m$$

where

h=coefficient of convection,
L=significant length of the body,
k=thermal conductivity of the fluid,
V=velocity,
ρ =density of the fluid,
μ=viscosity of the fluid,
c=specific heat of the fluid,
m and n range from .3 to .8 depending on the body and type of flow.

Since the thermal resistance between the heat source and the coolant is h times the surface area (A), and the significant length is not dependent on the type of coolant, we may write (letting m=n=.5 for the sake of this simple comparison):

$$\text{Resistance}\propto A\sqrt{Vk\,\rho c}$$

Now consider what happens if the planar power density has increased by a factor of 10. The only two alternatives would be to increase A by adding a finned heat sink or increase the velocity by a factor of 100. Obviously the increase in velocity will not yield a quiet system. The addition of a finned heat sink to the micropackage was considered but rejected because of the negative impact on replacement of the chips in the micropackage and the fact that large air ducts had a negative impact on cabinet packaging density and maintainability access.

If we consider the term $\sqrt{k\,\rho c}$ to be a representative

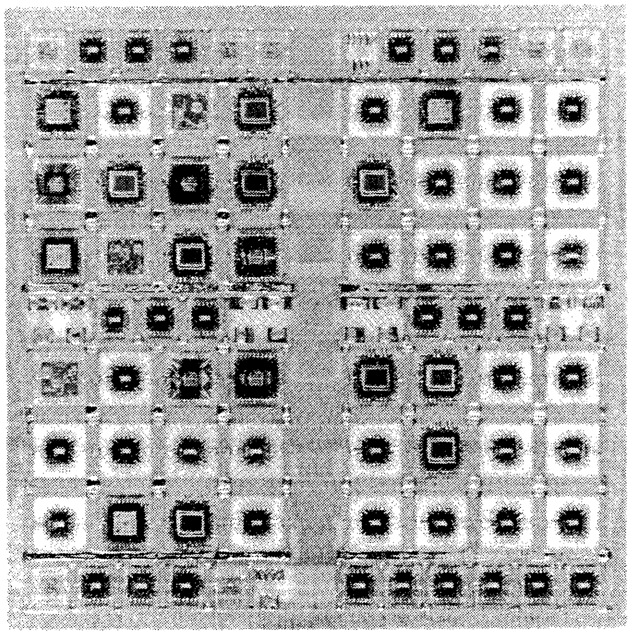


Figure 1—The micropackage pictured above may contain 76 integrated circuit chips plus decoupling capacitors. The 80 mm by 80 mm alumina substrate has several thick film signal and insulation layers screened and fired on it to form the equivalent of an ultra high density printed circuit board. This type of packaging is applied to SSI, MSI, LSI and mixes of chips.

figure of merit for various fluids, then we can easily substitute values for air and water into the term and find that air yields .051 and water yields 4.5 or about 100 times better. The obvious practical conclusion is to use the micropackage area, low velocity, and water instead of air. The problem which had to be overcome in order to take advantage of the high coefficient of convection for water was how to get the water to the heat source. The solution to this problem is explained later in this paper, but first, in order to better appreciate the need for such a solution, it is important to understand the difference in systems with true liquid cooling and quasi-liquid cooling.

TLC VS QLC

The most common application involving liquid cooling of computers (quasi-liquid cooling), as mentioned briefly in the introduction, employs air convection between the heat sources and water cooled heat sinks. In such a system, chilled water from some sort of refrigeration source is used to cool the internal liquid which circulates in a closed loop. The internal liquid in turn pre-chills the air which then cools the heat sources. The inherent problems should be obvious:

1. the relatively low h between the heat source and the air represents a significant thermal resistance;
2. the relatively low h between the air and the water cooled heat sink likewise adds to the thermal resistance;
3. the internal water must be colder than the air, hence leading to a critical condensation control problem (no user wants a rain storm in his computer);
4. the customer supplied chilled water usually must have a very low temperature.

Figure 2 represents the impact of such a system on the computer room's air conditioning and chilled water requirements.

True liquid cooling as used in the Honeywell Model 66/85, on the other hand, eliminates the air exchanges between the heat source and the internal water by using a conduction path between the back (opposite of the chip side) of the micropackage and the SLIC (which is described in detail later). With the first and second problems listed above being non-existent with TLC, problem 3 is solved by operating with a water temperature which is above the dew point of the ambient air, or even above the temperature of the ambient air. Problem 4 is significantly relaxed by problems 1 and 2 not existing.

If the computer user does supply chilled water into the computer room, the heat is rejected from the internal water to the chilled water through a compact water-to-water heat exchanger in the PERU. The internal water temperature is easily controlled by regulating the chilled water with a three-way valve and bypass system built into the PERU. This permits the chilled water temperature to range from

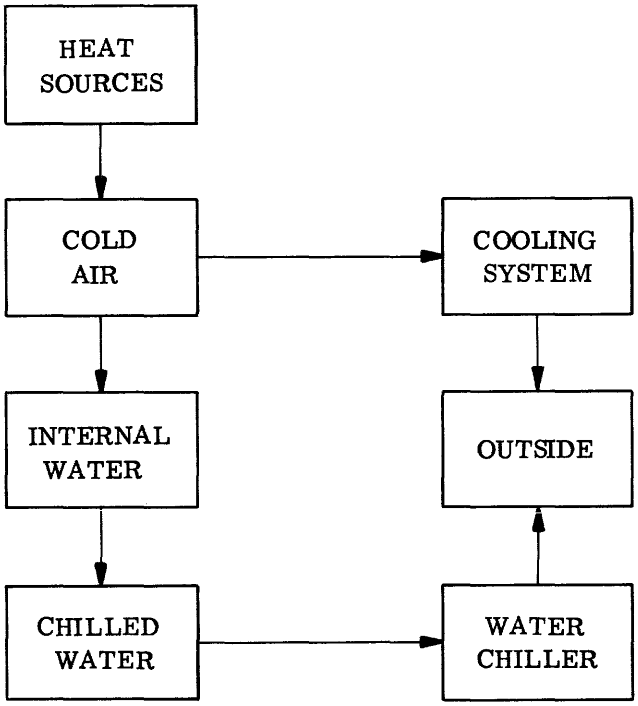


Figure 2—Heat flow with quasi-liquid cooling showing the requirements of the computer room cooling system

0°C (as long as it is not ice) to 20°C, yet the internal water is maintained at 32°C. This system is represented in Figure 3. The economic benefits of such a system should not be overlooked in these days of increasing utility costs. The power consumed by the computer itself is only part of the utility bill. The heat generated must be removed from the room and depending on climatic conditions, this may cost as much as $.53 for every $1.00 spent in operating the computer if conventional air cooling is used. If chilled water is cooled by a water tower and provided to the PERU, this cooling cost may be reduced to $.14 for every $1.00 used by the processor. If the user does not have a water tower, even a standard refrigeration type water chiller can reduce the $.53 to $.43.

If the computer user does not supply chilled water, this TLC system is still feasible because the relatively high internal water temperature permits an option which is not possible with quasi-liquid cooled systems. With this option, the water-to-water heat exchanger is omitted frpm the PERU and the COOL is placed next to it. The COOL unit is nothing more than a simple cabinet containing a large low speed blower and two large water-to-air heat exchangers (coil and fin units) in parallel. The floor space required for the COOL is the same size as for the PERU, only 760 mm by 760 mm.

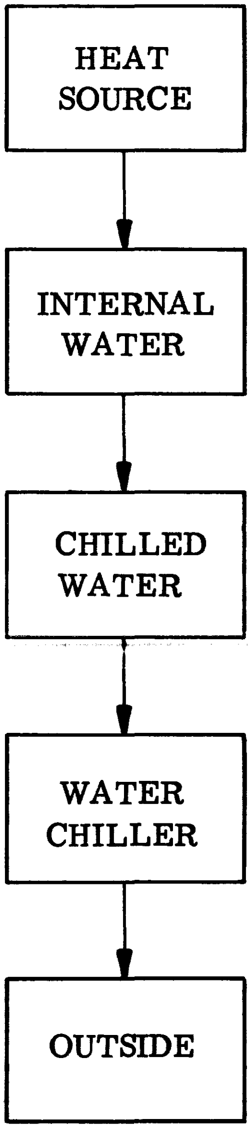


Figure 3—Heat flow with true liquid cooling when chilled water is available

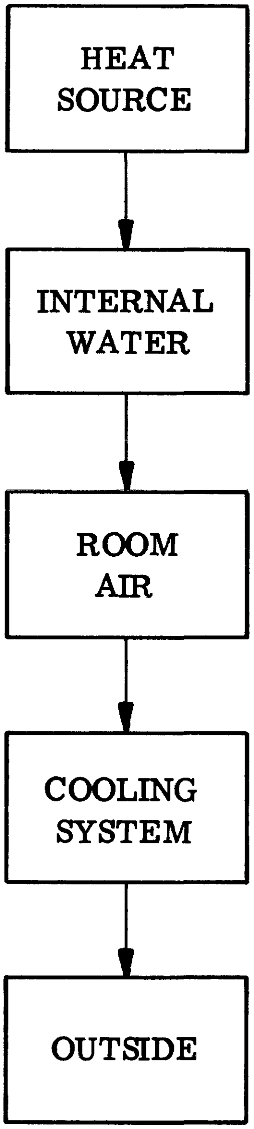


Figure 4—Heat flow with true liquid cooling when chilled water is not available and the option of heat rejection into room air is used

An obvious question is how can this arrangement be quieter than ordinary air cooling? The answer is twofold. First, for standardized manufacturing considerations, an ordinary air cooled system would have to be designed on the basis of the worst (or hottest) micropackage, although many of the micropackages would dissipate much less heat.

Therefore, the heat rejected through the COOL unit represents an averaging of all of the micropackage powers. Second, the water to air heat exchangers have a large surface area due to many fins, hence the total air flow can be low and the air management task is very simple. Because almost all of the air pressure drop is through the fins (virtually no plenum or duct losses), 4000 CFM can be moved with a low speed blower. The result is an 8°C difference between the internal water and the ambient air. Since the cooling system is designed to allow the water temperature to be as high as 40°C, it is obvious that even abnormally warm computer room air can adequately cool the internal water. Figure 4 represents this variation of the TLC system which is possible only because the intermediate air exchanges required in QLC are eliminated.

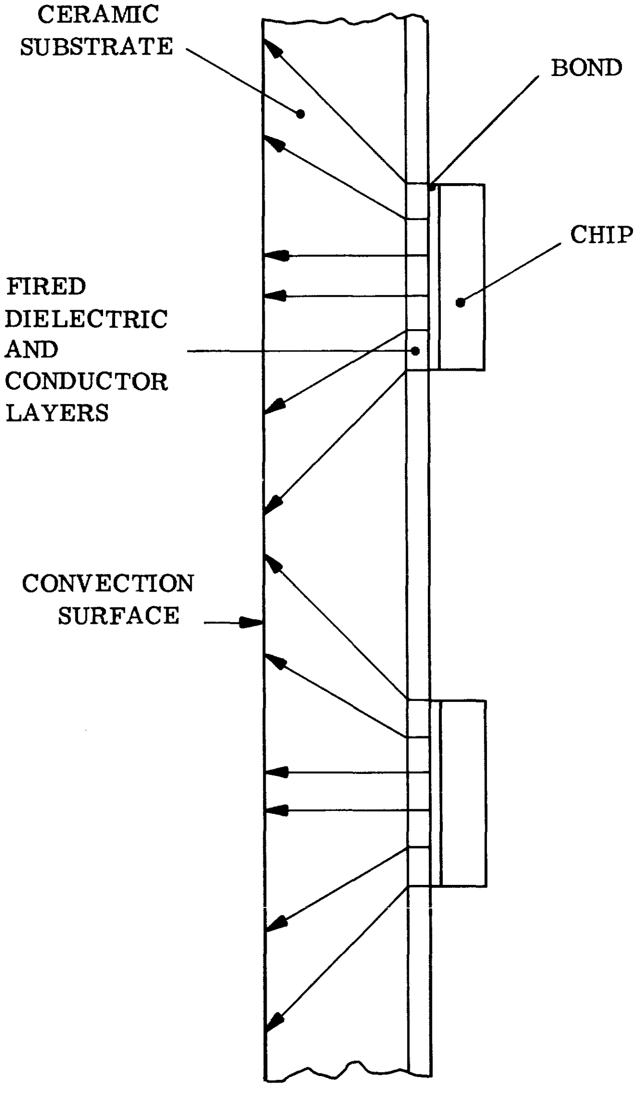


Figure 5—Representation of heat flow in a multichip micropackage

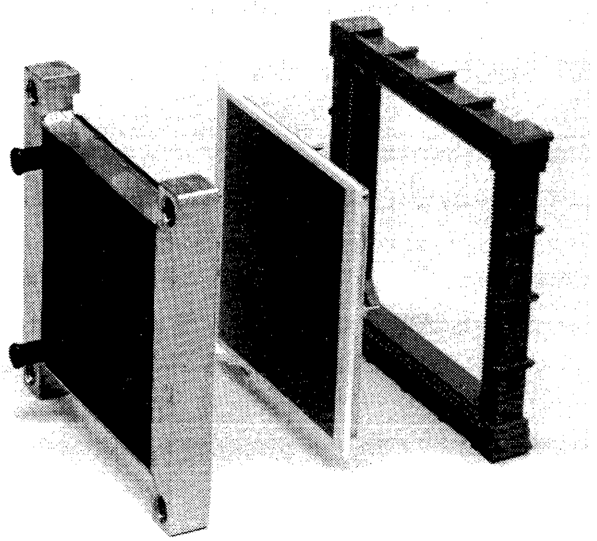


Figure 6—Explosion of the SLIC, micropackage, and connector. The hold down ring is bonded to the SLIC, and in the installed position, the assembly presses the micropackage into the connector and the connector against a printed circuit board which serves as the interconnection vehicle between micropackages

A SLIC WAY OF COOLING MICROPACKAGES

The silent liquid integral cooler (SLIC) is basically a simple chill plate with one important feature, the cooling surface automatically conforms to the shape of the micropackage and requires no oil, grease, or any other "filler" material to provide good thermal contact.

The thermal path in the micropackage is represented in Figure 5. The region populated by chips is 65 mm by 65 mm and the total dissipation could be as high as 60 Watts. An exploded view of the operational configuration of the SLIC and micropackage is shown in Figure 6. The hold down ring is required to hold the micropackage into the connector no matter what cooling method is used. For the sake of manufacturing reasons, the SLIC and hold down ring are integrated into one common assembly. Although the back of the micropackage is smooth, it is not flat as a result of the thick film fabrication process. The copper diaphragm on the SLIC, when pressurized with as little as 3.5 kPa (1 psi=6.895 kilo-Pascals), conforms to the slightly concave surface of the micropackage. The molded plastic back of the SLIC has baffles which produce uniform flow coverage in the chamber. The unassembled parts are shown in Figure 7.

Accessability for micropackage replacement is not impaired by the SLIC because the "plumbing" connections are made by flexible hoses, hence a natural hinge exists as shown in Figure 8. In fact, the replacement of a micropackage can be performed while the water is still flowing. This is possible because of the conservative design of the assembly. The maximum pressure which a SLIC can see in

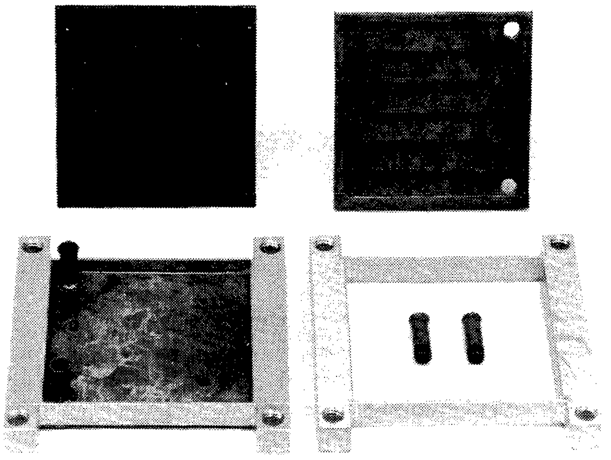


Figure 7—Top left is the blacked (for better bonding characteristics) copper diaphragm. Top right is the inside of the molded plastic back. Bottom right are the hold down ring and the two copper nipples. Bottom left is the completed assembly

service is only 35 kPa yet the SLIC assembly can withstand 700 kPa in the installed position and 200 kPa in the open position.

The thermal resistance between the micropackage and the water (which includes the surface contact resistance between the copper diaphragm and the micropackage as well as the convective resistance between the diaphragm and the water) is shown in Figure 9. At the flow design point of 1 litre per minute, the resistance is only .123°C/W (where the power basis is the total micropackage power, not an individual chip power). This value of thermal resistance is less than half of what is needed to keep the chip junction temperature to less than 85°C even with the worst case combinations of chips, micropackage locations, and internal water temperature (at PERU) of 40°C.

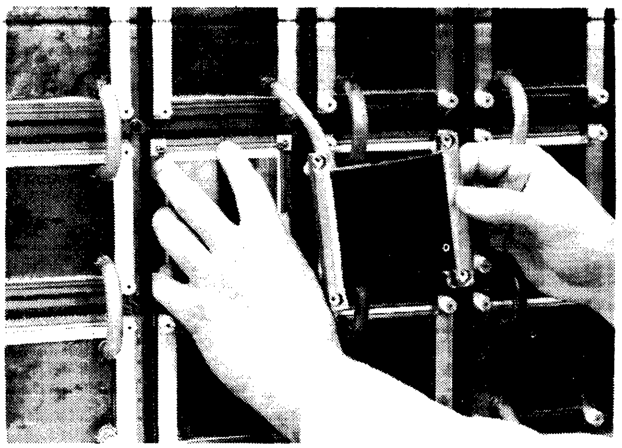


Figure 8—Micropackage removal without disconnecting any of the plumbing. The water may continue to flow during this operation.

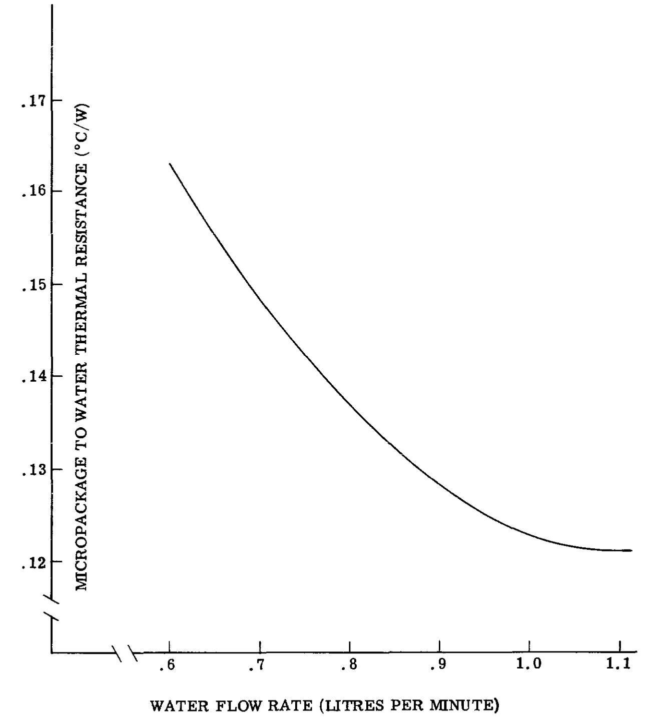


Figure 9—Thermal performance of a SLIC

The utilization of SLIC's in the total system will be covered after looking at the cooling of the power supplies.

SUPER SLIC'S FOR POWER SUPPLIES

Since the internal water is available in the cabinet, advantage is taken of this to cool the power supplies as well as the micropackages. This significantly reduces the air flow requirements and hence the noise level. Over 90% of the heat rejected by the power supplies is now removed by water cooling and this allowed replacing a double squirrel cage blower by two small propeller fans which provide flushing air for a few resistors, capacitors, etc.

A chill plate concept similar to the SLIC is used and hence it was dubbed the super SLIC. The basic electrical design required that the diodes be mounted on a common electrical bus which is easily achieved by using a rectangular aluminum plate. This same plate then is used to hold the reactors and SCR's which are mounted on smaller aluminum blocks and electrically isolated from the main plate by a thin layer of insulation (which is still adequately thermally conductive). Two of the plates with their mounted components form the "bread" of a sandwich structure. The need for grinding the surfaces of the aluminum flat is eliminated by using the super SLIC. This item consists of a rectangular frame with copper sheets soldered on the sides. Two edges

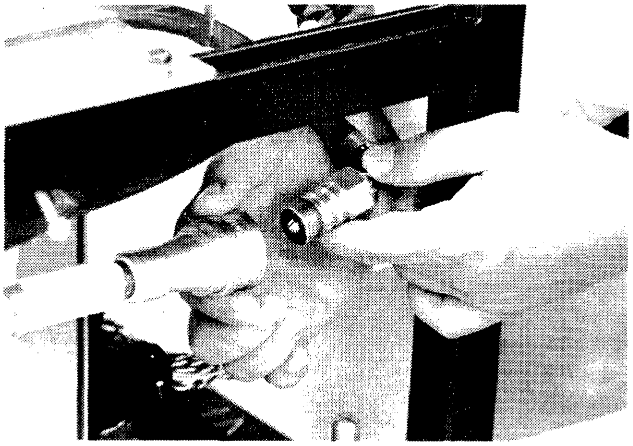


Figure 10—The pwer supplies are connected to the cabinet plumbing via automatic shut-off quick disconnects which permits replacement without shutting down the cooling system

of the frame are copper tubes with strategically placed holes to form inlet and outlet manifolds. The copper sheets flex outward when pressurized and provide the conformability required by the assembled sandwich. The super SLIC is electrically isolated from the aluminum plates by thin sheets of mylar. This yields a sandwich in which the super SLIC is the "meat" and the mylar is the "cheese."

The power supply is connected to the cabinet plumbing by flexible hoses and automatic shut-off quick disconnects (Figure 10). Hence, replacement of a power supply can be performed without shutting down the cooling to the rest of the cabinet.

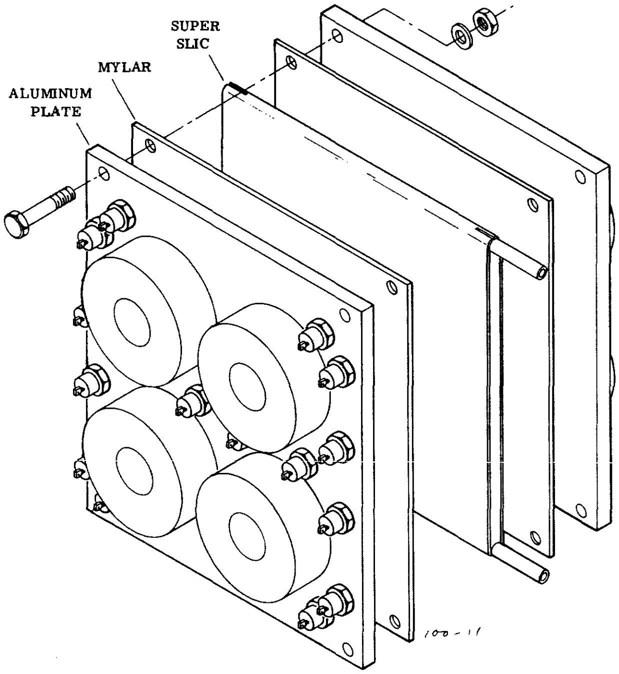


Figure 11—Representation of the power supply cooling elements

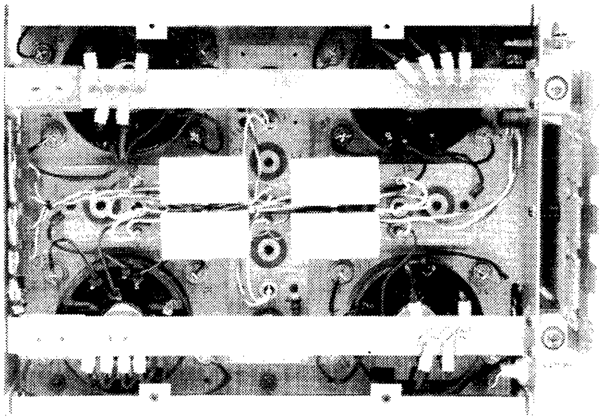


Figure 12—In this view of the power module portion of a 3000 Watt power supply are the diodes, SCR's, and reacters mounted on one of the aluminum plates of the cooling sandwich

The cooling efficiency of the super SLIC sandwich is so high that a 3000 Watt power supply can be cooled with a water flow rate of only 2 litres per minute. The pressure required to flex the sheets is less than 7 kPa but the actual operating pressure is 50 kPa because of the location of the power supplies in the cabinet. Even the 50 kPa pressure is less than 10% of the test pressure during production testing of every super SLIC.

Figure 11 is an exploded representation of the parts of a sandwich and Figure 12 shows the power module portion of a 3000 Watt supply.

SYSTEM IMPLEMENTATION AND OPERATION

The two preceding sections explained how the heat gets from the sources into the water. The block diagram of Figure 3 showed the overall approach to heat movement. This section describes how Figure 3 is implemented and how the SLIC's fit into the system.

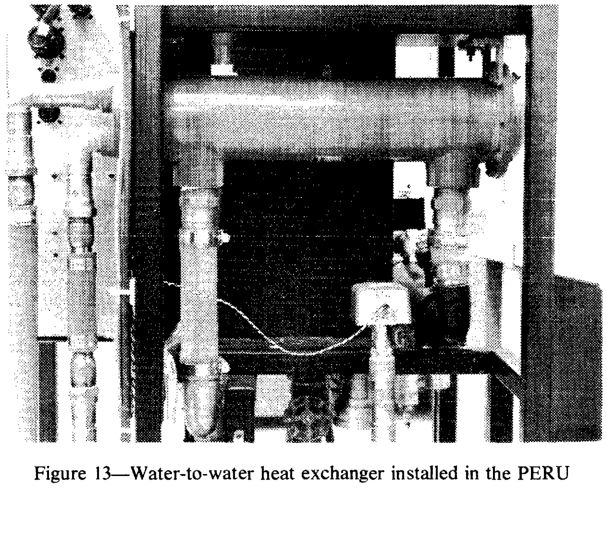


Figure 13—Water-to-water heat exchanger installed in the PERU

The PERU serves as the heart and kidneys of the system. From the distribution manifold of the PERU, internal water can flow to four independent cooling loops, one or two per main frame cabinet. The water returns to the in-line reservoir where it is then recirculated by the pumps through the shell side of the heat exchanger (or the COOL if that option is used) and to the distribution manifold again. The impact of water cooling on packaging sizes can be appreciated by noting the size of the water-to-water heat exchanger in Figure 13. This unit is capable of transferring 20 KW with a difference in entering water temperatures of only 10°C.

Any one of the cooling loops may be disconnected without interrupting the flow through the other loops since automatic shut-off quick disconnects are used (Figure 14). This permits repairs or cabinet removal without total loss of mainframe availability to the user (assuming a configuration of more than one processor, which is available in the Model 66/85 line).

This design concept of parallel flow extends into the cooling loops in the mainframe cabinets. Each loop is designed to cool micropackages on up to nine planar boards. Each flow path to a board is connected to the cabinet plumbing in parallel with all other boards. This allows many option configurations with no impact on the cooling performance. Even the 12 SLIC's on each board are connected in four groups of three SLIC's in series. In addition to the parallel flow through the board paths, the power supplies are connected to the cabinet plumbing in parallel. This highly parallel flow pattern allows the resultant low pressure in the system which directly contributes to the high safety margin and reliability of the system components. The flow pattern is shown symbolically in Figure 15.

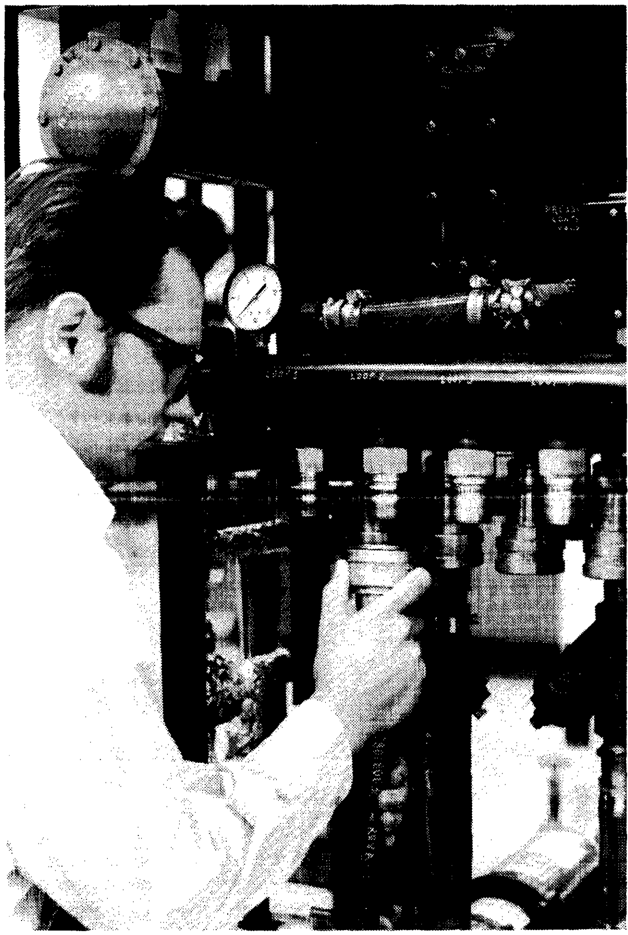


Figure 14—This view shows how up to four individual cabinet cooling loops can be attached to one PERU. The end of the water-to-water heat exchanger can be seen above the man's head

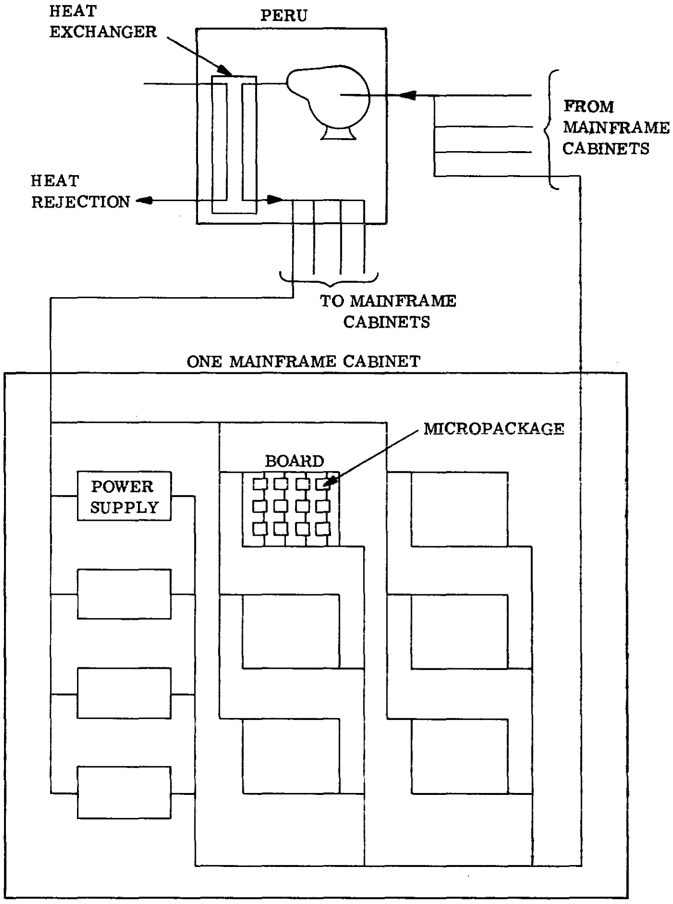


Figure 15—Block diagram of flow in the cooling system

The system pressure is set at the time of installation by adjusting a bypass valve on the PERU manifold, and no further regulation is required while the system is operational. The total flow rate through all four cooling loops for a Model 66/85 dual processor configuration is 150 litres per minute. This is provided by two magnetic driven seal-less pumps operating in parallel. Two parallel pumps were chosen instead of one large pump for machine availability reasons. If one pump should ever fail, the other pump alone provides adequate flow to cool the machine (recall that the flow design point was chosen to be far higher than required for a safe design). Check valves at the outlet of each pump prevent backflow through a failed pump. A pressure switch senses a reduction of pressure at the pump discharge and triggers a warning to the operator that the cooling system requires attention, but the computer does not shut down and continues its normal operation.

Over-pressure conditions (such as due to removing one or more cooling loops without adjusting the bypass valve on the manifold) are prevented by a pressure switch on the manifold which automatically turns off one pump if the pressure exceeds the normal operating pressure.

A two position float switch in the reservoir indicates first alarm, then shutdown conditions if there is a loss of internal water. Under the alarm condition, the computer is still available to the user.

In each cooling loop in the cabinet there is a pressure switch to indicate cooling availability. This switch serves as an interlock to prevent the power supplies from being turned on unless the cooling loop is connected to the PERU and water is being pumped. In addition to this switch, all power supplies have thermal switches which automatically disable them if they are not being cooled.

Except for the SLIC and super SLIC, the rest of the components in the system are "off-the-shelf" items or built from standard plumbing parts. The pumps, heat exchanger, mixing system (three way valve and servo motor), quick disconnects, etc., are used in a wide variety of applications ranging from industrial air conditioning control and marine engine cooling to pumping of corrosive chemicals. Therefore, these items are not only readily available as a result of mass production, but they are designed to operate reliably under conditions far more severe than this application to computer cooling.

The plumbing in the PERU is fabricated from copper/brass parts and the major assemblies are connected by reinforced hydraulic hose to eliminate all tolerance problems. The pressure ratings on these parts (including the hose) are 1700 kPa and higher. The plumbing assemblies in the cabinets are fabricated from schedule 40 PVC (rated at 4000 kPa) and these assemblies are connected with thick walled PVC flexible tubing (rated at 270 kPa).

The result of using "off-the-shelf" components and standard plumbing materials is a system of extremely conservative design, which can be assembled in small quantities at a low cost, virtually equivalent to mass produced cost. Coupling the high safety factor of the parts with the low pressure of the system and the warning devices previously mentioned yields a highly reliable cooling system and high computer availability to the user.

GO System—Design and implementation of an output generator*

by ROLAND R. BONATO and KENNETH C. YANG

The George Washington University
Washington, D.C.

ABSTRACT

This paper concerns a conceptual model for a general purpose output generator and its implementation. The Generated Output (GO) System is based on two major criteria: (1) user oriented and (2) easy maintenance. Its goal is to produce tailored output in an unambiguous, camera-ready form. Basically, the system is comprised of a derived file, a descriptor file and an interface module which integrates the two files according to externally supplied specifications. Derived data are usually numeric results such as percentages, totals, subtotals, etc. Descriptors are defined as alphanumeric labels whose function is to clearly identify derived data. Classification of different types of calculated and label files are discussed with accompanying examples and illustrations.

BACKGROUND

In a society where computer output is being widely used for business and scientific reporting, this medium has become an increasingly accepted vehicle for human communication. A review of computer output during the past two decades indicates that there is a trend toward more complex formatting. The change is mostly due to an expanding and more varied usership; i.e., as the consumer became more sophisticated, the demand for custom-tailored or even camera-ready output increased. For the most part, the DP community responded to these demands by developing a compiler approach to producing statistical tables. The Report Program Generator (RPG) by IBM and Table Producing Language (TPL) by the Bureau of Labor Statistics are examples of systems that were specifically designed to accommodate a variety of output displays.

INTRODUCTION

A major problem in the production of computer output involves the manipulation of alphanumeric descriptors, i.e., labels whose sole function is to clearly identify data formatted output. As opposed to calculations, labels are not readily derived from algorithmic models. At the Biometric Laboratory, The George Washington University, we reviewed the problems associated with output formatting and came to the following conclusions:

(1) Labeling requires extensive amounts of tedious formatting effort. Consequently, labeling tends to be sparse and abbreviated.
(2) Revision of output formats is equally tedious.
(3) General purpose programs, in order to maintain their generality, tend to resort to generic labels, e.g. % or $. However, there are many instances where such generic labels do not suffice.
(4) When unambiguous labeling is required, the programmer resorts to "one shot" programming consisting of repetitive calculation logic and unique alphanumeric formats; i.e., the programmer is forced to "re-invent the wheel" calculationwise because of labeling specifics.

In order to reduce these problems and provide flexibility in generating output, we have developed a system based on two major criteria:

(1) User oriented—users should be able to specify not only numerical results but also the terminology to be employed and the positioning and ordering of their output.
(2) Easy maintenance—utilization and maintenance of the system should be accomplished mostly by trained technicians.

The following sections describe the design, basic concepts, implementation, and example output from our system.

DESIGN

Given the above-mentioned criteria, we decided to treat derived data and their associated descriptors as separate entities, i.e., independently-generated files.

* Preparation of this paper was supported by Grant #5-R01-MH-22019-03 from the National Institute of Mental Health, U.S. Public Health Service.

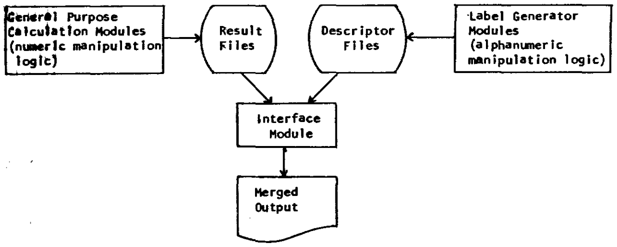


Figure 1—Design of an output generator

Derived data are viewed as calculated results such as percentages, totals, subtotals, etc. Descriptors are defined as those alphanumeric labels needed to identify results. Merged output is achieved via an interface module which accesses both the results and descriptor files as shown in Figure 1.

This design has the following advantages:

(1) General purpose calculation programs can be written independent of formatting constraints.
(2) The descriptor file can be modified and maintained as a separate entity.
(3) The interface module provides flexibility in the arrangement of final output, e.g., configuration of output can be tailored to the individual user without extensive reprogramming.

BASIC CONCEPTS

The purpose of the following discussion is to explore concepts that will yield results and descriptors with some degree of generality and a minimum of effort.

As viewed here, all output can be subdivided into a series of component parts.

Figure 2 represents the concepts involved in the GO system, from the most elementary to the most complex and their interrelationship. Starting at the top of the schematic, TOTAL OUTPUT is made up of PAGE IMAGES, and one or more TABLE(S) constitute a PAGE IMAGE. TABLES, in turn, are derived from RESULT CLASSES and LABEL TYPES. Finally, results and labels are composed of elements, i.e., a single calculation or an alphanumeric word.

A page image is defined as an output unit whose content forms a logical whole within the total output. Although the dimensions of a page image are theoretically unlimited, an actual page image introduces certain constraints; e.g., ma-

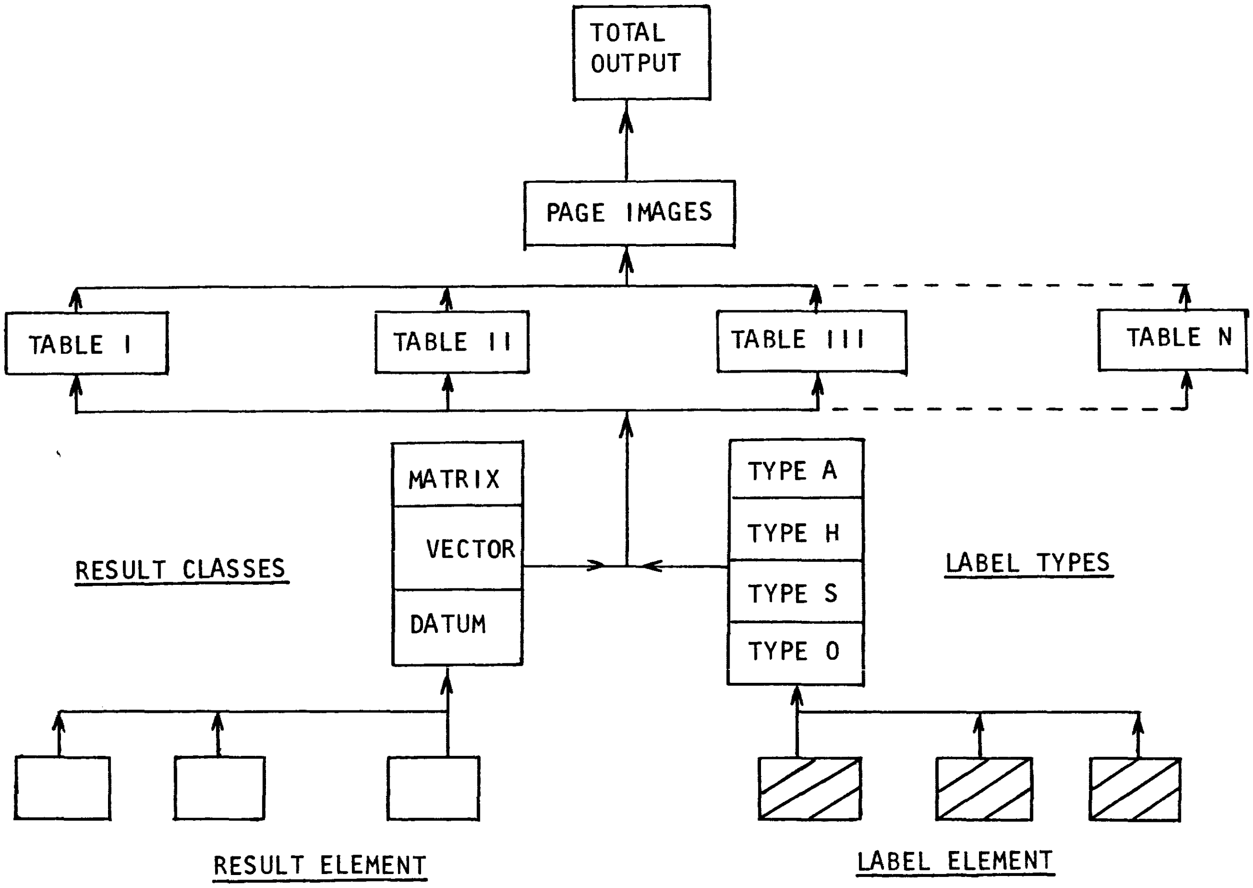


Figure 2—Components of output

chine limitations, such as finite number of print positions, tend to influence the page size.

A table is defined as a merged display of results and its descriptors or labels (whenever a single table is involved, the page image is the table). All results can be divided into three classes: a datum (1×1), a vector (1×N), and a matrix (N×N). Descriptors are divided into four types: All alphanumeric (TYPE A), Headers (TYPE H), Stubs (TYPE S) and Others (TYPE O).

Label definitions

(1) All alphanumeric (TYPE A) labels consist of one or more lines of pure alphanumeric output. As opposed to TYPES H and S, their position is minimally correlated with the result classes; i.e., the form and alignment of TYPE A is not determined by derived output. TYPE A is commonly used for table description, footnotes, etc.
(2) Headers (TYPE H) are defined as labels that describe columnar results. The position of TYPE H elements is a function of the format of the result matrix and vector.
(3) Stubs (TYPE S) are labels that define results rowwise and are similar to TYPE H in that the location of TYPE S elements is determined by results.
(4) Other (TYPE O) labels describe data plus special characters that are not necessarily associated with results elements.

Results definitions

The meaning of the three result classes is commonly known. However, the definitions apply to output requirements as opposed to the calculated results format.

Tables and page image

Usually, the most convenient type of output involves the juxtaposition of related data on a single physical page. This saves paging back and forth, transcribing, or cutting and pasting. In addition to convenience, interrelatedness and machine limitations, aesthetics may be a major determiner of the page image, e.g., neat and uncluttered output.

From the above, it becomes apparent that specification of page images is a function of the number, redundancy, symmetry, extent and variety of page elements. For example, when the same page image can be repeated across many physical pages (redundancy), the cost/benefit ratio is enhanced. Similarly, when label and result elements are evenly spaced, symmetrical tables are produced and the positioning of the table is simplified. Usually, the most "costly" tables to specify are a series of non-redundant asymmetrical labels and results.

IMPLEMENTATION

An experimental model of GO (Generate Output) System has been developed and implemented at the Biometric Laboratory to test the concepts previously described. Throughout the developmental phase, we have designed numerous utilizable functions and implemented 3 principal modules and 9 sub-modules, serving key functions, for the GO System. All programs are written in FORTRAN and PL/1 for an IBM 370/135—265K—OS/VSI System.

The following represent the modules that have been implemented for the GO System:

(1) GOMAIN (Interface Main Module)—This module accepts input parameters, refers to appropriate result and label files, and combines appropriate submodules for flexible output. Its flexibility can be observed from the ensuing examples.
(2) GOCLAR (Interface sub-module)—To clear the area, buffer, in which the results and labels will be merged.
(3) GOREST (Interface sub-module)—To restore a page image for repetitious use of the same page image with differential results.
(4) GOGRED (Interface sub-module)—used as input function for generation of graph displays.
(5) GOGRPD (Interface sub-module)—used in conjunction with GOGRED to generate and plot data for displays.
(6) GODATM (Interface sub-module)—to transfer datum results to output buffer.
(7) GOVECT (Interface sub-module)—to transfer vector results to output buffer.
(8) GOMATX (Interface sub-module)—to transfer matrix results to output buffer.
(9) GOUPDT (Interface sub-module)—used as update/modify function for previously-created page images.

TABLE I—Page Image

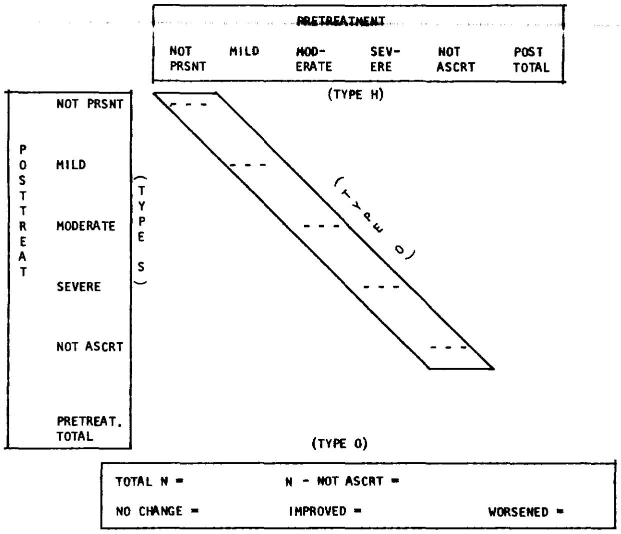


TABLE II—Merged Page Image and Results

(TYPE A)

PSYCHIATRIC RATING SCALE - ANXIETY

PRETREATMENT

POSTTREAT	NOT PRSNT	MILD	MODERATE	SEVERE	NOT ASCRT	POST TOTAL
NOT PRSNT	0	14	11	4	0	29
MILD	1	2	22	11	0	36
			MATRIX			
MODERATE	0	1	2	16	1	20
SEVERE	0	1	2	7	1	11
NOT ASCRT	0	0	1	0	0	1
PRETREAT. TOTAL	1	18	38	38	2	←VECTORS

TOTAL N = 97   N - NOT ASCRT = 94

NO CHANGE = 11   IMPROVED = 78   WORSENED = 5

DATUM   DATUM   DATUM

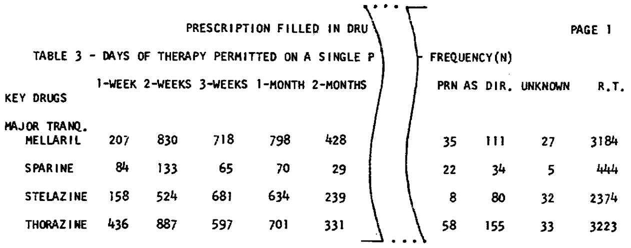
(10) GOPRNT (Interface sub-module)—prints the merged output which resides in the buffer in user-specified format.

(11) GOPAGE (Label Generator Module)—to create Page Directory constituted of page images tailored to user-specified output.

(12) GOSTAB (General Purpose Calculation Module)—Multiways cross-tabulation and computations of n, ΣX, $\bar{X}$, and σ conducted (inclusive of row %, column %, table %, cumulative row %, cumulative column % for n and ΣX). The results are documented and stored on temporary disk file (result file) to be utilized by the GOMAIN.

During the developmental and design phase of the GO System, we also envisioned a spin-off from this concept,

TABLE III—Frequency Table

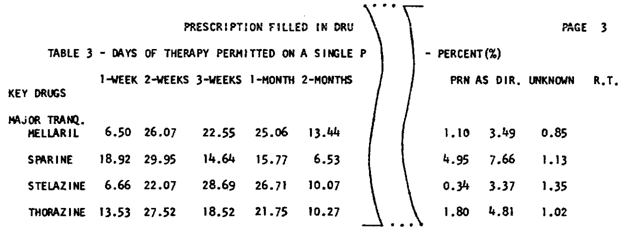


PRESCRIPTION FILLED IN DRU … PAGE 1

TABLE 3 - DAYS OF THERAPY PERMITTED ON A SINGLE P … - FREQUENCY(N)

KEY DRUGS	1-WEEK	2-WEEKS	3-WEEKS	1-MONTH	2-MONTHS	PRN	AS DIR.	UNKNOWN	R.T.
MAJOR TRANQ.									
MELLARIL	207	830	718	798	428	35	111	27	3184
SPARINE	84	133	65	70	29	22	34	5	444
STELAZINE	158	524	681	634	239	8	80	32	2374
THORAZINE	436	887	597	701	331	58	155	33	3223

TABLE IV—Percent Table

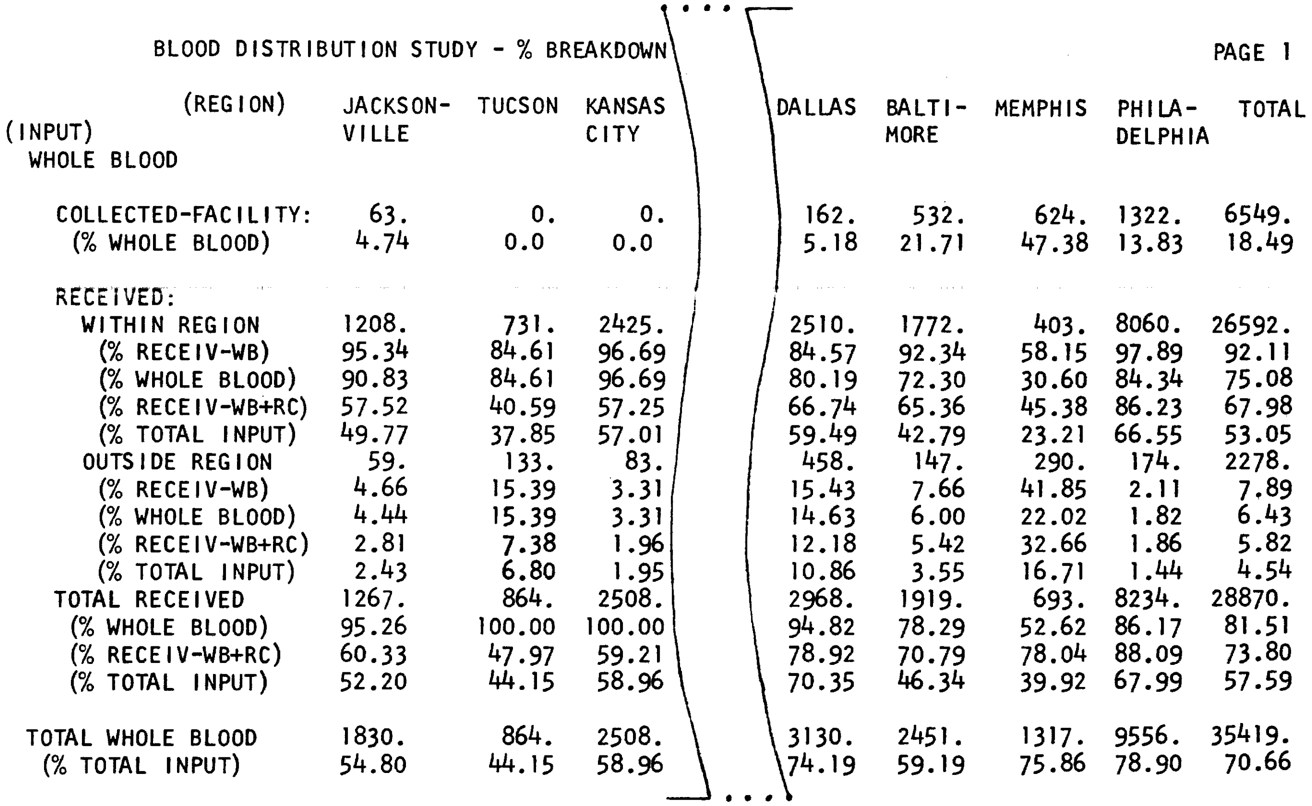


PRESCRIPTION FILLED IN DRU … PAGE 3

TABLE 3 - DAYS OF THERAPY PERMITTED ON A SINGLE P … - PERCENT(%)

KEY DRUGS	1-WEEK	2-WEEKS	3-WEEKS	1-MONTH	2-MONTHS	PRN	AS DIR.	UNKNOWN	R.T.
MAJOR TRANQ.									
MELLARIL	6.50	26.07	22.55	25.06	13.44	1.10	3.49	0.85	
SPARINE	18.92	29.95	14.64	15.77	6.53	4.95	7.66	1.13	
STELAZINE	6.66	22.07	28.69	26.71	10.07	0.34	3.37	1.35	
THORAZINE	13.53	27.52	18.52	21.75	10.27	1.80	4.81	1.02	

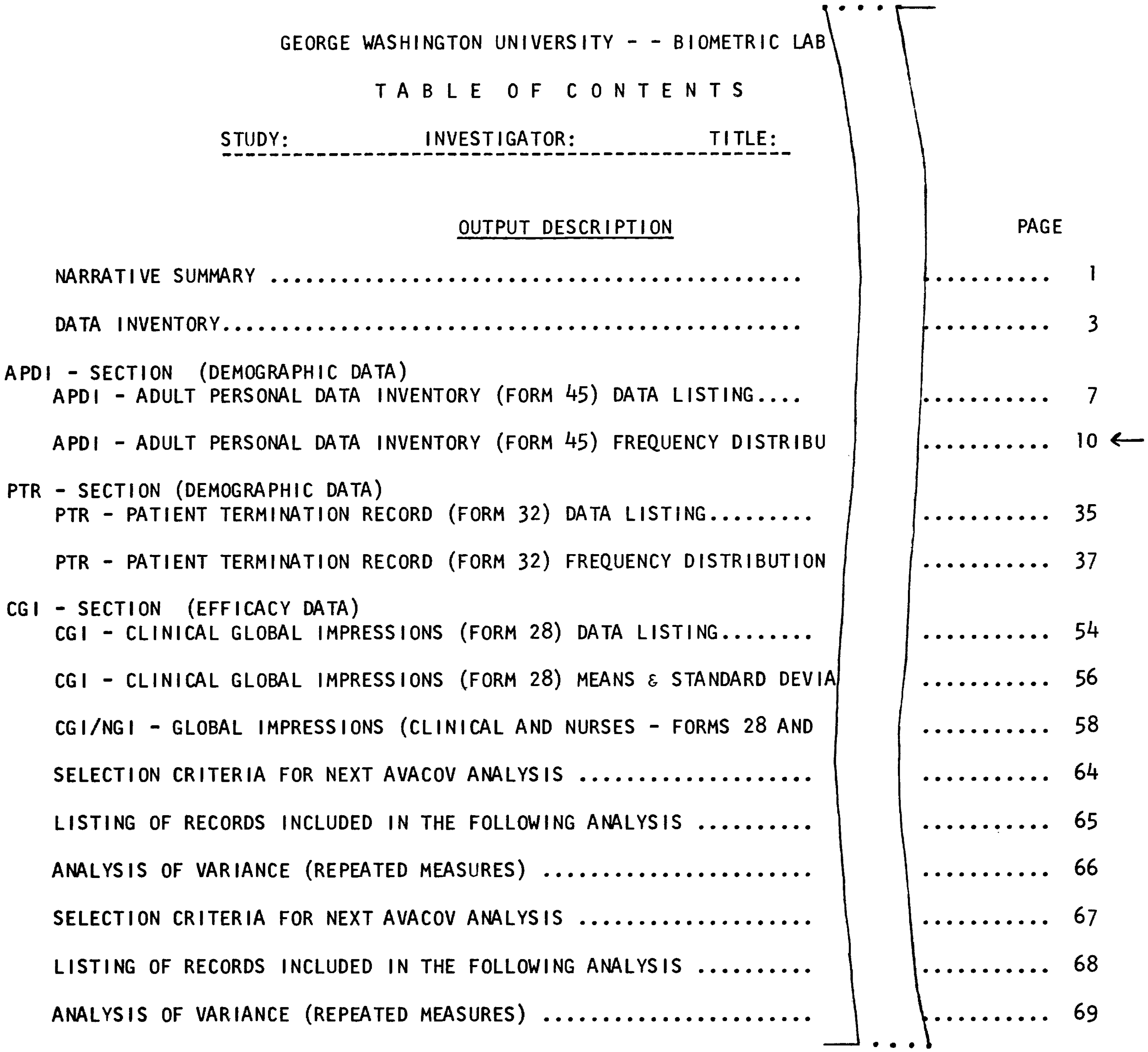
TABLE V—Summary Table (Scientific Application)



BLOOD DISTRIBUTION STUDY - % BREAKDOWN … PAGE 1

(REGION) (INPUT)	JACKSON- VILLE	TUCSON	KANSAS CITY	DALLAS	BALTI- MORE	MEMPHIS	PHILA- DELPHIA	TOTAL
WHOLE BLOOD								
COLLECTED-FACILITY:	63.	0.	0.	162.	532.	624.	1322.	6549.
(% WHOLE BLOOD)	4.74	0.0	0.0	5.18	21.71	47.38	13.83	18.49
RECEIVED:								
WITHIN REGION	1208.	731.	2425.	2510.	1772.	403.	8060.	26592.
(% RECEIV-WB)	95.34	84.61	96.69	84.57	92.34	58.15	97.89	92.11
(% WHOLE BLOOD)	90.83	84.61	96.69	80.19	72.30	30.60	84.34	75.08
(% RECEIV-WB+RC)	57.52	40.59	57.25	66.74	65.36	45.38	86.23	67.98
(% TOTAL INPUT)	49.77	37.85	57.01	59.49	42.79	23.21	66.55	53.05
OUTSIDE REGION	59.	133.	83.	458.	147.	290.	174.	2278.
(% RECEIV-WB)	4.66	15.39	3.31	15.43	7.66	41.85	2.11	7.89
(% WHOLE BLOOD)	4.44	15.39	3.31	14.63	6.00	22.02	1.82	6.43
(% RECEIV-WB+RC)	2.81	7.38	1.96	12.18	5.42	32.66	1.86	5.82
(% TOTAL INPUT)	2.43	6.80	1.95	10.86	3.55	16.71	1.44	4.54
TOTAL RECEIVED	1267.	864.	2508.	2968.	1919.	693.	8234.	28870.
(% WHOLE BLOOD)	95.26	100.00	100.00	94.82	78.29	52.62	86.17	81.51
(% RECEIV-WB+RC)	60.33	47.97	59.21	78.92	70.79	78.04	88.09	73.80
(% TOTAL INPUT)	52.20	44.15	58.96	70.35	46.34	39.92	67.99	57.59
TOTAL WHOLE BLOOD	1830.	864.	2508.	3130.	2451.	1317.	9556.	35419.
(% TOTAL INPUT)	54.80	44.15	58.96	74.19	59.19	75.86	78.90	70.66

TABLE VI—Tables of Contents



```
                 GEORGE WASHINGTON UNIVERSITY - - BIOMETRIC LAB

                        T A B L E   O F   C O N T E N T S

              STUDY:             INVESTIGATOR:             TITLE:
              ----------------------------------------------------


                              OUTPUT DESCRIPTION                              PAGE

    NARRATIVE SUMMARY ............................................  ..........   1

    DATA INVENTORY................................................  ..........   3

APDI - SECTION  (DEMOGRAPHIC DATA)
    APDI - ADULT PERSONAL DATA INVENTORY (FORM 45) DATA LISTING....  ..........   7

    APDI - ADULT PERSONAL DATA INVENTORY (FORM 45) FREQUENCY DISTRIBU ..........  10 <—

PTR - SECTION (DEMOGRAPHIC DATA)
    PTR - PATIENT TERMINATION RECORD (FORM 32) DATA LISTING.........  ..........  35

    PTR - PATIENT TERMINATION RECORD (FORM 32) FREQUENCY DISTRIBUTION ..........  37

CGI - SECTION  (EFFICACY DATA)
    CGI - CLINICAL GLOBAL IMPRESSIONS (FORM 28) DATA LISTING........  ..........  54

    CGI - CLINICAL GLOBAL IMPRESSIONS (FORM 28) MEANS & STANDARD DEVIA ..........  56

    CGI/NGI - GLOBAL IMPRESSIONS (CLINICAL AND NURSES - FORMS 28 AND  ..........  58

    SELECTION CRITERIA FOR NEXT AVACOV ANALYSIS ....................  ..........  64

    LISTING OF RECORDS INCLUDED IN THE FOLLOWING ANALYSIS ..........  ..........  65

    ANALYSIS OF VARIANCE (REPEATED MEASURES) .......................  ..........  66

    SELECTION CRITERIA FOR NEXT AVACOV ANALYSIS ....................  ..........  67

    LISTING OF RECORDS INCLUDED IN THE FOLLOWING ANALYSIS ..........  ..........  68

    ANALYSIS OF VARIANCE (REPEATED MEASURES) .......................  ..........  69
```

which could enhance the documentation need of the drug studies data we process, i.e., the implementation of a book system. The Book System searches the completed output files for a given study and then organizes the output in book form with a table of contents, organized and paginated output, and a key terminology index.

EXAMPLE OUTPUT

When the output requires formulation of symmetrical tables, a page image on the printer paper can be envisioned as a field of 132×66 matrix, and any number of sub-fields can be constructed within the field. Table I shows the upper left quadrant of a Pretreatment/Posttreatment cross-tabulations page image. Since the output requires juxtaposition of four identical tables, only this table is generated and then reflected onto three other quadrants for the complete page image. Three types of label (H, S, and O) are identified in this table.

Table II shows calculated results from GOSTAB being merged with the table produced above. One type of label (A) and all three classes of results are identified in this table.

TABLE VII—Subject Age Table (page 10)

```
                    STUDY:          INVESTIGATOR:           TITLE:
                    ----------------------------------------------------------------

    APDI - ADULT PERSONAL DATA INVENTORY (FORM 45)  FREQUENCY DISTRIBUTIONS

GROUP 1


 TABLE NO.  1 ------------SUBJECT AGE
                                                     HISTOGRAM OF PERCENT
 CATEGORY      CODE   FREQ-   PER-    CUM-   CUM-    ------------------
   NAME        NO.    UENCY   CENT    FREQ.   %        10   20   30   40   50   60   70   80
                                                     ....x....x....x....x....x....x....x....x...
21 YRS.OLD.     21      5      5.2      5     5.2   :11
23 YRS.OLD.     23      8      8.2     13    13.4   :1111
24 YRS.OLD.     24     11     11.3     24    24.7   :111111
25 YRS.OLD.     25     14     14.4     38    39.1   :1111111
27 YRS.OLD.     27     20     20.6     58    59.7   :111111111
29 YRS.OLD.     29     16     16.5     74    76.2   :11111111
33 YRS.OLD.     33     13     13.4     87    89.6   :111111
43 YRS.OLD.     43      7      7.2     94    96.8   :11111
44 YRS.OLD.     44      3      3.1     97    99.9   :1

   N = 97      MEAN =  28.54        S.D.=   8.00

   MISSING DATA CODE= 99   FREQ.=   0  PER CENT =   0.0   TOTAL N = 97


                                                        (10) ←
```

TABLE VIII—Index

```
                STUDY:                  INVESTIGATOR:          TITLE:
                ------------------------------------------------------------------------

                                        I N D E X

DEMOGRAPHY OF PATIENTS AND STUDY
  MEAN AGE...................................................10 ←
  TOTAL NUMBER OF DAYS IN THIS STUDY.........................37
  EARLY TERMINATORS..........................................38
  OUTPATIENTS DISPOSITION AT TERMINATION.....................53

PSYCHIATRIC RATING
  SIGNIFICANT AT .05 LEVEL...................................66,69,120,122,126,127,128,
                                                             186
  SIGNIFICANT (G. G.)........................................66,69,127,128,131,137,139,

TOXICITY
  RIGIDITY...................................................228,229,231,232
  TREMOR.....................................................228,229,231,232
```

Tables III and IV illustrate the tailoring of output according to user specifications. For this application, GOSTAB accepted a two-way cross-tabulation of Drug by Duration (42×14) design, results generated and stored in design order; i.e., 14 elements (frequency, percent, cumulative percent, etc.) are stored together for each cell of the design. However, the user wanted the data displayed as separate tables (frequency table, percent table, cumulative percent table, etc.). Here, only one page image was created and results were distributed selectively to the appropriate tables for output.

Table V shows summaries that are frequently used in both industry and science, i.e., percentages, subtotals, and totals.

Tables VI, VII, and VIII are illustrated to show some extracted materials from one of our completely documented studies. (Page 10, ←arrow marked, has been selected to demonstrate page reference capability of this book system).

CONCLUSION

Based on our experience with the GO System, we firmly believe that our approach is a viable one. We continue to develop more general algorithms for the Interface executive monitor, so that it will accommodate more flexible output. Descriptors and results files are generated separately and are maintained by trained technicians. Our first generalized computational module (GOSTAB) proves that other modules can be readily incorporated into the GO System.

A talking computer terminal

by JAMES A. KUTSCH, JR.
West Virginia University
Morgantown, West Virginia

ABSTRACT

In an attempt to provide a communications medium for blind computer users, the requirements for an optimal method of communications were set forth. Previous solutions to the problem were studied and their shortcomings were noted. Through a combination of ideas from previous approaches and the author's personal experience, a new and unique direction was taken, which resulted in a host independent talking computer terminal.

The hardware for the terminal, consisting of a microprocessor, a speech synthesizer, a keyboard, and a modem, were assembled to form a prototype system. An analysis of algorithms for the translation from text to synthesizer commands was conducted, resulting in a decision to use a technique designed by McIlroy from Bell Laboratories, with modifications to tailor it to the needs of the talking terminal. The necessary software was written and the talking terminal was demonstrated using the University of Illinois' PDP 10 as the host computer. It is currently being used at West Virginia University on an IBM 360-75 as a research tool of the author.

Pictured is the Talking Terminal which is currently connected to West Virginia University's IBM 360-75. It is used extensively by the author and others as a research tool.

INTRODUCTION

Finding a reading aid or method for the blind has been a long standing research problem. The earliest significant contribution to the problem was the work of Louis Braille in the mid 1800's. More recently, computers and other sophisticated electronic equipment have been used in an attempt to find a better solution.

Because of the specific nature of the problem, or perhaps because it affects only a small percentage of the population, it has not received the research attention it deserves. Some major contributions have been made, but they are few and far between. Further, almost all of the research that has been done was conducted by sighted individuals, with little or no input from blind persons who would be the eventual users of any discovery.

A sighted person's opinion of what he would want or need if he were blind is not always a true reflection of the needs of the blind. The author, himself blind, hopes that his personal experience will lend an insight into the problems of the blind which is sometimes lacking in other research.

MAN-TO-MAN COMMUNICATIONS

Communications through a non-verbal medium is a difficult problem for a blind individual. Since the visual channel ~~is not available; all printed and otherwise visual information~~ must be converted to data suitable for one of the remaining senses. The most commonly chosen secondary channels are auditory or tactile. Both are seriously limited in bandwidth as compared with vision.

Common solutions include transcription to Braille and the reading of printed material either face to face or via a tape recorder. Even though many textbooks and novels are available on tape and records from the Library of Congress and Recordings for the Blind and in Braille from the American Printinghouse for the Blind, there are still serious drawbacks to the loss of the visual channel. Of primary concern is the availability of this material. That is, although brailled and recorded books are adequate, they are not as readily available as the printed word. Not every book, paper, or magazine that a blind person might wish to read has been converted. Further, the conversion is time-consuming and requires the assistance of a sighted person.

A major contribution to the solution of this problem was made by Telesensory Systems, Inc. when they began production of the OPTICON (optical to tactile converter).[1] A small camera is moved across a line of printed information. A tactile image of each letter so scanned is presented through a matrix of vibrating reeds. These raised impressions are read (felt) by the user. The user of the OPTICON can, without assistance, read the printed page. It should be noted that it takes considerable training to recognize the raised impressions of the letters, and even then, reading rates remain disappointingly low. Experienced users have not been able to exceed 70 words per minute.[2]

MAN-TO-MACHINE COMMUNICATIONS

A blind computer user is faced with the same problems of communications except that he is communicating with a computer rather than another person. Obviously, the input to the computer is not a problem, since keypunches and terminals are keyboard devices and although the location of the keys may vary, they are quite similar to those of a typewriter. Thus, input to the computer can be achieved with only the inconvenience of learning a new keyboard, which is also required of the sighted computer user.

The transfer of information from the computer to the blind user is similar to the man-to-man communications problems discussed above. However, there are two major differences. First, the data cannot already have been transcribed into Braille or recorded on tape because every computer output listing is unique. This means that the blind user must make arrangements to have someone transcribe or read his output or he must read it himself with an image-to-tactile converter. Further, it is quite difficult, if not impossible, for the average blind person to use a time-sharing terminal interactively without someone with him to read the output or without the use of an OPTICON.

The second difference is that the data is already in computer readable form. If the computer could itself present the data in something other than a visual medium, the man-to-machine communications problem would be solved. Now, only how to convert and present the data needs to be determined.

DETAILS OF THE PROBLEM

The purpose of this study is to design an optimal method for a blind person to communicate with the computer and to assemble whatever hardware and software is necessary to achieve such communications. The criteria against which the adequacy of a solution can be measured will be set forth so that any shortcomings of a particular system can be more easily discovered.

One of the most important qualifications of any communications technique is that it should be available whenever the individual wishes to use the computer system. This precludes the use of another human being to read the output, since scheduling can be quite difficult. Therefore, the communications method should involve some machine or mechanism that is available whenever the computer itself is available to the average user.

Second, the communications method should be as complete and as versatile as possible. It should be able to be used in many aspects of computer communications. For example, if the computer system can be used in a batch mode through punch cards and listings or in a time-sharing mode from a terminal, the communications method should be applicable to both.

Third, the converted data presented to the user should be as thorough and complete as that presented to a sighted user. All system messages, job control language, log-on preambles, etc. which are generated from normal use should be presented unmodified to the blind user. Further, all system resources should be available. This precludes the use of restricted or specifically modified languages, compilers, editors, etc. for use by the blind.

Fourth, the communications method should not require that the user acquire some special or unusual skill, e.g., Morse code or the ability to distinguish musical chords. The user of the system should be able to interact with the computer with no more training than is expected of a sighted user.

Fifth, in the same way a teletypewriter can be attached to many widely varied computers, the communications method should be computer independent. This would allow the blind computer user to communicate with many different computers without requiring totally different methods for different computers. This goal is considerably more difficult to realize, but is very important.

Finally, the communications method should be equivalent in cost to devices required by a sighted person to communicate with a computer. A company wishing to hire a blind person should not be expected to pay large sums of money for equipment to enable the blind person to do the same job as a sighted person who would need no special equipment.

These six criteria will be used to measure the effectiveness of various methods of communications. Input to the computer will not be discussed because it is easily achieved through keyboard devices which blind people can operate without any difficulty. Emphasis will be on ways to convert and present data through a non-visual medium.

INADEQUACIES OF PREVIOUS SOLUTIONS

Braille is the most common and perhaps the easiest form of output for the blind. However, it is not without disadvantages. The Braille character consists of a matrix of dots, two wide and three high. Counting horizontal and vertical spacing between characters, 40 Braille characters would be 120 characters wide and four lines high on a printer. Thus, what would appear on one line of 120 characters of print requires a maximum of twelve lines of dots and spaces in Braille. Even with the use of compression techniques for blanks and short lines, Braille is still very bulky. For example, a thirty volume encyclopedia would consist of 145 volumes of five inch thick books in Braille. Further, computer generated Braille requires some sort of change to the line printer, which means a time delay in most computer installations. Special forms jobs are seldom run more than a few times a day. Because of its bulk, Braille output is quite wasteful, especially if only a few lines of each listing are needed, as is commonly the case in debugging a program. Braille output does provide a good means of storing information for later use and providing a listing which can be studied many times. However, it is easily destroyed by placing many listings or other heavy objects on top of it, causing the raised dots to be erased. Properly embossed Braille on special paper has an average life of fifty readings.[3] (Embossing against a soft base does not produce as well as when a metal die is used. The soft base limits dot height by allowing the dot base to expand.)

On the surface, the OPTICON seems to present a very satisfactory solution. It is computer independent, readily available, reads all forms of output (it can even be used to read the face of a cathode ray tube display), and presents no restrictions on what computer services are used. It is, in fact, quite adequate for reading listings in a batch environment. But, since it presents the tactile data on the user's fingertip, it is inconvenient to use at a time-sharing terminal, since the user must continually move his hand from the OPTICON to the keyboard and back again. Also, the reading speed with the OPTICON is a consideration against its use, as stated earlier.

Additionally, with both Braille and the OPTICON, a sensory fatigue problem enters after prolonged use without rest. This fatigue could be compared to eye strain. The Braille or OPTICON reading finger becomes tired and the ability to detect and distinguish characters lessens, causing errors and slowdown in reading. Although experienced readers may be able to work for several hours before becoming fatigued, others cannot continue for more than a half hour without giving the reading finger a rest. Since most computer programmers spend several hours at a time at a terminal, this drawback would present serious problems.

DIGITAL SPEECH SYNTHESIS SYSTEMS

The final medium to discuss is that of synthetic speech. Many companies are producing various types of voice response systems.[4] However, in most cases, system details are proprietary. Synthetic speech seems to be the optimal medium for the computer to communicate with blind users, indeed, with man. There is sufficient reason to believe that no arbitrary letter-by-letter code could ever be as efficient and understandable as the spoken language of the user. To bridge the gap between the computer and the blind user, the computer must "speak."[5,6]

Recently, Masters Specialties Company has developed a technique for digitalizing and storing whole words in metal oxide semiconductor (MOS) read only memory (ROM) chips. By a complex plotting of waveforms, engineers have converted analog audio signals into digital signals requiring a minimum of storage.

In the earliest voice response systems words, phrases, and occasionally syllables were recorded on photographic film or magnetic drums.[4,11] These methods are very useful in systems that require a small fixed vocabulary, e.g., time and temperature units. However, their adaptation to more sophisticated systems is limited.

Rather than dealing with large units such as phrases or words, the most versatile synthesizers deal with phonemes, the smallest unit of the spoken language. They accept digital phoneme commands and return their audible counterpart. This approach requires sophisticated software to convert from letters to phonemes. There are several algorithms for this translation ranging from dictionary look-up techniques[6-8] to synthesis by rule.[9-10]

The problem with this method of synthesis is that the words to be synthesized must be converted from graphemes to the corresponding phonemes. (Graphemes are the 26 letters of the alphabet and all special symbols.) This is not a trivial process since the English language does not follow any simple set of rules for letter to sound correspondence. This is most apparent with vowels. For example, the sound corresponding to the grapheme "O" in "women" and "Bob" is clearly quite different. Since phoneme translation is context dependent, these algorithms usually require large tables and a moderate amount of computation. However, computer generated speech systems using synthesis by rule have been designed to produce quite satisfactory and understandable output.[10,12]

An example of such a system is the one at Michigan State University.[12,13] This system is a very significant step towards an optimal communications method between a computer and a blind user. Nevertheless, there are some serious shortcomings. First, the translation to phonemes is done in the host computer system, not in the terminal. Thus, the system is computer dependent. Although the software might be moved to another computer, such a transfer would almost certainly involve modification to the software, perhaps major revisions if the transfer were to another model or another manufacturer's computer. Further, only a subset of the system's resources are available to users of the special terminal. This may be adequate in beginning computer science education but is not at all satisfactory for a more advanced and experienced computer user. The most serious drawback, however, is that since the conversion to phonemes is a process on the host computer, some messages are sent directly to the terminal, bypassing the translation process. These characters, if sent directly to the speech synthesis device, would not produce words, and if sent to a teleprinter or CRT for printing, they could not be read by the user.

A TALKING TERMINAL

Considering all aspects of the above discussion, an optimal communications system for blind computer users was designed. Such a system should clearly be an interactive terminal, rather than a batch system, to allow more complete access to computer facilities. All batch facilities are usually available through an interactive terminal, but seldom is the reverse true. That is, in a terminal system, long jobs can be scheduled for execution in batch mode and have the output sent back to the terminal for later reading. However, interactive computing, and especially text editing, is available in a batch environment only in a very limited way.

Interactive text editors are of great advantage to the blind user. Corrections to program files can be made easily by reference to line numbers without the inconvenience and difficulty of finding specific punched cards in a deck.

Many sighted programmers use on-line terminals to write and debug their programs. This two-way communication speeds up the process. However, the blind programmer does not have this capability. Clearly, this disadvantage is felt. Recently, the ACM Special Interest Group for Computers and the Physically Handicapped circulated a questionaire among blind programmers about their special needs and interests. The results were published in SIGCAPH Newsletter Number 8, July 1, 1973. In response to "What special tools or equipment would you like to see developed?" readers wrote: "A faster means of reading than Braille;" "A machine to read ink output from the computer;" "Auditory output or input echo;" "Random access books;" "An easier way of finding or inserting cards in a deck;" "A card reader enabling me to make corrections myself."

Computer-generated speech of the phoneme synthesis variety is the best means of presenting the output. However, the translation from graphemes to phonemes clearly should not be done in the host computer due to the computer dependency problem and the inability to translate certain messages as mentioned above. Therefore, an intelligent terminal of some description must be incorporated so that the translation can be done in the terminal. This solves both problems. Since all messages are sent to the terminal and are translated there for presentation to the synthesizer, no messages can bypass proper translation. Also, the terminal can be constructed in such a way that it appears to be a teletypewriter or other conventional terminal to the host computer system. Thus, computer independence is maintained.

Additional advantages are present which can be attributed to the intelligence of the terminal. That is, some special processing of the data presented to the terminal can be performed if desired, e.g., reading only the first few characters of a line, selecting a mode of operation whereby the terminal spells all output, and other local features.

It appears that such a talking computer terminal would be the optimal method of communications for blind computer users. Specific details of the talking terminal will follow.

HARDWARE COMPONENTS

An analysis of a teletypewriter or a CRT type terminal yields the following components: a keyboard to enter the data; a coupling device such as a modem to communicate with the computer; and an output device, such as a teleprinter or cathode ray tube display. The talking terminal retains the keyboard and coupling device, and adds a speech synthesis device for output presentation, driven by a micro-processor. The micro-processor is necessary to maintain host independence, since synthesis devices need special codes to drive them. The standard ASCII characters sent to a terminal need to be converted to these special codes.

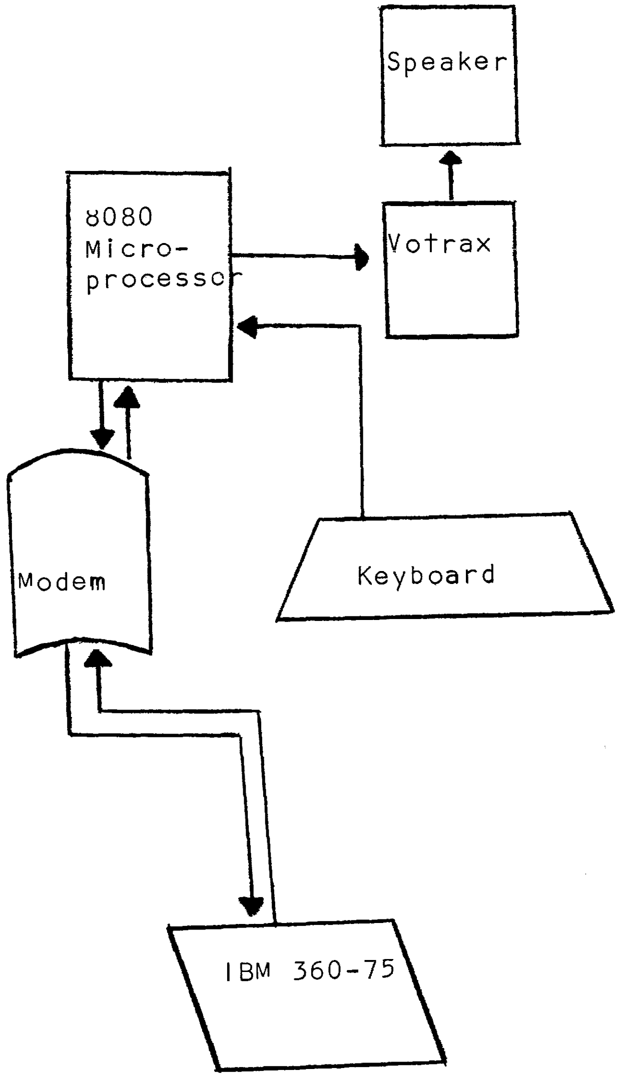


Figure 1—The diagram shows the relationship of the individual components with each of the others. Arrows indicate the path of data transfer.

In the selection of a specific manufacturer and model of the above equipment, several factors were taken into consideration. The talking terminal is considered to be a prototype device for experimentation. Therefore, ease in entering, modifying, and debugging the system's software was of great importance. In a final version of the terminal such features as front panel display lights and control switches, which were essential in the system's development, would not be necessary, and, in fact, might be undesirable. Other considerations in hardware selection were keeping the cost of the terminal as close as possible to the cost of a standard terminal, within the constraints of delivery time and availability of components.

The Votrax unit, manufactured by the Vocal Interface Division of Federal Screw Works was selected for the speech synthesis device.[14] This decision was based on several factors. First, the majority of speech synthesis applications found in the literature make use of the Votrax, and therefore, many of the algorithms for translation from text to phoneme codes are designed specifically for use with the Votrax. Further, after hearing the device demonstrated on several occasions, it was deemed adequate for the purposes of the talking terminal based on clarity of speech and speed.

The micro-processor system was selected to provide the intelligence for the terminal. Due to the slow data rate of ordinary speech, (on average, two to three words per second), the speed of the computer was not an important factor, even with a complicated algorithm for text-to-phoneme translation.

The Altair 8800 computer system manufactured by MITS, Inc. was used in the talking terminal. This decision was based on availability, low cost, modular design, and a bus structure that allows the system to be configured with any amount of memory and any number of input/output ports.

Input/output drivers and 16K bytes of static random access read-write memory were obtained from Processor Technology to complete the Altair computer system. Read-write memory was selected for the prototype terminal so that program changes could be easily effected. A final version of the terminal would require only 2K bytes of read-write memory for buffers and program variables. The remaining 14K bytes could be replaced with read only memory, permitting non-volatile storage of system software and tables.

SOFTWARE DESCRIPTION

The software for the terminal consists primarily of routines to perform the translation from English text to the digital commands necessary to drive the speech synthesis device. Routines to handle the communications between the terminal and the host computer system, as well as those to manage character storage buffers, complete the list of the terminal's software. The algorithms for the text to phoneme translation used in the terminal are those described by McIlroy from Bell Laboratories.[10] However, the algorithms have been modified to better suit the needs of the talking terminal.

The decision to use McIlroy's algorithms was based on several factors. First, his approach was designed specifically for use with the Votrax synthesizer. His rules and tables map directly into the phonemes used by the Votrax. (The phonemes used by the Votrax are not in a one-to-one correspondence with those in the International Phonetic Alphabet.) Further, McIlroy's algorithms deal with each word individually. That is, no attention is paid to context during the translation to phonemes. The talking terminal is designed to be used mostly, if not primarily, with computer languages. Abbreviated system messages of the type frequently found on the average time sharing system and other output data are usually not complete sentences. Therefore, the more expedient, less complicated method of independent processing of each word is better suited to this application than a context-dependent process.

The main body of the software is a 10,000 byte table which contains the letters of the alphabet, special symbols, some exception words, and word fragments, all with their corresponding phoneme equivalents.

The program consists of routines to access this table and preprocessing routines which mark long and short vowels.

CONCLUSIONS

The talking terminal does provide an optimal means for a blind programmer to communicate with the computer. However, the applications to the sighted world should not go without mention. For instance, there are security systems where an individual must constantly monitor a CRT screen for messages. This tedious task could be eliminated by using the talking terminal in place of the silent CRT. The employee is then freed to do other tasks and still not miss any incoming messages.

The terminal has potential in computer aided instruction. Young children have conversation ability developments that far exceed their reading skills. The talking terminal could communicate with these children more extensively than the printed page. Also, in a classroom environment, the talking terminal could be coupled to a public address system allowing everyone in the room to hear the computer's output. Thus, one talking terminal would do the job of many conventional terminals.

It is hoped that the design and construction of the prototype talking computer terminal will enable blind programmers to be more self-sufficient and productive and will find many applications among sighted users.

REFERENCES

1. Bliss, J. C., "A Relatively High-Resolution Reading Aid for the Blind," *IEEE Trans. Man Machine*.
2. Bliss, J. C., M. H. Catcher, C. H. Rogers and R. P. Shepard, "Optical-to-Tactile Image Conversion for the Blind," *IEEE Trans. Man-Machine System*, Vol. MMS-11, March 1970, pp. 58-65.
3. Loeber, N. C., "Proposed Braille Computer Terminal Offers Expanded World to the Blind," *Proc. AFIPS Fall Joint Comp. Conf.*, Vol. 39, 1971, pp. 79-87.
4. Homsby, T. G., Jr., "Voice response systems," *Modern Data*, November 1972, pp. 46-50.
5. Liberman, A. M., F. S. Cooper, D. P. Shankweiler and M. Studdert-Kennedy, "Perception of the speech code," *Psychol. Rev.*, Vol. 74, 1967, pp. 432-461.
6. Lee, F. F., "Reading Machine: From Text to Speech," *IEEE Trans. on Audio and Electroacoustics*, Vol. AU-17, No. 4, Dec. 1969, pp. 275-282.
7. Allen, J., "Machine-to-Man Communication by Speech, Part II: Synthesis of Prosodic Features of Speech by Rule," *Proc. AFIPS Spring Joint Comp. Conf.*, Vol. 32, 1968, pp. 339-344.
8. Allen, J., "Reading Machines for the Blind: The Technical Problems and the Methods Adopted for Their Solution," *IEEE Trans. Audio and Electroacoustics*, Vol. AU-21, No. 3, June 1973, pp. 259-364.

9. Gerstman, L. J., and J. L. Kelly, "An Artificial Talker Driven from Phonetic Input," *Journal of Acoustical Society of America,* Vol. 33, 1961, pp. 835(A).
10. McIlroy, M. D., "Synthetic English Speech by Rule," Bell Telephone Laboratories, March 1974.
11. Rabiner, L. R., and R. W. Schafer, "Digital Techniques for Computer Voice Response: Implementations and Applications," *Proc. IEEE,* Vol. 64, No. 4, April 1976, pp. 416–433.
12. Rahimi, M. A., and J. B. Eulenberg, "Modes of Information Presentation for the Blind Programmer," *Proc. Assoc. for Computer Machinery Annual Conf.,* 1974.
13. Rahimi, M. A., and J. B. Eulenberg, "A Computing Environment for the Blind," *Proc. AFIPS National Computer Conference,* Vol. 43, 1974, pp. 121–124.
14. *Votrax Audio Response System Operators Manual,* Vocal Interface Division, Federal Screw Works.

Hard-copy computer output and its future

by IRVING L. WIESELMAN
Dataproducts Corporation
Woodland Hills, California

ABSTRACT

Descriptions provided for classes of products available for hard-copy output of computers include: impact and non-impact printer technologies; serial character and parallel line printers; shaped character and dot matrix character images; and plotters which use movable pens or dot matrix imaging with non-impact printing technologies. The significance of factors such as print quality, speed, flexibility, reliability and cost, which determine the selection for a given application, will be discussed.

The future of these products depends on the utilization and enhancement of microelectronics, new materials and new manufacturing processes, as well as the needs of users in the marketplace. New products will have even higher price/performance ratios, better reliability and increased output flexibility and graphic capabilities.

INTRODUCTION

The principal hard-copy output of computer systems is alpha-numeric data produced on computer output printers which range in speed from 10 cps (characters per second) to 18,000 lpm (lines per minute), and the prices range from $1,000 to $300,000.[1] The two basic methods of printing are impact and non-impact. Impact printers utilize mechanical pressure to transfer the character image from an inked ribbon to the paper. Non-impact printers utilize other technologies such as thermal printing where the character image is formed by using heating elements with the shape of the character to heat a sensitized thermal paper which changes color with applied heat. Printing is performed either a character at a time in a serial format by a character printer, or a line at a time in a parallel format by a line printer. Character images are printed either as shaped characters, which is the familiar form of book printing, or as a set of dots representing the character out of a dot matrix pattern. Thus, a printer is classified by three characteristics: impact or non-impact, character or line printer, shaped or dot matrix characters. Table I lists printer speed classes with the associated types of printers and typical U.S. end user price ranges when interfaced to a computer.

The three general types of plotters in use are the drum, the flat-bed and the electrostatic printer/plotter.[2] It is also possible to use certain types of computer output printers for plotting if the printed symbol spacing can match the plotting requirements. The ink-based plotters contain means of moving the pen in two dimensions across the paper. The flat-bed plotter uses a pen which is moved in two dimensions. The drum plotter utilizes paper motion for one dimension and moves the pen for the other. Electrostatic plotters move the paper for one dimension and use a set of styli across the paper which is selected to cause printing on dielectric-coated paper. Plotters are utilized for hard-copy whenever graphical data is the prime output of the computer.

The applications for different speed classes of output printers have been dependent on the volume of output required and the price of the printers. The most widely used printers are the low cost 10 cps (characters per second) character printers used primarily in terminals. Higher speed character printers, with speeds up to 200 cps, are also used with minicomputers and small business systems. Medium to high-speed printers in the 300 to 2000 lpm class represent the major output devices for small, medium and large computers. Very high-speed printers in the 4000 to 18,000 lpm class are utilized by very large computer systems with volumes of printout greater than 1 million forms per month.

The particular choice of a hard-copy output device for a given system depends on the application, the print quality and/or graphic output requirements, the volume of the output, the speed of data availability and output speed capabilities, the flexibility and availability of output character sets and symbols, system costs, expendable costs and, finally, the maintainability and reliability. The significance of these factors as they affect the choices will be discussed.

IMPACT PRINTING

Printer units consist of a printing mechanism which produces the printed characters and a paper moving mechanism which moves the paper past the printing mechanism. Impact printer mechanisms consist of some means of scanning the characters past the printing positions, some means

TABLE I.—Typical U.S. End User Printer Equipment Prices and Speeds

IMPACT CHARACTER (SHAPED CHARACTER)				
Speed—Characters/sec.			10-55	60-120
Price—$(000)			1-6	4-7
IMPACT CHARACTER (DOT MATRIX CHARACTER)				
Speed—Characters/sec.			30-100	115-660
Price—$(000)			2-8	3-12
IMPACT LINE (SHAPED CHARACTER)				
Speed—Lines/min.	90-250	300-700	800-1800	2000
Price—$(000)	3-17	3-51	22-87	87-102
IMPACT LINE (DOT MATRIX CHARACTER)				
Speed—Lines/min.			125-500	
Price—$(000)			4-10	
NON-IMPACT CHARACTER (DOT MATRIX CHARACTER)				
Speed—Characters/sec.			10-300	
Price—$(000)			1-5	
NON-IMPACT LINE (DOT MATRIX CHARACTER)				
Speed—Lines/min.		300-600	1000-3600	4000-21,000
Price—$(000)		5-10	4-25	145-310

of impacting the character to transfer the character image through the inked ribbon to the paper and a ribbon moving mechanism. There are many techniques of scanning the characters past the print station, and these will be discussed with individual printer types.

There are two basic methods of impacting the character. One method is to impact the ribbon and paper with the character to be printed, as is done with an ordinary typewriter. This is known as a front-striking character impact mechanism. The other technique is to place the character to be printed behind the ribbon with the paper in front of the ribbon. The character image is formed when the hammer impacts the paper causing pressure on the ribbon and the character. The characters are scanned past the hammers and are moving when impact occurs as shown in Figure 1. The latter hammer-impact technique is the one employed in all drum, chain, and train printers and this represents the majority of the installed impact medium and high-speed printers.

The ribbon mechanism uses either a wide towel-like ribbon which passes the print station in the vertical direction, or a narrow ribbon like a typewriter which passes the station horizontally. One form of the paper moving mechanism consists of utilizing paper with sprocketed edges so that a sprocket-feed mechanism moves the paper. The friction feed rollers are normally used in low-speed printers or in very high-speed printers which do not intermittently stop for printing. Printing of forms is normally accomplished by means of sprocketed paper since the form must be aligned with the printer mechanism to print properly on the form. The two basic sizes of printers permit 80 column or 132 column print widths at 10 columns per inch. Higher print densities are available on some printers which permit 132-column printing on 11-in. wide forms.

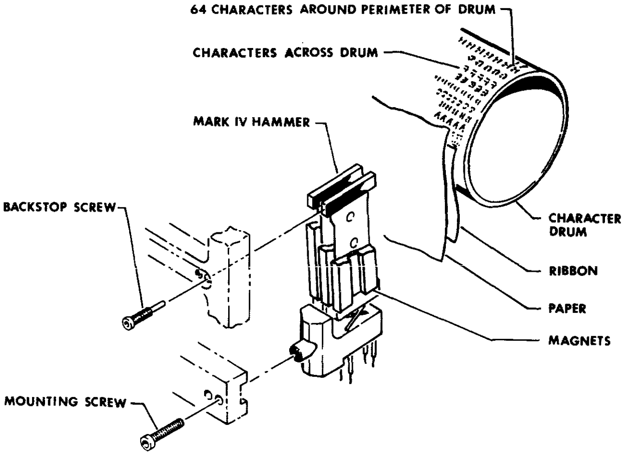


Figure 1—Dataproducts Mark IV hammer impact technique on drum printer

Printer units also contain control electronics and power supplies. There is an interface to the computer or to the control unit which is used to transfer data to be printed and format control information. The control electronics contain timing control, electromechanical control, power circuits such as hammer drivers and, if required, code translators. Character printers generally accept a character at a time, and the printer mechanism prints the character and moves to the next column to print the next character accepted. Line printers contain print line buffers. All the characters for a print line are sent to the printer and they are stored in the buffer. The printer control electronics scan the buffer and actuate the hammers, at the correct time, to print the line. The power supplies provide power to operate the control electronics and to operate the electromechanical portions of the printer.

Impact serial character printers (shaped characters)

The printing mechanism of a serial character printer moves serially across the printing area, usually from left to right, as the characters are printed a column at a time. Both the character impact and hammer impact methods are employed. The majority of serial printers, over 700,000, are the character impact printers produced by Teletype Corporation. Teletype printers use a cylinder with characters on the cylindrical surface to print at 10 cps. IBM uses a replaceable sphere with characters on the spherical surface on their Selectric printers to print at 15 cps. Univac uses the hammer impact method with a one-character-wide drum to print at 30 cps. The Xerox-Diablo and Qume printers use a daisy wheel with characters mounted on flexible arms at the outer perimeter and print at speeds up to 55 cps. These printers can also be used for plotting, since incremental motions of 1/24 in. are possible.

General Electric Terminet printers use a belt on which the characters are mounted. Hammers behind the belt impact the characters and then, in turn, impact the ribbon and paper. It is not really a character printer since the print mechanism does not move across the print line. Some versions are character printers since the hammers fire in groups from left to right, while other versions are line printers up to speeds of 340 lpm.

Impact serial character printers have been utilized in two different types of applications. The difference in the application depends on the print quality obtainable with the printer. Printers in the teletype class, such as the Teletype Model 33, have been primarily used in communication situations where print quality is not very important and low cost is a desirable feature. On the other hand, the IBM Selectric and the daisy wheel printers are utilized for printout where good print quality is a requirement, such as the output of word processing equipment for business letters.

Impact dot matrix printers

Most of the printers in the field using impact dot matrix printing are character printers, but there are some which are line printers. Most character dot matrix printers use a 7-wire matrix to produce matrix characters ranging from 5×7 to 9×7 dots. The use of an 8-wire or a 9-wire matrix permits character generation for characters which extend below the print line. Many different character sets can be generated with the same print head using different dot matrix character generating ROM's (read-only-memories). Character sets with different shapes than English, such as the Japanese Katakana, may be generated by merely changing the ROM.

Recently, Centronics and others have introduced printers with ROM's to generate high density character sets of up to 15 cpi (characters per inch), instead of 10 cpi normally used for computer printout. In this way, a 132-column printout can be produced on an 11-in. width sheet. Plotting may also be performed by dot matrix printers using the dot grid available from the print mechanism. The plotting capability requires additional electronic control and software, and is available on very few matrix printers.

Several line printers in relatively low production volume are available which use dot matrix printing. There are two basic approaches used for printing. The first approach, used by Tally, Okidata and Printronics, uses a matrix comb of wire actuators across the page which moves back and forth to print dots for one row of dots across the sheet. The sheet is moved to print the 7 or 9 dots vertically to generate characters. Potter manufactures several models with speeds up to 500 lpm which utilize a rotating print drum consisting of a cylinder with helical ridges on its periphery. The impact hammer, which is several columns wide with the height of a dot, is used to print all the dots for the columns it covers. The impact of the hammer on the paper and ribbon at the time that the ridge is at an appropriate position, causes the dot to appear.

The use of character dot matrix printers has greatly increased in the past few years. They permit printing at lower prices for applications between the 30 cps shaped character printers and the 100-300 lpm low-speed line printers. A whole new series of products is just emerging in the 30 cps to 100 cps speed range with generally higher prices than the Teletype Model 33, but with speeds at least tripled and with reliability greatly increased over the Teletype printer. Higher speed models use multiple heads to increase speeds. The prices for these printers in the 60-120 lpm speed range have been priced lower than line printers in the 100-300 lpm range. Although the dot matrix printers are lower priced than line printers, they generally suffer from lower reliability and require more maintenance.

Impact line printers

The hammer impact class of line printers constitutes the major population of medium and high-speed printers. Technology improvements have increased the speed from 600 lpm in 1960, to 2000 lpm in 1972. In addition to increasing speeds, the prices for equipment have been reduced and the reliability increased. Although printing appears to occur a line at a time, it actually occurs by sets of characters being printed simultaneously on a line and eventually all the sets of characters get printed to form a line. The print mechanism contains one hammer per column of print position. The printable character set is scanned past the hammers. When a character reaches the column to be printed with that character, the printer control causes the hammer to impact the character at that column.

The two basic methods for moving character sets past the hammers are the horizontal moving techniques and the vertical moving drum. IBM pioneered the chain- and train-drive printers with its 1403N1 at 1100 lpm. while the majority of non-IBM printers use drums. Chains and trains use slugs constrained by tracks which need lubrication to keep friction down and to minimize wear due to sliding friction. Drums are more reliable since they are rotated on bearings and have no friction in moving the character set. Chains and trains are more flexible since character sets can be larger and can be changed by the operator. Drums, on the other hand, last longer because 35 times as many of the same characters are used for printing.

The other approaches to horizontal scanning use belts which move in one of two planes. The first moves like the tread of a tractor and is used by the Teletype Model 40 and the Dataproducts Charaband. The second moves like a belt that wraps around one's waist and is used in the G.E. Terminet and the IBM and Univac steel belt printers. These belt mechanisms vary in the amount of friction encountered due to the basic design approach. The Dataproducts Charaband uses print slugs attached to a band riding on a ball-bearing mechanism which minimizes friction and wear and, thus, eliminates complex lubrication systems and increases life. Printers using belts have speeds which range from 30 cps to 2250 lpm.

Printer performance is measured by print quality, flexibility, speed, reliability and maintainability.[3] The principal factors which determine print quality are horizontal and vertical character registration, character smear, character tilt, character clipping, ghosting, character voids, and variations in character density. These print quality factors are determined by the design of the print mechanism, the type of ribbon used and the ability of the print mechanism to stay adjusted. Designs such as Dataproducts printers use components, such as the Mark IV friction-free hammer mechanisms, which retain their adjustment for over 150 million printed lines.

The speed of line printers is a function of the character scan speed, the size of the character set, the characters printed on the line and the time it takes to move the paper to the next line. The speeds of line printers vary from 75 to 2000 lpm with increments generally of 300 lpm. The most widely used speeds are 75, 150, 300, 600, 900, 1200, 1500 and 2000 lpm.

The last factors of reliability and maintainability are influenced by the general technology used and the mechanization of the principal subsystems consisting of the character scanning mechanism, the paper feed mechanism and, most important of all, the hammer impact mechanism. Most manufacturers use an electromagnet and push rod which impacts a pivoted or flex-pivoted hammer slug. Parts of the electromagnet mechanism are subject to wear as well as the push rod-slug interface.

The Dataproducts friction-free Mark IV hammer actuator consists of the hammer impact slug which is mounted on a flat coil and suspended by two flex-pivot springs which also carry current to energize the coil. The coil is placed between permanent magnets and, hence, an electromagnetic field is produced which causes the hammer to impact the paper, ribbon and the character to be printed. Field experience indicates that the mean number of strokes between failures for the hammer actuator is approximately 2,000 million, and flight time adjustments are required at about 150 million strokes. It is a more reliable hammer mechanism and requires less maintenance than other designs.

Impact line printers—somewhat serial

This class of printer is used for medium to low-speed line printing in the range of 100 to 700 lpm. Lower product cost is achieved than is possible with a line printer by sharing some of the components in the printer, hence, fewer parts are used.

Dataproducts pioneered this approach in 1969, and produced its 80-column Model 2310 printer by sharing 20 electrical hammer drivers for 80 hammer actuators. This same sharing technique is used in the Dataproducts 2910 military line printer with speeds ranging between 356 and 1110 lpm depending on the number of columns printed. IBM uses a double width hammer in its new steel band printer for the 3770 terminals and the System/32 at speeds between 50 and 150 lpm. One hammer is used for two columns, hence, only one-half the hammer mechanisms are required. Another version of IBM's steel band printer is available with one hammer per column which prints at 400 lpm.

Dataproducts offers the 2230 printer which uses one-half the number of hammer actuators to achieve a print speed of 300 lpm. The hammer bank with the Mark IV friction-free hammer, is flex-mounted and servo-driven to two positions by means of a voice-coil positioner. An optical transducer is used for position sensing and a magnetic transducer is used for velocity sensing.

NON-IMPACT PRINTERS

A variety of technologies exists for printing with non-impact printers. Some utilize special papers such as electrostatic, electrolytic, photographic or thermal, and others utilize ordinary paper. Printing speeds range from 10 cps to 45,000 lpm. All impact printers have the property that they can produce multiple copies at the time of printing, since the impact pressure through the carbons is transmitted to the copies. Non-impact printers produce only one original copy.

The two technologies of non-impact printing which utilize normal paper are the ink-jet and xerographic printers. Low speed ink-jet printers are used in word processing applications. High-speed xerographic printers are used for high volume printing. These printers are just emerging in the marketplace.

The other techniques of non-impact printing employ special papers which have properties for producing print without impact. The most widely used is thermal paper which is printed upon by heating dot elements which form dot matrix characters on the paper. Thermal printers utilizing this technique are employed as interactive keyboard terminals. Next in use is electrostatic printing which is performed by using a paper which is dielectric coated and used in applications which require printing and plotting.

Each of the technologies will be discussed with the printers' speed range, advantages and disadvantages, price range and significance for use as hard-copy output devices.

Thermal matrix printers

Most of the thermal printers in the field utilize an array of 5×7 individual elements within one head. Each element in the array can be switched on and off to impart heat. The head stops at each printing position as it is stepped across the paper. Dark marks are produced on the thermally sensitive paper at those points of the array which have been heated. Speeds of up to 120 cps are available using this technology, but most of the printers operate at 30 cps.

The prime suppliers of thermal printers are Texas Instruments and NCR, with TI being the dominant supplier. Until recently, impact dot matrix printers have been considerably more expensive than thermal printers. The new 30 cps impact printers of Digital Equipment Corporation and GE are still more expensive than the thermals, but they are closer in price than they were. Thermal printers are also being utilized for calculators and small personal computers. They are quiet and have a lower initial cost than comparable impact printers. If the volume of printing is not high, then the cost of the paper is not a deterrent for its use. Paper costs can be two to four times the cost of normal paper.

Electrolytic and electrographic printers

Both processes use specially coated papers which change in color with the application of a voltage on the writing element. The electrolytic process is a wet process where moist paper is drawn between electrodes. The electrographic process is a dry process which uses electrosensitive paper which has a metallic sheen and retains finger mark impressions when handled. Applying a voltage to the paper causes a light surface layer to burn away and leave a dark layer underneath. Both processes have been used in facsimile systems, military communication equipment and commercial terminal equipment.

The printing mechanism in commercial terminal equipment, such as the UNIVAC printer, uses a dot matrix print head containing 9 styli that etch 7 columns of dots per character as the print head moves across the page.

The technology provides fairly low printer prices with fairly high serial printer speeds, up to 300 cps. The principal problem with utilizing the technology is the appearance of the paper and its expense. The approach could be used for output plotting at the plot densities available from the head configuration. Further developments in the paper technology could make this approach more viable in the marketplace.

Electrostatic printers

Electrostatic printers utilize specially coated paper and are line printers, since a row of conductive stylii are used with a density of between 100 and 200 stylii per inch.[4] Each stylus is selectively charged according to the required output, so that each character is formed out of a mosaic of charged spots on the paper. The data for successive rows are produced as the paper moves. The paper is subsequently passed through the toner bath where the charged areas attract ink particles. The appeal of the technology stems from its ability to both print and provide relatively rapid plotting. Printing speeds vary between 300 to 3600 lpm for most printers, except for Honeywell which has a speed of 18,000 lpm. The prices are quite competitive with the fastest impact printers and often are significantly lower. Very fine resolutions are possible, which makes a variety of good quality character styles available.

Three vendors are supplying printer/plotters: Versatec, Varian and Gould.

Versatec is the dominant supplier. A new printer has emerged which is low cost for printing only and is manufactured by Houston Instruments. Honeywell has a high-speed version of this technology printing up to 18,000 lpm. The prices range between $5,000 to $13,000 for lower speeds, and up to $165,000 for the high-speed Honeywell printer.

The principal disadvantage of this technology is the cost of paper. If very high printing volumes are expected, then the cost of paper could be prohibitive to its use. Honeywell sells their paper at a fairly low price, but charge for use of the printer on a per copy fee. When these printers are used as printer/plotters, then the cost of paper is not a significant factor since the convenience of the output is the prime driving force for its use. Plotters are available with widths up to 72 inches using this technology.

Magnetic printers

One currently available printer, manufactured by Inforex, uses a technology that can be described as indirect magnetic. In this printer, a tape coated with magnetic material is passed over a recording head, which creates a magnetic latent image of a complete character mosaic. This tape is then toned with a magnetic ink powder. When a full line of text is ready, the tape is placed in contact with the paper and the ink particles are transferred. The ink powder is fused into place as the tape is wound on past an erasing head. Speeds of 200 lpm can be achieved in this way. It is being manufactured in fairly low volume, and has poor printing quality. It is not too cost effective and is not a significant technology in the marketplace today. Future developments could make this technology more viable.

Ink-jet printers

Ink-jet printers produce a jet of ink droplets which are directed against plain paper. The droplets first pass through an electric field which places an electrostatic charge on the droplet. They next pass through a deflection plate which deflects the droplets in proportion to their charge. At points where no ink mark is required, the ink droplet is deflected into a gutter, leaving the paper clear. There are two ink-jet printers in the marketplace today at opposite ends of the speed spectrum. Ink-jet printing technology is also used in a variety of applications outside of the data processing industry, such as the printing of containers.

The IBM 6640 document printer produces output at 92 cps with print quality that is very close to that produced by the IBM Selectric typewriter, and is used in word processing applications. This print quality is achieved by utilizing a dot matrix structure of dots at 240 per inch both horizontally and vertically. A single ink jet is used which is directed vertically and moves horizontally in a serial manner. Although the 6640 does not currently have a graphics plotting capability, it certainly would be possible to provide very good quality graphical output provided the control were designed into the printing mechanism. This certainly could happen in the future.

The Mead Dijit printer system produces printout at 45,000 lpm, or 600 feet per minute, in terms of press speeds. The print head utilizes 100 jets per inch and a single jet for each droplet position. It is an expensive printer, and is currently being used in specialized applications such as very large volume direct mail letters. Its utilization is closer to that of a high-speed printing press with variable information being printed as controlled from a computer output.

More ink-jet printers will be appearing in the marketplace in the future. Higher speed serial printers will be available with lower print quality than the IBM 6640. The success will ultimately depend on how reliable and maintainable the printers are, and how cost effective they are in relation to comparable impact printers. Specialized applications will be found for the very high-speed printer, but it will not be used widely for high-speed computer printout, since other methods which are somewhat slower are available and are cost effective in the marketplace.

Xerographic printers

Xerographic printers utilize a printing process identical to that utilized by xerographic copiers. The difference between a xerographic copier and a xerographic printer is the method of imaging. The Xerox 1200, first delivered in 1974, utilizes a photographic drum with character images similar to a line printer drum, but uses flashes of light to image the characters on the xerographic printing drum. The IBM Model 3800, first delivered in 1976, uses a laser beam character generator to generate a dot matrix character on the surface of the photoconductor. In both cases, after the images are formed, the toner is applied to the photoconductor surface and the image is created by the toner on the surface. The image is then transferred to output paper and fused into place. Both systems contain a forms overlay feature which allows a form created as a photographic image to be reproduced on the printed output. This eliminates the need for utilizing forms which are pre-printed and saves the costs of purchasing forms printed by a printing press.

The Xerox 1200 uses 8½×11 in. sheets, prints at 13.3 characters/inch and 8 lines/inch, at a speed of 60 sheets/minute, or 4000 lpm. It prints 95 ASCII symbols including upper and lower case alphabetics, numerics and special characters. The cost is $145,000 and a usage charge, based on the number of copies, is added to take care of maintenance.

The IBM 3800 utilizes continuous sprocketed forms and a dot matrix structure for its output characters, with 180 dots/inch in the horizontal direction and 144 dots/inch in the vertical direction. The dot structure is fine enough to produce output characters which appear to be shaped character images. The dot matrix structure permits many different character sets to be printed by the printer with the possibility of mixing character sets on a particular sheet.[5] There is no graphic capability in terms of graphical curves, but it would certainly be possible to provide that capability with suitable software modifications by IBM. The output speed depends on the size of the form and the number of lines per inch utilized in the printout and ranges from 8,180 lpm to 20,820 lpm. The cost is $310,000, and there is a maintenance charge based on the number of forms used per month.

In order to utilize a printer with a speed capability and with the cost of the IBM 3800, the user should have a volume greater than 1½ million copies of printout per month. The pricing of the printer is such that it becomes cost effective if the user does have volumes this high or greater, and utilizes a considerable amount of pre-printed forms. Computer operations which utilize more than 3 high-speed computer output printers, are candidates for these high-speed non-impact xerographic printers.

DIGITAL PLOTTERS

Digital plotters provide graphical output in the form of lines on paper from digital inputs. Digital plotters utilize writing instruments, such as pens to mark the paper, which are moved in two directions to generate the graphical output. Another type of plotter is available, which utilizes the technique described under electrostatic printers.

There are two basic types of moving pen plotters: the flatbed plotter and the drum plotter. The flatbed plotter plots on a flat sheet of paper which is held in place, and the pen is moved in two dimensions to draw lines from one point to another. Roll sheets can also be used which consist of paper, vellum or plastic. The paper sizes range from 11×17 in. to 54×76 in. Pen speeds range from 3 in./sec. to 100 in./sec. The number and types of pens range from 1 to 8 including liquid ink, ball point, fiber tip and scribe. The prices range from $4000 to $200,000 depending on the size of the equipment, number of pens, flexibility and accuracy.

Drum plotters utilize paper motion to provide one direction of motion and the pen to provide the other. In addition, the drum can be moved in either the forward or reverse direction to achieve complex graphical output. Very long graphical outputs can be achieved by using roll sheets and continuing the drawing over many sections. Paper sizes vary from 11 in. to 30 in. in width. The plotting speeds range from a minimum of 200 steps/sec. to a maximum of 5000 steps/sec. The types of recording media used and pens are similar to that which is used with the flatbed plotters. The number of pens ranges from 1 to 4. The prices range from $3,500 to $23,000. The majority of plotters used in the field are drum plotters, since they are much lower priced than the flatbed plotters. Very large drawings, such as are used for drafting applications, are usually handled by flatbed plotters.

Electrostatic plotters use a row of styli in the horizontal direction to produce dots. These range in density from 80 to 200 dots/in., and plotting widths range from 8½ to 72 in. Vertical plotting is performed by moving the paper as in a drum plotter. The plotting capability is similar to that which could be achieved by using as many pens as are needed for a drum plotter. The principal advantage of these plotters is speed. The horizontal plotting speed is one scan in the time it takes to move one vertical increment. Since paper speeds vary from 0.5 in./sec. to 7 in./sec., the horizontal plotting speed can be as high as 6000 in./sec. The time for plotting is not dependent on the complexity of the plot, but only on the paper speed. For pen plotters, the speed also depends on the pen velocity, the number of pens, and the complexity. Prices for the printer/plotter mechanism range from $6,000 to $52,000. Complete systems which interface to computers add from $2,000 to $12,000. Dielectric-coated paper ranges from two to four times the cost of plain paper.

The required physical characteristics of the graphical output produced by plotters are described in terms of the accuracy, resolution, and repeatability of the device. The accuracy of the plotter is the error between where the points should be and where they actually are. Accuracy is expressed either in terms of a percentage of the total span of the graph, or in terms of absolute accuracy in inches. Accuracies range from .001 in. to .012 in., and from .05 percent to .1 percent vertically, or from half a step to a single step vertically. Resolution determines the minimum dimension which can be drawn and the minimum distance between lines. This ranges from .001 in. to .005 in., or from 50 points/in. to 200 points/in. Repeatability refers to the ability to return to a previously plotted point. The ranges for repeatability are based on the resolution and accuracy, and are generally closer in value to the resolution than to the accuracy.

The method of plotting for pen plotters depends on the ~~command structure available in the software and can effect~~ the quality of the output plot. One of the most widely used methods is to move the pen one increment to the right, the left, up or down, or diagonally right-left-up, or diagonally right-left-down. This is a total of 8 different kinds of motion which are possible. Another approach uses dots placed at given locations determined by coordinate addresses. A third method utilizes straight lines drawn between two specified points. The quality of the final output depends on the resolution of the system and the kinds of commands which are available. If the resolution is fine enough, the jagged appearance of curves looks smooth to the viewer.

There are a multitude of choices available to the user. Once the requirement for the graphical output is decided upon, one can find a plotter which meets the needs. The size and type of paper needs to be specified, the number of pens required (perhaps with different colored inks), the accuracy, the resolution, the repeatability and, finally, the speed. If there are incompatible requirements, such as a speed requirement which is too high for the capability in terms of resolution, then judgments must be made as to what parameters are really the most important.

THE FUTURE

The characteristics of various alternatives to obtaining hard copy from computers have now been discussed. The changes in price/performance for existing products in the past few years have been primarily due to improvements in large scale integration (LSI), new materials technology, and in production automation. New products using new technologies, such as the ink-jet printer and the laser beam electrophotographic printer, are based on the refinement of new inventions and concerted investment in new development.

The expanded use of LSI and the emergence of microprocessors has not only made dramatic improvements in existing products, but has also affected the way in which we conduct our day-to-day affairs. Electronic hand calculators are available for under $10.00. Microprocessors have been applied to control the automobile engine so that more efficient fuel utilization occurs as well as producing fewer pollutants. The utilization of LSI microprocessors, inexpensive semiconductor ROM's and random-access-memories has caused computer capabilities to be distributed with networks and low cost terminals.

The use of microelectronics in hard-copy output equipment has not only reduced the cost for electronic control, but it has provided more flexibility and the increased use of electronics to perform functions which were previously handled by mechanical or electromechanical equipment. The reliance on a paper tape reader for vertical format control was replaced by Dataproducts when lower cost shift registers became available. These are planned to be replaced by microprogrammed logic to provide additional flexibility. Dot matrix formed characters, which have the appearance of being produced from shaped character im~~pact fonts. are possible due to the lower cost of storage~~. The IBM 6640 uses a dot structure of 24×40=960 dots now, as compared to other dot structures as low as 5×7=35 dots to produce characters.

The possibilities for lower cost production techniques, using more plastics and less metals, are emerging in products for the marketplace. The hammer bank for the Dataproducts 2230 printer, using the Mark IV hammer actuator, has been designed so that the hammers and the magnets are mounted in plastic parts which are attached to tubular segments. New magnetic materials are available which provide magnetic fields which are 50 percent stronger than those currently used. This means reducing electromagnetic fields and, hence, lower currents, less power, and simpler driving electronics. New plastic materials and new epoxies also mean lower cost mechanisms. Hence, new versions of the 2230-type printer will appear with much lower costs.

The emergence of the laser beam as the energy source for character generation has affected both non-impact and impact printing. The very high-speed non-impact printers will use laser character generation, and other printers besides IBM will be available in the next few years. Although high-speed impact printers are available today which print at 2000 lpm, it is expected that future high-speed impact printers will peak at 1500 lpm, and that the higher speed printers will be non-impact printers. The impact printers in the 1000 lpm speed region will continue to be cost reduced as the technology for the cost reduction of the lower speed printers is applied to higher speeds.

The emergence of the IBM 6640 printer is an example of a new product based on technology available today and a concerted investment in new development. Ink-jet printers had been in the marketplace, but they were not successful because of reliability problems and non-competitive price/performance ratios. IBM decided to invest in ink-jet technology and found solutions to many of the problems causing poor reliability. The electronic sophistication required to produce the high quality print of the 6640 could only be possible with the use of microelectronics available today.

The use of finer dot matrix structures for printing has had an effect on graphical output. The use of electrostatic technology for printing or plotting is an already established and accepted technique. Not many impact dot matrix printers have been used for plotting, but they could be if they were modified to handle the task. Both the laser beam electrophotographic technology and the ink-jet technology have the basic capability to do a respectable plotting job comparable to drum plotters. Whether or not these techniques will be used for graphical output depends on the demands of the marketplace. If there is a real need which could develop into a sizable market, then manufacturers will commit their resources to the developments required to satisfy the need.

In conclusion, the future will provide improved products based on the printing technologies used today with even higher price/performance ratios, better reliability, and increased output flexibilities. Additional graphical capabilities exist in many forms of output equipment, but they will not be made available to users without sufficient pressure from them. Impact printers will still dominate the marketplace, but there will be increased use of non-impact printers for specific applications.

REFERENCES

1. Wieselman, Irving L., "Printer Technology And Its Future," *Modern Data,* November 1975, Vol. 8, No. 11.
2. Datapro Research Corporation, "All About Digital Plotters," *Datapro 70,* January 1977, Delran, New Jersey.
3. Freund, Ken, "Make No Mistakes When You Buy Your Next Printer," *Electronic Products,* April 28, 1975, Vol. 17, No. 11.
4. Bakey, Tom, Jill Peters and David Sloan (Varian Staff), *Printer/Plotter Considerations,* Varian Data Machines, Palo Alto, California, 1972.
5. International Business Machines Corporation, *Introducing the IBM 3800 Printing Subsystem and Its Programming,* GC26-3829-4, File No. S370-03, White Plains, New York, 1976.

Variable-length hash area entries

by M. H. McKINNEY
University of Southwestern Louisiana
Lafayette, Louisiana

ABSTRACT

A study is presented of the behavior of hashing when implemented with variable-length entries in a hash area. Neither bucketing nor pointers are necessary to achieve the variability. Significant space savings may be realized by reducing the amount of space required for each entry while providing for entries of any length. Savings in time may also be realized by improving the locality of reference. Simulation results are presented which demonstrate the potential savings along with some interesting behavioral aspects of using a variable-length entry approach. Variability by a fixed block size as well as variability by atom is discussed. An algorithm to accomplish the placement of variable-length entries in the hash area is given.

INTRODUCTION

Hashing involves transforming the numeric representation of some key value into an address such that the address corresponds to the location within a storage area (hash area) at which the key value and accompanying non-key data are stored. Here, the term "key" carries the conventional meaning, i.e., a *key* is the attribute by which an entity is uniquely identifiable. When two keys hash to the same address, a *collision* is said to occur. Collisions are to be expected when hashing and must be handled by a *collision recovery algorithm*. This may be handled in a number of ways, although the simplest such collision recovery algorithm is the *linear probe open address* (linear rehash) method.[1] In the event of a collision, the linear rehash method simply advances repetitively to the next higher contiguous location in the hash area until the desired location is found. The frequency of collisions depends on the distribution of addresses generated by the hashing algorithm, the collision recovery algorithm used, and the denseness of occupancy of the hash area. The measure of hash area denseness is called *loading*, which is defined as the percentage of occupied space in the hash area.

Although hashing has received considerable attention in the literature,[1-7] placement of data directly into the hash area is discussed with reference only to fixed-length partitions of the hash area such that the partitions are either the same size as the fixed-length logical entries or large enough to accommodate multiple logical entries. The term *bucketing* is used to describe the latter approach with the fixed-length partition being termed a *bucket*. Thus, with bucketing, collisions are no problem until the bucket is full.

Accommodation of variable-length entries is generally accomplished by one of three methods:

(1) Utilizing fixed-length partitions in the hash area such that the length of the partitions is equal to the length of the longest variable-length entry.
(2) Utilizing bucketing such that each bucket may accommodate one or more variable-length entries.
(3) Utilizing a hash area containing only pointers to the variable-length entries which are stored elsewhere as a dense list. This is often referred to as the *scatter index table* method.[2]

The first and, generally, the second method require predetermination of the maximum entry length of all the entries. In addition, these methods may render an appreciable amount of space unusable due to internal fragmentation within the partitions. The second method also requires a directory within each bucket to indicate its contents.

The scatter index table method requires a hash area for the pointers and another area for the data entries. With a random file residing on a direct access storage device, one physical access may be required to reference the pointer area with another to reference the dense list area. Collisions may cause several such dual accesses before the desired location is reached. A similar situation exists when the hash area and dense list are maintained in a paged virtual memory system. Each reference may well cause a page fault.[2]

The variable-length entry approach, by reducing the amount of hash area space required for each entry and by using a linear rehash, maximizes locality of reference. Within the context of hashing, *locality of reference*, or simply *locality*, is defined as the distance beyond the initial probe address at which the entry is actually located. Locality is used as the primary performance metric in this paper. Certainly, locality of reference is desirable in a paged system as well as when performing random access to files on direct access storage devices.

DESCRIPTION OF THE ALGORITHM

The term *atom* is defined as the smallest addressable unit of information (byte, word, or array element) as is appropriate to the particular application. A *block* is a fixed-sized grouping of one or more contiguous atoms. A *segment* is a contiguous group of blocks containing an entry. The *entry length* is defined as the number of blocks in the segment containing the entry. A *free block* is a block of unused space. A *descriptor* is an atom within each block that describes the contents of the block. The algorithm as shown in the appendix uses the convention that a descriptor of value zero implies a block of free space, while any non-zero value implies that the block is part of a segment. The algorithm also uses the convention that the descriptors of each block of a segment indicate the number of remaining blocks in the segment. For example, the first descriptor of a five block segment contains the value five, the second descriptor contains the value four, etc.

The most fundamental approach involves variability by a block size greater than one atom, although atomic variability is possible with a few restrictions. Both types of variability are discussed below.

Transformation of an entry's key to an address in the hash area may be performed by any of the well-known hashing methods[1,3] with the restriction that the hash address (location of the initial probe) references an atom containing a descriptor. Thus, if the descriptor occupies the first atom of each block, and there are m blocks in the hash area with n atoms per block, the range of the hash algorithm output, h, must be from 0 to m−1. The hash address is then h*n+1, with the hash area ranging from atom(1) through atom(m*n).

Adding new entries

The methodology for insertion of new entries into the hash area is as follows. Upon computation of the hash address from the entry key, the atom at the hash address (a descriptor) is inspected to determine whether or not the block is free. If the block at the initial probe is free, a forward scan is made to determine if there exist enough contiguous free blocks to accommodate the entry. If adequate free space exists, the entry is inserted beginning at the hash address. If there is not enough contiguous free space at, and immediately beyond, the hash address, the segment preventing the insertion is skipped. The skip is made by advancing the forward scan according to the value in the descriptor in the first block of the segment. If the block beyond the skipped segment is itself part of a segment, this collision is also skipped. Skipping continues until either enough contiguous free space is found or some predetermined locality limit is exceeded. In the latter case, one of many auxiliary collision handling methods[3] may be used. If the block at the initial probe is part of a segment, all the remaining blocks of the segment involved in the collision are skipped by advancing the scan according to the value in the descriptor at the initial probe. Then the algorithm proceeds as described above.

Example

The simplified example in Figure 1 shows a hash area containing 20 usable blocks. The hash area is initialized by filling it with descriptors of value zero to indicate free space. The last two blocks shown are used by the algorithm given in the appendix to simplify detection of the end of the hash area during a forward scan, i.e., the free block at location 21 assures that a free block is detected, and the descriptor at location 22 causes the scan to stop and test for a wrap-around before testing for adequate contiguous free space.

The blocks may contain any number of atoms greater than one. If, for example, each of the 20 blocks contains 100 atoms, the actual usable hash area size is 1980 atoms (2000 total atoms minus the 20 descriptor atoms). For simplicity, only the descriptors are shown.

The first entry is a 5 block segment shown hashed to block 9. The forward scan determines that adequate contiguous free space exists beginning at that location, so the segment is inserted there. The next entry is a 6 block segment which is placed in blocks 14-19. It should be noted that this segment is inserted in those blocks if the initial probe is anywhere in the range of block 4 through block 14. If, for example, the entry key hashes to block 4, there exist only 5 contiguous free blocks at and beyond that location. Therefore, when the scan detects the descriptor of 5 for the segment at location 9, a skip is made directly to location 14 by advancing over the number of blocks indicated in the descriptor at location 9. As another example, if the initial probe is made to location 11, a skip is similarly made over the last 3 blocks of the segment to location 14.

The third entry requires 4 blocks and is placed in block locations 1 through 4 if the initial probe is made to any of the blocks 6 through 20 or block 1. If the initial probe is made to any location 6 through 20, the forward scan detects the end of the usable portion of the hash area and the scan wraps around to the first block in the hash area. For simplicity of programming a segment is never split, i.e., segments are always inserted in physically contiguous blocks. The last entry is inserted in blocks 5 and 6 if the hash address refers to any block but 6 or 7.

Location and deletion of entries

The only complication of locating an entry using the variable-length approach arises when the initial probe is made to a block of a segment. In that case, it is not readily apparent whether the block at the initial probe is the first, or some subsequent, block of a segment. This determination, however, is made as follows. If the initial probe is made to a block, h, of a segment, the immediately prior block, p, is examined. If the descriptor at p is numerically greater than the descriptor at h, the initial probe is known

```
             Relative block location in hash area               *  *
01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22
------------------------------------------------------------------
                        Initial values
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1
------------------------------------------------------------------
                      hash to location 9
                         5--4--3--2--1
 0  0  0  0  0  0  0  0  5--4--3--2--1  0  0  0  0  0  0  0  0  1
------------------------------------------------------------------
                   hash to any location 4-14
                                        6--5--4--3--2--1
 0  0  0  0  0  0  0  0  5--4--3--2--1  6--5--4--3--2--1  0  0  1
------------------------------------------------------------------
             hash to either location 1 or any 6-20
 4--3--2--1
 4--3--2--1  0  0  0  0  5--4--3--2--1  6--5--4--3--2--1  0  0  1
------------------------------------------------------------------
                hash to any location but 6 or 7
             2--1
 4--3--2--1  2--1  0  0  5--4--3--2--1  6--5--4--3--2--1  0  0  1
------------------------------------------------------------------
            * dummy blocks for wrap-around detection.
```

Figure 1—Example of the method

to be referencing other than the first block of the segment. Thus, the collision may be skipped without performing any key comparison because a segment is only stored at, or beyond, the initial probe location. Once the skip is made, the scan is synchronized such that thereafter the first block of each segment is directly addressed.

Deleting a segment involves locating it and setting all the block descriptors to zero (free block identification). When frequent delete activity occurs and reorganization is deemed appropriate, the reorganization more closely resembles garbage collection[8] than conventional hash area reorganization.[7]

Modification of an entry such that the modification changes the entry length may be performed by deleting the original entry and then adding the new one. If the new entry is smaller than the original one, the new entry may possibly be placed with better locality than the original entry.

Variability by atom

Atomic variability is achieved by imposing certain restrictions on the data in the segments. An adaptation to byte variability using EBCDIC is described below. Similar adaptation to ASCII character, word, or other atomic variability may be made.

Obviously, there must be some means for determining whether an atom represents free space, a length code, or data so that the scan becomes synchronized with the first byte of a segment. This distinction is made by taking advantage of unused coding combinations in the atoms containing data. For example, there are no printable characters in EBCDIC having a bit configuration numerically lower than 01000000, i.e., decimal value 64. Thus, a length code of 1-63 may be used with no confusion if the segment contents are restricted to character strings. Free atoms may use the bit configuration 00000000. The only differences between such an implementation and the block variability described above are:

(1) With atomic variability, the first, and only the first, atom of each segment contains a length code.
(2) If an initial probe is made to an atom that is neither free nor a length code, an atom-by-atom scan is made to skip over the segment. Once a length code is encountered, direct skipping over segments may be performed.

An alternate approach to atomic variability removes the restrictions on the length codes and the data values. This approach involves imposing the restriction that the length codes must always immediately follow an atom coded as free. Thus, there is no confusion as to which of the atoms represent a length code. Obviously, an extra atom is required for each entry, but length codes may be of any magnitude storable within an atom, and data-containing atoms may contain any bit combination other than that chosen to indicate free space.

SIMULATION AND RESULTS

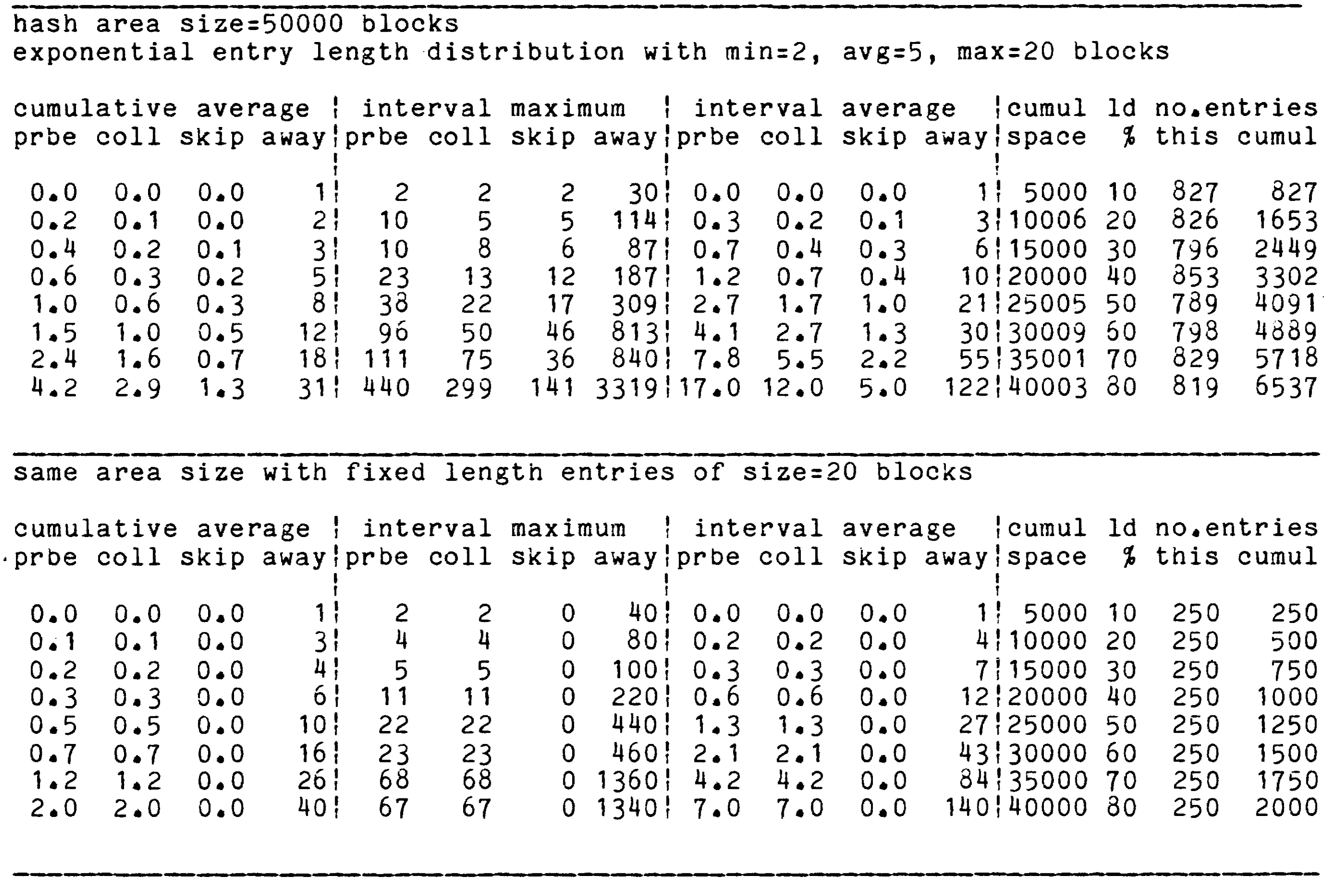
The simulation models are implemented in PL/I on the Honeywell 68/80 MULTICS system. The hash addresses are determined by use of a uniform distribution generator, while the entry lengths are developed by either a uniform, an exponential, or a Poisson distribution generator. When appropriate, the generated entry lengths are truncated such that regeneration is performed when a number outside the specified range is developed. Standard MULTICS random distribution generators are used to develop hash addresses and entry lengths with the exception of the Poisson distribution generator which follows the method described in Reference 9.

Entries are generated and inserted into a simulated hash area using block variability with only one atom per block in order to conserve simulation memory space. Clearly, an increase in block size accompanied by a corresponding increase in hash area size would produce similar statistical results.

Two variations of the simulation model are used for the following analysis. One displays statistics at specified loading intervals, and the other produces statistics at a specified cumulative average locality value. Figure 2 contains part of an actual run of the first version of the model. This model generates entries and inserts them into the simulated hash area until insufficient contiguous free space remains to insert another entry. To reduce confusion, however, Figure 2 does not show the complete run. To provide a direct comparison, the model is rerun assuming the same area size but with all entry lengths equal to the maximum length. This corresponds to a fixed-length entry approach.

It can be seen in Figure 2 that a cumulative average, along with the maximum and average figures for each display interval are produced. Within each of these groups, figures are given on the number of probes, collisions, and skips. The number of skips is the number of groups of free space encountered that are too small to accommodate the entries, and the number of collisions is the number of segments encountered while seeking the requisite contiguous free space. The number of probes is based upon the sum of the collisions and skips for each of the individual entries. The columns headed "away" indicate how many blocks away from the initial probe address the entries are ultimately placed, i.e., this is the measure of locality. Also, for each display interval, the cumulative space used, the loading percentages, the number of entries inserted during the display interval, and the cumulative number of entries are given.

Figure 3 is a plot taken from the complete simulation run



```
hash area size=50000 blocks
exponential entry length distribution with min=2, avg=5, max=20 blocks

cumulative average ¦ interval maximum  ¦ interval average  ¦cumul  ld no.entries
prbe coll skip away¦prbe coll skip away¦prbe coll skip away¦space  %  this cumul
                   ¦                   ¦                   ¦
 0.0  0.0  0.0    1¦  2    2    2    30¦ 0.0  0.0  0.0    1¦ 5000 10  827   827
 0.2  0.1  0.0    2¦ 10    5    5   114¦ 0.3  0.2  0.1    3¦10006 20  826  1653
 0.4  0.2  0.1    3¦ 10    8    6    87¦ 0.7  0.4  0.3    6¦15000 30  796  2449
 0.6  0.3  0.2    5¦ 23   13   12   187¦ 1.2  0.7  0.4   10¦20000 40  853  3302
 1.0  0.6  0.3    8¦ 38   22   17   309¦ 2.7  1.7  1.0   21¦25005 50  789  4091
 1.5  1.0  0.5   12¦ 96   50   46   813¦ 4.1  2.7  1.3   30¦30009 60  798  4889
 2.4  1.6  0.7   18¦ 111  75   36   840¦ 7.8  5.5  2.2   55¦35001 70  829  5718
 4.2  2.9  1.3   31¦ 440 299  141  3319¦17.0 12.0  5.0  122¦40003 80  819  6537

same area size with fixed length entries of size=20 blocks

cumulative average ¦ interval maximum  ¦ interval average  ¦cumul  ld no.entries
prbe coll skip away¦prbe coll skip away¦prbe coll skip away¦space  %  this cumul
                   ¦                   ¦                   ¦
 0.0  0.0  0.0    1¦  2    2    0    40¦ 0.0  0.0  0.0    1¦ 5000 10  250   250
 0.1  0.1  0.0    3¦  4    4    0    80¦ 0.2  0.2  0.0    4¦10000 20  250   500
 0.2  0.2  0.0    4¦  5    5    0   100¦ 0.3  0.3  0.0    7¦15000 30  250   750
 0.3  0.3  0.0    6¦ 11   11    0   220¦ 0.6  0.6  0.0   12¦20000 40  250  1000
 0.5  0.5  0.0   10¦ 22   22    0   440¦ 1.3  1.3  0.0   27¦25000 50  250  1250
 0.7  0.7  0.0   16¦ 23   23    0   460¦ 2.1  2.1  0.0   43¦30000 60  250  1500
 1.2  1.2  0.0   26¦ 68   68    0  1360¦ 4.2  4.2  0.0   84¦35000 70  250  1750
 2.0  2.0  0.0   40¦ 67   67    0  1340¦ 7.0  7.0  0.0  140¦40000 80  250  2000
```

Figure 2—Sample simulation run

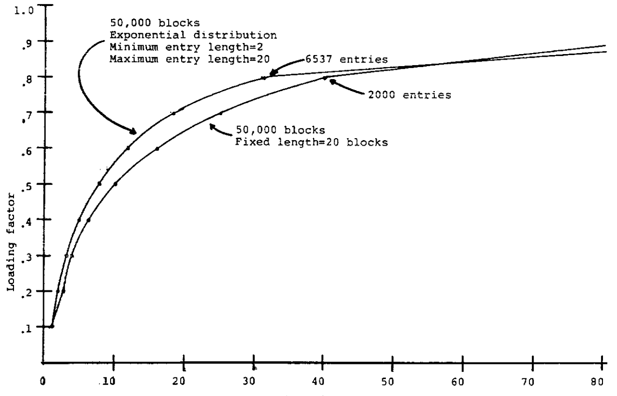


Figure 3—Loading vs. locality

producing Figure 2 showing the loading versus the locality for both the variable-length and the fixed-length approaches. Even though the curves appear quite similar, the actual performance is significantly different. For example, when the loading factor is 80 percent, the fixed-length entry hash area contains 2000 entries, while the variable-length entry hash area contains 6537 entries. This may also be seen by examination of the 80 percent loading (1d percent) columns on the sample output shown in Figure 2. Note that for the fixed-length entry version, the locality is 40 with 2.0 average probes, but the locality is 31 blocks away with 4.2 average probes for the variable-length entry version. This indicates that, along with greatly increased entry storage capability, the variable-length approach also achieves better locality of reference.

Due to the above considerations, a more appropriate measure of performance is that shown in Figure 4 using the plot of the number of entries versus the locality. In Figure 4, the performance of the variable-length approach is seen to be superior to the fixed-length approach even as the average entry length approaches the maximum entry length.

Figure 5 contains a plot of the number of entries versus locality for various values of different distributions of entry length. It demonstrates that regardless of the distribution used, the variable-length approach is superior to use of fixed-length entries when the minimum/maximum entry length ratio is $^1/_{10}$. Examination of relative performance with larger minimum/maximum entry length ratios is described below.

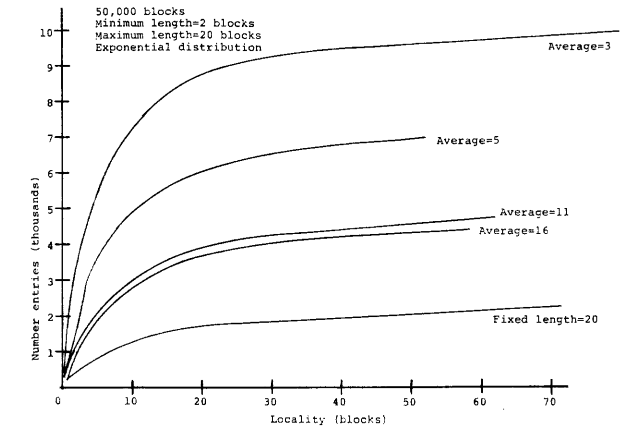


Figure 4—Entries vs. locality

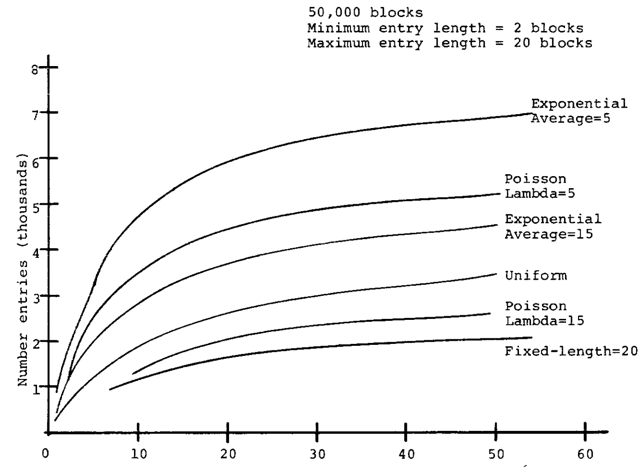


Figure 5—Entries vs. locality by distribution

Figures 6 and 7 are used to demonstrate the performance as the minimum entry length approaches the maximum entry length. These plots reflect performance when the average cumulative locality becomes 40 using both exponentially and uniformly distributed entry lengths. The fixed-length entry size is 20, thus a locality of 40 is equivalent to two collisions. The model uses a hash area of 50,000 blocks, thus 2500 fixed-length entries are maximally possible. Theoretically[1] at 80 percent loading using a linear rehash, 2000 fixed-length entries are possible with an average of two collisions. As can be seen in Figure 2, empirical results bear this out. The abscissae of Figures 6 and 7 reflect the minimum entry length. A maximum length of 20 blocks is used for all the results shown.

The plot for the exponentially distributed entry lengths in Figure 6 shows that with a minimum entry length of 2 and an average entry length of 3, approximately 9,500 entries may be stored while keeping an average locality of 40

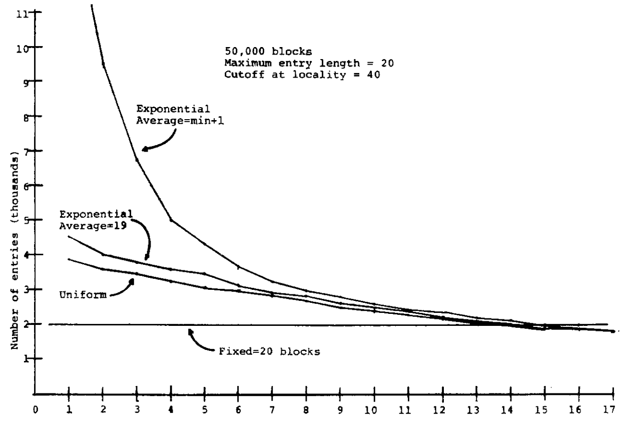


Figure 6—Number entries vs. min: max ratio

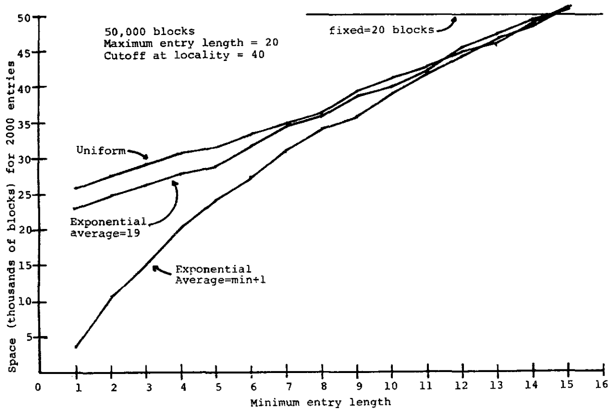


Figure 7—Space required vs. min:max ratio

blocks. Similarly, approximately 5000 entries may be maintained when the minimum entry length is 4 and the average entry length is 5. Furthermore, the performance using variable-length entries is superior to fixed-length entries until the minimum/maximum ratio is approximately 70 percent regardless of the average entry length or the distribution used!

Figure 7 is a plot of an extrapolation reflecting the hash area size required to maintain 2000 entries within the locality metric of 40. It should be noted that with exponentially distributed entry lengths such that the minimum length is 2 and the average entry length is 3, approximately 11,000 blocks are required for the hashing area in order to achieve the same performance as 50,000 blocks using the fixed-length approach.

A slight modification to the algorithm as presented above effects a small improvement in performance at the expense of elegance in the insertion and location algorithms. The original algorithm is affected only when an initial probe is made to a free block, but adequate contiguous free space does not exist beyond the initial probe to allow insertion of an entry. The modification incorporates a leftward scan from the initial probe address to determine if adequate contiguous free space exists immediately surrounding the initial probe to allow an insertion. If so, the entry is placed such that the first block is as close to the initial probe address as possible. Simultations using this modified approach show that, on the average, 1.3 percent more entries may be stored with the same locality performance. This conclusion is based upon 188 runs similar to those used to develop Figure 6. In approximately 70 percent of the runs, this alternate approach yielded more total entries.

CONCLUSIONS

Simulation results comparing the variable-length entry approach to a fixed-length entry approach demonstrate the viability of the use of variable-length entries to a hash area.

Using a fixed-length entry approach such that the entry length is the same as the maximum size variable-length entry, the variable-length approach is superior until the minimum entry length is approximately 70 percent that of the maximum entry length. Exceptional performance is achieved when the maximum entry length is several times that of the minimum.

Unlike the fixed-length and the bucketing approaches, the use of variable-length entries does not require a predetermination of the maximum entry length.

The simulation models described in this paper can be used to estimate performance of particular situations by modifying the entry length distribution generation and other parameters to fit a particular situation. The algorithm itself is quite simple and is adaptable for use with variability by any desired block size.

The advantages of locality of reference are obvious when hashing to addresses on external storage devices, and in the context of virtual memory systems, page fault minimization is also advantageous. Thus, the use of variable-length entries should be at least considered whenever the logical entry length possesses any variability. Regardless of the desirability of locality of reference, significant space savings may be realized by use of variable-length entries.

In addition to the use of variable-length entries in such applications as symbol tables and conventional random files, some other interesting potential applications are indicated. For example, the sharing of address space by multiple files having different record lengths may be reasonable. In addition, the notion of block descriptors might be expanded so that a hardware/firmware implementation of variable-length data storage may be realized. Preliminary investigation of such an implementation is now under way.

ACKNOWLEDGMENTS

I would especially like to thank two of my good friends and colleagues, Dr. Wayne Dominick and Dr. Bruce Shriver, for their extremely critical readings and pertinent comments regarding this work.

REFERENCES

1. Knuth, D. E., *The Art of Computer Programming, Vol. III: Sorting and Searching*, Addison-Wesley, Reading, Mass., 1973.
2. Morris, R., "Scatter Storage Techniques," *Comm. ACM* Vol. 11, No. 1, January 1968, pp. 38-44.
3. Maurer, W. D. and T. G. Lewis, "Hash Table Methods," *Computing Surveys* Vol. 7, No. 1, March 1975, pp. 5-19.
4. Brent, R. P., "Reducing the Retrieval Time of Scatter Storage Techniques," *Comm. ACM* Vol. 16, No. 2, February 1973, pp. 105-109.
5. Coffman, E. G. and J. Eve, "File Structures Using Hashing Functions," *Comm. ACM* Vol. 13, No. 7, July 1970, pp. 427-432.
6. Severance, D. G., "Identifier Search Mechanisms: A Survey and Generalized Model," *Computing Surveys* Vol. 6, No. 3, September 1974, pp. 175-194.
7. Bays, C., "The Reallocation of Hash-coded Tables," *Comm. ACM* Vol. 16, No. 1, January 1973, pp. 11-14.
8. Knuth, D. E., *The Art of Computer Programming, Vol. I: Fundamental Algorithms*, Addison-Wesley, Reading, Mass., 1968.
9. Maisel, H. and G. Gnugnoli, *Simulation of Discrete Stochastic Systems*, SRA, Chicago, 1972.

APPENDIX—THE ALGORITHM

```
/* Enter here after obtaining hash_address from hashing algorithm /*
/* hash_address must point to a descriptor */
/* all lengths refer to the number of blocks */
/* area_size is the number of atoms in hash area which
  is equal to the total blocks times length_block */
insert_entry:proc;
  q=hash_address;
  wrap_switch,space_found_switch=false;
  do while (space_found_switch=false);
     if area(q)=free
       then do;
          contiguous_space=0;
          p=q;
          do  while (area(q)=free & contiguous_space<length_entry);
              contiguous_space=contiguous_space+1;
              q=q+length_block; end;
          if  q>area_size+1
              then if wrap_switch=false
                     then do;
                        wrap_switch=true;
                        q=1; end;
                     else call(insufficient_contiguous_space);
              else if contiguous_space=length_entry
                     then space_found_switch=true;
                     else q=q+area(q)*length_block; end;
       else q=q+area(q)*length_block; end;
  length_remaining=length_entry;
  do i=p to q-1 by length_block;
     area(i)=length_remaining;
     length_remaining=length_remaining-1; end;
  /*  here install the data portion of the new entry */
end insert_entry;
```

Decomposition of data flow graphs on multiprocessors*

by W. C. BRANTLEY, JR., G. W. LEIVE and D. P. SIEWIOREK
Carnegie-Mellon University
Pittsburgh, Pennsylvania

ABSTRACT

Methodologies are presented for decomposing algorithms on a classical multiprocessor. The class of algorithms considered are those that can be represented as data flow graphs without decision elements. Two passive sonar signal processing modes are used as detailed examples. Decompositions are performed by modeling data memory bandwidth and data memory interference as the primary constraints on execution speed. Algorithms with low interference between data flow graph nodes require only simple models of memory interference to be successfully decomposed. For high interference algorithms the memory interference can be (1) modeled using a linear model of interference, (2) modeled by a queuing network of exponential servers which is solved computationally, or (3) modeled exactly and then simulated. These three techniques give estimates of the aggregate effect of memory interference to be factors of 10.9, 2.11, and 1.82, respectively (the latter being the most accurate).

INTRODUCTION

In recent years interest has heightened in multiprocessor structures constructed from low cost mini and microprocessors.[1] While such multiprocessors offer a significant cost/performance improvement over traditional uniprocessors, that advantage cannot be obtained unless effective methodologies exist for decomposing problems to execute in parallel. Effective decompositions have been made for specific applications[2] and small grain parallelism;[3] however, no methodology has been developed for a general problem. This paper presents a methodology for decomposing problems represented as data flow graphs on a classical multiprocessor. The methodology requires only minimal restrictions on the data flow graph. These restrictions are met by at least one important class of applications, passive sonar processing, which will be used as illustration. The restrictions on the data flow graphs are:

- The system must operate within realtime on the multiprocessor. Realtime operation means that in a given repetitive cycle the system data processing rate exceeds the data input rate.
- The data flow graphs are compositions of macros between which data flows. In the case of signal processing these are convolutions, digital filters, FFTs, and transforms.
- Macros are considered to be indivisible and nonpreemptable; once started they run to completion.
- The processing times for macros are modeled as constants.
- Macros are initiated by the availability of data.
- There are no data dependencies in control flow; data dependent decisions must be made within the macros.

Constraints on the methodology and the example decompositions are:

- The decomposition techniques discussed are only valid for a predictable, repetitive data flow process operating in steady state: system start-up transients are ignored.
- The methodology only applies to multiprocessor architectures with identical processors.
- The decomposition methodology assumes static binding of macros to processors.

The second section of this paper depicts the architecture of the multiprocessor assumed in the example decompositions. General methodology and methodology for low interaction data flow graphs is given in the third section. The fourth section illustrates the methodology on a passive sonar algorithm: Constant Percentage Resolution LOFAR (**LO**w **F**requency **A**nalysis **R**ecording). Another class of data flow graphs, those with high interaction, is treated in the fifth section followed by an example decomposition of **C**onstant **R**esolution LOFAR **B**eam**f**orming (CRBF) in the sixth section. The last section summarizes the results and indicates some areas for future research.

* This investigation was partially supported by the Naval Research Laboratory, Washington, D.C. under Contract Number N00173-76-C-0048.

MULTIPROCESSOR ARCHITECTURE

The decompositions are performed for a classical multiprocessor typified by C.mmp.[1] C.mmp (Figure 1) consists of 16 Digital Equipment Corporation PDP-11 processors communicating through a central crosspoint switch (Smp) to 16 memory modules. The major features of C.mmp are:

Mp: The primary memory consists of 16 modules of 65k 16 bit words. The memory access or cycle time (including switch and bus delay) is approximated as 1.0 microseconds.

Smp: The memory/processor crossbar switch connects any of the 16 processors to any of the 16 memory modules with a maximum concurrency of 16.

Pc: The processing elements may be any member of the PDP-11 family. The current implementation actually has a mixture of PDP-11/20s and PDP-11/40s.

Dmap: The address mapping component is the Dmap which intercepts 18 bit addresses generated by the processor and converts them to 21 bit physical memory addresses.

Functionally specialized processor definition

With PDP-11/40 processors, C.mmp is unable to execute any of the passive sonar signal processing modes in realtime.[4] Since the primary goal was to evaluate the memory-switch architecture, each C.mmp processor must be replaced with a functionally specialized processor (P.fs). P.fs is defined such that memory bandwidth and memory interference are the architectural constraints on the decompositions. Memory access time will be the limiting constraint on execution speed if P.fs executes primitive data operations in one main memory cycle time. To satisfy this requirement, instructions for executing the signal processing functions would be microcoded and stored in a high speed ROM within each P.fs. All accesses to main storage are for data. A P.fs will directly replace each C.mmp processor for the example decompositions.

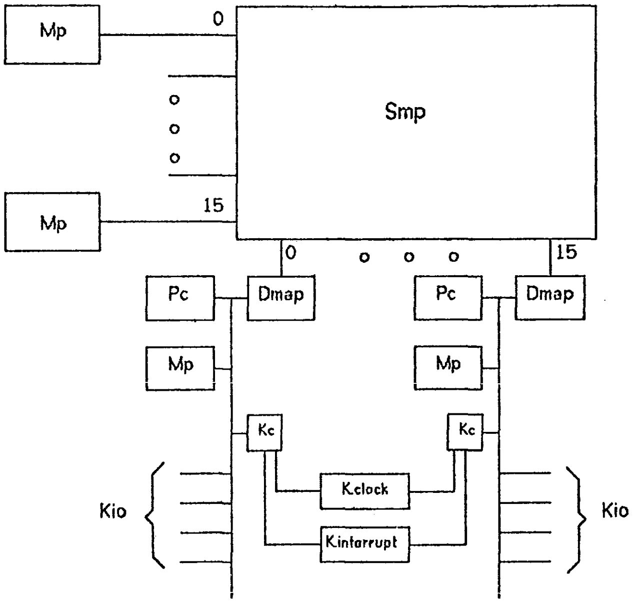


Figure 1—The architecture of C.mmp

The specification for P.fs may be put into perspective with other signal processors by considering the time to do an FFT. A butterfly of the FFT would take 10 microseconds or 10 memory cycles on the C.mmp architecture using P.fs processors. Therefore, a 1024 complex point FFT could be executed in 51.2 msec on the augmented C.mmp. The commercially available SPS-41[5] is a functionally specialized, high performance processor which takes 8.3 msec for a 1024 complex point FFT. In comparison with the SPS-41, the augmented C.mmp FFT with the P.fs specification is quite conservative.

GENERAL METHODOLOGY AND LOW INTERACTION DATA FLOW GRAPHS

In this section, general methodologies for decomposing algorithms are discussed. First, an outline is presented of a preliminary analysis that can be used for general decompositions. Subsequently, the methodologies for low memory interference and high memory interference data flow graphs are discussed. The goal of the methodology is to maximize the number of identical instantiations of a given data flow graph running on the multiprocessor concurrently and in realtime. Each instantiation of the data flow graph is called a channel.

Preliminary analysis

The following items are determined before trial decompositions are started:

- A data flow graph for a single channel must be available or must be developed.
- The processing time for each node of the data flow graph must be determined or estimated.
- The node processing times must be normalized to a convenient time interval (one second was used throughout the examples).
- The number of memory references (per unit of time) on each arc of the data flow graph must be determined (estimated) and then normalized to the same interval used for the processing times.
- The input data rate must be determined and normalized.

The following preliminary analyses are performed to detect degenerate cases:

- If any node requires more than one normalized unit of time to execute, there is no feasible decomposition (nodes are not decomposable).

- If the sum of all the node execution times for all the channels is less than one normalized unit of time, the processing requirements are met by a uniprocessor.
- Ignoring memory interference, determine the processing required for one channel.
- An upper bound on the number of channels that can be supported is determined by dividing the number of available processors by the total normalized processing time required for one channel.

Initial decompositions—Low interaction data flow graph

The following steps are preformed in a trial decomposition. The example is keyed to the methodology identifiers below.

ML-1 All special nodes which are not part of the cyclic processing defined by the data flow graph are bound to the minimum number of processors and memories that satisfy processing time requirements. This class of nodes would include input and output functions.

ML-2 Normalize node times. If the normalized execution time for any node exceeds one, no possible decomposition exists.

ML-3 Decompose one channel by assigning the largest number of adjacent nodes to processors such that the sum of all normalized node times assigned to any processor does not exceed one.

ML-4 Replicate the decomposed channel until available processors are exhausted.

ML-5 To assign overflow channels, determine if any processor has enough slack time to execute at least the largest node of a channel. If there is not enough time available, no overflow channels can be allocated. Determine if the sum of the available slack times is large enough to process at least one channel.

ML-6 Decompose overflow channels until slack is exhausted or until a complete channel will not fit in the remaining slack time.

ML-7 Determine the effect of memory interference on the trial decomposition by using a linear model of memory interference. If the normalized execution time requirement (including memory interference) for any processor exceeds one, remove nodes from that processor until the requirement drops below one. Remove any nodes that do not belong to integral channels.

Memory interference determination—Low interaction data flow graph

Memory interference encountered in tree structured data flow graphs is caused by several processors attempting simultaneous block transfers involving a single memory bank. Memory interference is seen as an apparent increase in memory access time. The apparent increase in memory access time is linear with the number of processors in conflict. To illustrate this concept, consider the following logical connection of processors and memories in Figure 2.

In Figure 2 memory M1 is accessed by processors P1, P2, and P3 for a total of b, c, and e accesses, respectively. Memory M2 is accessed by P1 and P3 for a total of a and d accesses, respectively. Interference results if the processors attempt to reference a memory simultaneously. For example, consider transfers b, c, and e to maximally interfere (i.e., they start simultaneously) and c<e<b. From the point of view of P1, b accesses now takes b+min(b,c)+min(b,e)=b+c+e time units. Similarly, from the point of view of P2, its c accesses to M1 will require c+min(c,b)+min(c,e) or 3c time units.

DECOMPOSITION EXAMPLE—LOW INTERACTION DATA FLOW GRAPH

Constant Percentage Resolution (CPR) LOFAR is an example of a passive sonar mode whose data flow graph has low interaction. CPR LOFAR computes the power spectral density by octaves for each channel. Figure 3 is the data flow graph for one channel of CPR LOFAR. Typically, as many as 16 channels of data are processed concurrently.

The CPR LOFAR modes are described below. The node execution times given were determined by scaling similar node times for a specific military signal processor.[4] The scale factor was determined by taking the ratio of the P.fs FFT butterfly time to the butterfly time for the military signal processor. Exact algorithms for the nodes are given in References 6-8.

INPUT: Receives the digitized transducer data and provides blocked data to DEMOD.

DEMOD: Divides the real input data into two parts: a low pass filtered part which is decimated by a factor of two and passed to the next DEMOD stage, and a high pass filtered part which is transformed

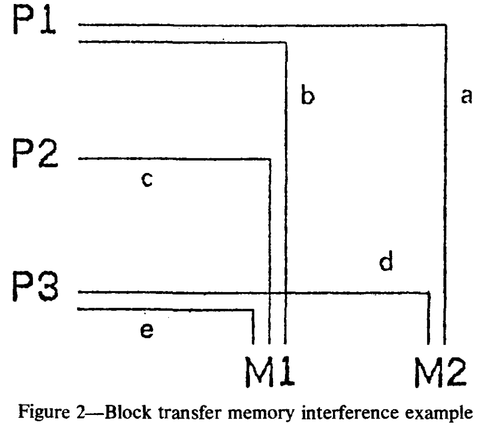


Figure 2—Block transfer memory interference example

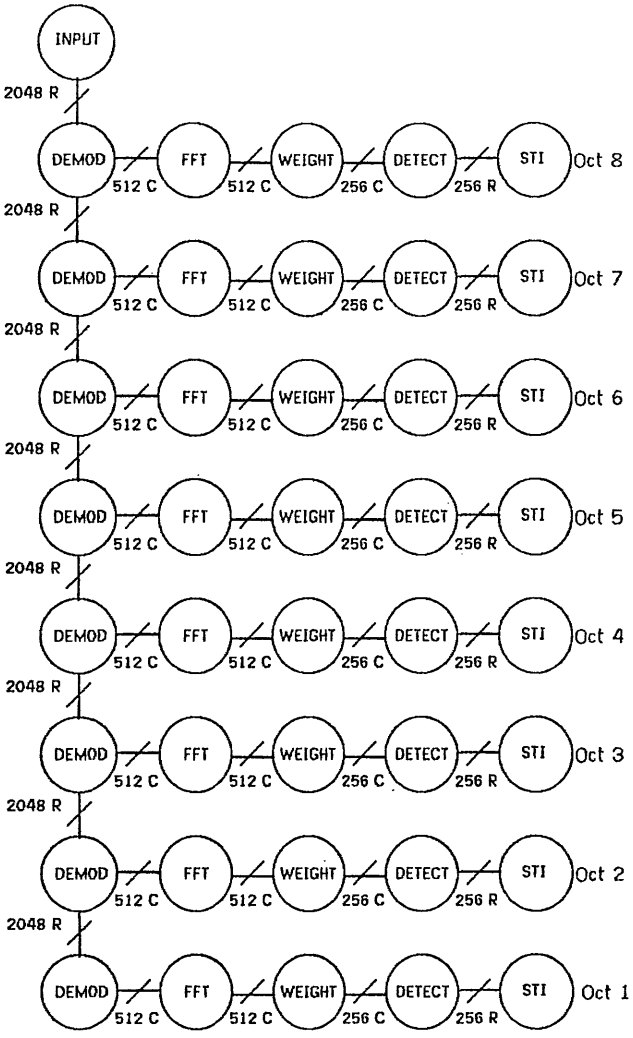


Figure 3—CPR LOFAR data flow graph

into the real and imaginary components of complex data points. The complex data points are bandshifted to baseband and the resulting complex array is passed to the FFT. From 2048 real points as input (2048 R in Figure 3), DEMOD produces 512 complex points for FFT (512 C) and 1024 real points for the next DEMOD. The execution time for DEMOD is 97.5 msec. on P.fs.

FFT: The Fast Fourier Transform (FFT) is a complex FFT that converts the time domain data to frequency domain. The execution time for the 512 complex point FFT is 23.2 msec.

WEIGHT: Enhances certain aspects of the data by computing weighted sums of four complex terms.[6] From each 512 complex point buffer from the FFT, WEIGHT produces 256 complex points. The execution time for WEIGHT is 9.1 msec.

DETECT: Calculates an approximation to the magnitude of each complex intensity. DETECT produces 256 real points from 256 complex points. The execution time for DETECT is 3.3 msec.

STI: Short Term Integration averages a number of spectra to smooth data and to eliminate false alarms due to non-recurring noise in a sample. STI takes the block of data from DETECT and accumulates to a real vector of size 256 real points. The execution time for STI is 3.3 msec.

POST: Is a summary term for various calculating, formatting, and display processing to interface the data to the display. POST is not shown in Figure 3.

The sample rate is 8192 Hz on each channel. Real data are represented with one 16 bit word and complex data with two such words. More details of the decomposition can be found in Reference 4.

Preliminary analysis

The initial step of the decomposition determines if it is possible to decompose CPR LOFAR on the modified C.mmp architecture. The execution times given show that the node requiring the most time is DEMOD. The ratio of required execution time to available time (97.5 ms/250 ms) is less than one, so a feasible decomposition is possible.

Decomposition

By method step ML-1, one processor/memory pair is dedicated to data acquisition and display storage. The remaining 15 processors are available for CPR LOFAR computations. The I/O processor receives data samples from all transducers and distributes the data samples to the appropriate channel processes. The I/O memory bank stores the processed data from all octaves of all channels. The I/O processor is available for any post processing of the data and for distribution of the processed data to the display.

The normalized execution time of node i in octave j for channel k is represented as $t_{i,j,k}$ for the remainder of the CPR LOFAR decomposition. The normalized execution time for octave j of channel k is $T_{j,k}=\Sigma_{i=1}^{m} t_{i,j,k}$, where the limit m is the maximum number of nodes in an octave (m=5 in this example).

CPR LOFAR (refer to Figure 3) is a tree data flow graph where octaves depend on preceding octaves for data input. The input data buffer to each octave is separated into two groups: half of the input is processed by the receiving octave and the other half of the data is filtered then passed to the next octave. An octave can initiate when its input buffer is full. Input buffers for all octaves are the same size.

An octave executes twice for each execution of its successor. Octave eight executes once for each data buffer supplied by INPUT. The normalized octave execution (method step ML-2) times are given in seconds per second:

$T_{8,k}=0.5456$	$T_{4,k}=0.0341$
$T_{7,k}=0.2728$	$T_{3,k}=0.0171$
$T_{6,k}=0.1364$	$T_{2,k}=0.0085$
$T_{5,k}=0.0682$	$T_{1,k}=0.0043$

Step ML-3 of the methodology is used next. Since the largest normalized octave execution time ($T_{8,k}$) is less than one, CPR LOFAR is decomposed by octaves rather than by nodes. To minimize memory interference, octave groups are bound to processors starting with octave eight. Successive octaves are bound to a processor until unity normalized time would be exceeded by adding another octave. In this example, octaves six, seven and eight fit on one processor ($T_{6,k}+T_{7,k}+T_{8,k}=0.9548<1$). The execution time for the other five octaves is $\Sigma_{j=1}^{5}T_{j,k}=0.1312$ seconds/second. Thus one processor can support the lower five octave processing requirements for $1/0.1312=7.56$ channels.

The decomposition on 16 processors for 12 channels of CPR LOFAR is as proceeds as follows using steps ML-4, ML-5, and ML-6. One processor is allocated to I/O processing. The remaining 15 processors are allocated in two groups of seven processor/memory pairs. Within a group, six processors execute octaves eight, seven, and six for each of six channels. The seventh processor within a group executes octaves five through one for the six channels. One processor on C.mmp is not needed. Figure 4 is a diagram of a portion of the decomposition for CPR LOFAR.

The single I/O processor (P.io) and I/O memory (M.io) are shown in the diagram. A processor and memory dedicated to the upper three octaves of one of the twelve channels (P.o86 and M.o86) is shown. A processor and a memory dedicated to the lower five octaves of six channels (P.o51 and M.o51) is shown. The data access variables (a through f) represent the accesses for only one channel. The variables are normalized to data accesses per second so that memory interference can be determined (method step ML-7). The values associated with the data access variables are:

a=248 words per second.
b=9984 words per second.
c=an unspecified (large) number of accesses available for display data processing.
d=9548ØØ words per second.
e=132138 words per second.
f=1Ø24 words per second.

The maximum number of apparent memory accesses/second for each processor (one channel) is:

P.io: a+min(a,e)+b+min(b,d)+min(b,f) +c=21448+c
P.o86: d+min(d,f)+min(d,b)=9658Ø8
P.o51: e+min(e,a)+f+min(f,d)+min(f,b)=135458

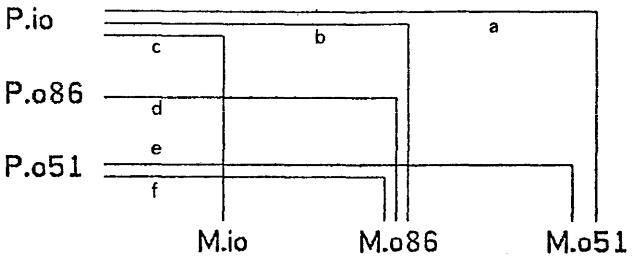


Figure 4—Part of CPR LOFAR decomposition

The maximum memory access rate for C.mmp is 10^6 words per second. The maximum normalized execution times are computed as:

(Number of channels) (apparent accesses)÷ (max accesses)

P.io: (12) (21448)/1Ø^6=Ø.257 (does not include "c") (utilization available to "c"=1.Ø−Ø.257)
P.o86: (1) (9658Ø8)/1Ø^6=Ø.966
p.o51: (6) (135458)/1Ø^6=Ø.813

The normalized execution times for all processors are less than one for the best and worst cases of memory interference. All processes are able to complete execution in sufficient time to maintain realtime operation of the system. When a processor is finished processing a block of data, it will execute a WAIT instruction to wait for the I/O processor to interrupt it with the next block of data.

METHODOLOGY FOR HIGH INTERACTION DATA FLOW GRAPHS

Data flow graphs having a high degree of interaction between some nodes cannot utilize the decomposition technique described for low interaction data flow graphs. The linear model of memory interference used for low interaction ~~data flow graphs assigns far more processors and~~ memories than are necessary when memory interference is intensive but complex enough to be modeled as a random process. The example is keyed to the methodology identifiers below. The decomposition technique with a more sophisticated model of memory interference is:

MH-1 Allocate the low interaction nodes to memory-processor pairs using the procedure described for low interaction data flow graphs.

MH-2 Assign the data shared by the highly interactive nodes to the remaining memories and assign the nodes to the remaining processors. Formulate a queuing model of the resulting network of processors and memories. The queuing model is a fully connected network of servers, each with a queue, where servers represent memory modules and customers represent processors. Associated with each edge of the network is a probability that the edge will be traversed when the source server finishes. These probabilities are determined by the assignment of nodes and their data.

MH-3 For each server in the queuing network determine the average time between the arrival time to the queue and the departure time of the customer (called the time in system). Time in system for a server corresponds to the effective memory cycle time of the memory as seen by a processor. Approximate the memory cycle to be exponentially distributed with mean equal to the real memory cycle time (a constant). To estimate the time in system, use Buzen's method.[9]

MH-4 Using the effective cycle time for each memory and the number of memory references made by processors memories, calculate normalized finish times. Increase normalized finish times to include interaction with processors in the low interaction group. From finish times calculate slack times (i.e., one minus the finish times). The slack times must be positive.

MH-5 If any slack times are negative, reassign nodes.

The methodology for high interaction data flow graphs is illustrated by CRBF in which memory interference is a dominant part of the processing load. Unlike CPR LOFAR, CRBF cannot be broken into memory-processor pairs that have little interaction. The processors performing the BF nodes uniformly access all of the memories containing FFT results and all of the memories for the resulting beams (Figure 5).

DECOMPOSITION EXAMPLE—HIGH INTERACTION DATA FLOW GRAPH

Constant resolution LOFAR beamforming (CRBF) calculates the power spectrum of the acoustical energy received along an azimuth.[7] Data for beamforming is acquired by sampling the output of an array of transducers in a known geometry. The beamforming algorithm is derived from the geometry of the transducers. Beamforming may be performed in the time domain or in the frequency domain; frequency domain beamforming is considered here. The equation for frequency domain beamforming is

$$R(\omega)= \sum_{n=0}^{2M-1} S(\omega,n) \exp((-2\pi \text{ i n d } \sin\theta)/c)$$

Where $S(\omega,n)$ is the Fourier transform of the signal from the nth channel, i is $(-1)^{1/2}$, and ω is $2\pi f$.

Figure 5 is the data flow graph of the CRBF mode which forms B beams from C transducers. The various nodes in the CRBF data flow graph are:

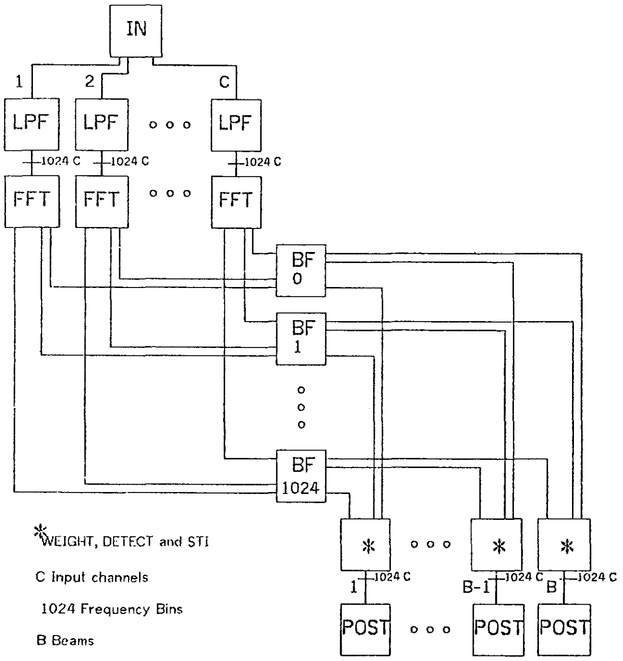


Figure 5—CRBF data flow graph

INPUT: Is identical to the input process of CPR LOFAR.

LPF: The Low Pass Filter limits the high frequency components of the sampled data and separates samples into inphase and quadrature components. From a block of 2048 real points, LPF produces a block of 1024 complex points. The execution time for LPF, T_{LPF}, is 61.7 msec.

FFT: The Fast Fourier Transform (FFT) is a complex FFT that is used to convert the complex time domain data to complex frequency domain data. The execution time for the 1024 complex point FFT, T_{FFT}, is 51.2 msec.

BF: Forms B beams for the jth frequency bin from FFTs of the C channels using Gortzels method.[7] The execution time for BF is 56.3BC msec.

WEIGHT: Is identical to the WEIGHT of CPR LOFAR. From each 1024 complex point buffer from the FFT, WEIGHT produces 1024 complex points. The execution time for WEIGHT, T_{WEIGHT}, is 25.7 msec.

DETECT: Calculates an approximation to the magnitude of each complex intensity (same as CPR LOFAR). From 1024 complex points, DETECT produces 1024 real points. The execution time for DETECT, T_{DETECT}, is 12.9 msec.

STI:	Is the same as STI of CPR LOFAR. The size of STI is 1024. The execution time for STI, T_{STI}, is 12.8 msec.
POST:	Is identical to POST of CPR LOFAR.

The first order analysis (method step MH-1) of the capacity of C.mmp for CRBF ignores memory interference within the pre-BF or post-BF nodes. The decomposition proceeds by first allocating $N_{PRE}=[C*(T_{LPF}+T_{FFT}+T_{IN})]^+$ processors and memories for pre-BF processing ($[x]^+$ symbolizes the least integer greater than or equal to x). Next, $N_{POST}=[B*(T_{WEIGHT}+T_{DETECT}+T_{STI}+T_{OUT})]^+$ processors and memories are assigned for the post-BF nodes. The remaining $N_{BF}=16-N_{POST}-N_{PRE}-1$ processors and B+C of the memories are allocated for the BF nodes. The maximum number of BF nodes any processor must compute is $[1024/N_{BF}]^+$. Slack time, $S_{BF}(C,B)$, is the minimum idle time remaining in each one second period for any of the BF processors. If S_{BF} is positive then, to first order, B beams can be formed from C channels of data on C.mmp using P.fs.

$$S_{BF}(C,B)=1000-4*(T_{I/O}+C*B*T_{BF}*$$
$$(1/(1024*Up(N_{BF}, B+C))*[1024/N_{BF}]^+) \text{ msec.}$$

Where $C*B*T_{BF}$ is the beam forming time given for one bin, Up is the processor utilization from Table 1, and $T_{I/O}$ is the time to move the data and results in the BF memories to other memories. To simplify computations, $T_{I/O}$ is calculated assuming no memory interference, but this assumption must be verified. Figure 6 is a graph of the capacity of C.mmp in beams (B) and channels (C).

BF with five channels and four beams will now be decomposed in detail. The computation time to reduce 1000 msec of data for the pre-BF and post-BF nodes are T_{PRE} and T_{POST}, respectively, giving:

$$T_{PRE}=4*(T_{LPF}+T_{FFT})$$
$$=4*(61.7+51.4)=452.4 \text{ msec.}$$
$$T_{POST}=4*(T_{WEIGHT}+T_{DETECT}+T_{STI})$$
$$=4*(25.7+12.9+12.8)=205.6 \text{ msec.}$$

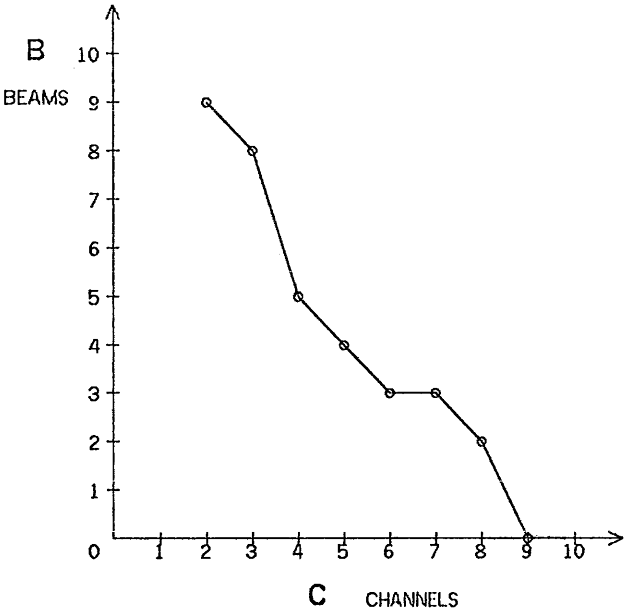


Figure 6—Capacity of C.mmp doing CRBF

Therefore, $N_{PRE}=3$ and $N_{POST}=1$. With one processor for I/O, eleven processors are left for BF nodes so that one BF processor will compute 94 BF nodes and the other ten will compute 93. Memory is allocated as follows: one bank for I/O, one bank for the post-BF processor, three banks for the three pre-BF processors, and B+C=nine banks for the eleven BF processors. The allocation of processors and memories is shown in Figure 7. The pre-BF nodes are divided among the three processor-memory pairs (PRE_1, PRE_2, PRE_3): PRE_1 has LPF_1, LPF_2, and FFT_1; PRE_2 has LPF_3, LPF_4, and FFT_3; and PRE_3 has LPF_5, FFT_5, FFT_2, and FFT_4 (subscript on a node indicates channel number). Both channel two and four have their LPFs in one processor and their FFTs in another processor, causing 8192 words to be written to memory PRE_3 from both processors

TABLE I—Up(N,M) in percent

Number of Memories, M

		1	2	3	4	5	6	7	8	9	1Ø
N	1	1ØØ.Ø	1ØØ.Ø	1ØØ.Ø	1ØØ.Ø	1ØØ.Ø	1ØØ.Ø	1ØØ.Ø	1ØØ.Ø	1ØØ.Ø	1ØØ.Ø
o	2	5Ø.Ø	66.7	75.Ø	8Ø.Ø	83.3	85.7	87.5	88.9	9Ø.Ø	9Ø.9
	3	33.3	5Ø.Ø	6Ø.Ø	66.7	71.4	75.Ø	77.8	8Ø.Ø	81.8	83.3
P	4	25.Ø	4Ø.Ø	5Ø.Ø	57.1	62.5	66.7	7Ø.Ø	72.7	75.Ø	76.9
r	5	2Ø.Ø	33.3	42.9	5Ø.Ø	55.6	6Ø.Ø	63.6	66.7	69.2	71.4
o	6	16.7	28.6	37.5	44.4	5Ø.Ø	54.5	58.3	61.5	64.3	66.7
c	7	14.3	25.Ø	33.3	4Ø.Ø	45.5	5Ø.Ø	53.8	57.1	6Ø.Ø	62.5
	8	12.5	22.2	3Ø.Ø	36.4	41.7	46.2	5Ø.Ø	53.3	56.3	58.8
N	9	11.1	2Ø.Ø	27.3	33.3	38.5	42.9	46.7	5Ø.Ø	52.9	55.6
	10	1Ø.Ø	18.2	25.Ø	3Ø.8	35.7	4Ø.Ø	43.8	47.1	5Ø.Ø	52.6
	11	9.1	16.7	23.1	28.6	33.3	37.5	41.2	44.4	47.4	5Ø.Ø
	12	8.3	15.4	21.4	26.7	31.3	35.3	38.9	42.1	45.Ø	47.6
	13	7.7	14.3	2Ø.Ø	25.Ø	29.4	33.3	36.8	4Ø.Ø	42.9	45.5
	14	7.1	13.3	18.8	23.5	27.8	31.6	35.Ø	38.1	4Ø.9	43.5
	15	6.7	12.5	17.6	22.2	26.3	3Ø.Ø	33.3	36.4	39.1	41.7
	16	6.3	11.8	16.7	21.1	25.Ø	28.6	31.8	34.8	37.5	4Ø.Ø

Memories

Processors	I/O	PRE_1	PRE_2	PRE_3	POST	BF_{C1}	BF_{C2}	BF_{C3}	BF_{C4}	BF_{C5}	BF_{B1}	BF_{B2}	BF_{B3}	BF_{B4}	S1	S2
I/O	-	16.4	16.4	8.2	16.4	-	-	-	-	-	-	-	-	-	-	-
PRE_1	-	699.2	-	8.2	-	8.2	-	-	-	-	-	-	-	-	-	-
PRE_2	-	-	699.2	8.2	-	-	8.2	-	-	-	-	-	-	-	-	-
PRE_3	-	-	-	863.6	-	-	-	8.2	8.2	8.2	-	-	-	-	-	-
POST	-	-	-	-	822.4	-	-	-	-	-	4.1	4.1	4.1	4.1	-	-
BF_1	-	-	-	-	-	46.0	46.0	46.0	46.0	46.0	46.0	46.0	46.0	46.0		
BF_2	-	-	-	-	-	46.0	46.0	46.0	46.0	46.0	46.0	46.0	46.0	46.0	-	-
BF_3	-	-	-	-	-	46.0	46.0	46.0	46.0	46.0	46.0	46.0	46.0	46.0	-	-
BF_4	-	-	-	-	-	46.0	46.0	46.0	46.0	46.0	46.0	46.0	46.0	46.0	-	-
BF_5	-	-	-	-	-	46.0	46.0	46.0	46.0	46.0	46.0	46.0	46.0	46.0	-	-
BF_6	-	-	-	-	-	46.0	46.0	46.0	46.0	46.0	46.0	46.0	46.0	46.0	-	-
BF_7	-	-	-	-	-	46.0	46.0	46.0	46.0	46.0	46.0	46.0	46.0	46.0	-	-
BF_8	-	-	-	-	-	46.0	46.0	46.0	46.0	46.0	46.0	46.0	46.0	46.0	-	-
BF_9	-	-	-	-	-	46.0	46.0	46.0	46.0	46.0	46.0	46.0	46.0	46.0	-	-
BF_{10}	-	-	-	-	-	46.0	46.0	46.0	46.0	46.0	46.0	46.0	46.0	46.0	-	-
BF_{11}	-	-	-	-	-	46.5	46.5	46.5	46.5	46.5	46.5	46.5	46.5	46.5	-	-

Figure 7—Thousands of memory cycles required by processors per second

PRE_1 and PRE_2. P.io deposits 16384, 16384, and 8192 words into the first, second, and third pre-BF memories, respectively. P.io also reads 16384 words from the post-BF memory. Figure 7 shows all data references between each processor and memory. Data is written to the BF memories at the end of each of the FFTs in the pre-BF processors and data is read from the BF memories by the WEIGHTs in the post-BF processors. More details of the decomposition are given in Reference 4.

Following step MH-2, a queuing model of memory interference model is developed. The linear model of memory interference developed for low interaction data flow graphs is too pessimistic since it assumes references to a bank of memory occur in bursts. In the case of BF nodes, the accesses are distributed in time. The interference between the BF processors is modeled by the queuing network of Figure 8 where servers and customers represent memories and processors, respectively. The queuing model operates as follows: when a customer (processor) enters a queue (initiates a memory request), service (a memory cycle) is begun immediately if the customer is alone in the queue, otherwise he awaits his turn. After service, the customer enters, with equal probability, one of the queues for his next memory cycle. The ratio of average service time to average time in system is the utilization of a processor, Up. Up depends on the number of memories, M, and the number of processors, N. For C.mmp neither N nor M can exceed 16.

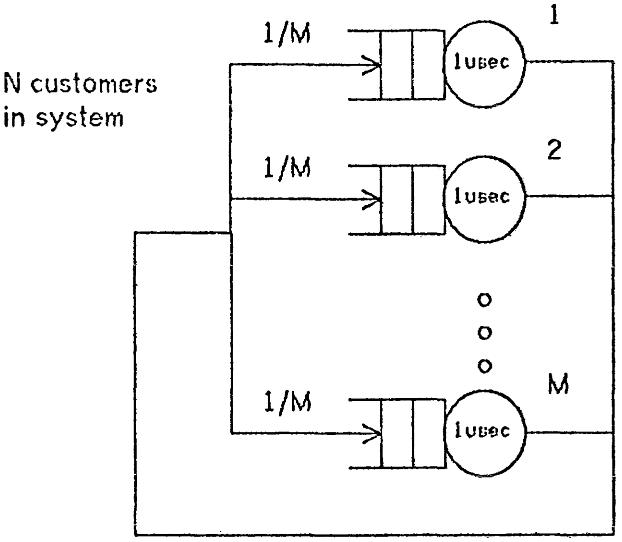


Figure 8—Queuing model of C.mmp

Memory cycle time is a constant one microsecond and corresponds to service time in the model. Since the service times are constant, the queuing network must be simulated to get Up. If the service times, however, are modeled as an exponential distribution then Up(N,M) can be determined analytically using the method of Buzen.[9] From the table of Up(N,M), for eleven processors and nine memories, the processors are waiting 1−Up(11,9)=52.6 percent of the time, leading to an apparent processor cycle time of 1/Up(11,9)=2.11 microseconds instead of one microsecond. For comparison, a simulation of eleven processors and nine memories having constant service time gives 55.5 percent processor utilization or an effective cycle time of 1.82 microsecond. In this case, the exponential assumption is a more conservative estimate of the effective processor cycle time. With the Up data, CRBF can be decomposed for a classical multiprocessor as follows: Isolate the BF nodes to minimize their interference with other nodes. Group the remaining nodes into two classes: those preceding the BFs (LPF and FFT) and those following the BFs (WEIGHT, DETECT, and STI).

Now the effect of memory interference on the decomposition is determined (steps MH-3 and MH-4). To operate in realtime, each processor must finish processing before the next block of data arrives. Finish times of the PRE-BF nodes are:

$$4*T_{BF}*B*C**94/(1024*Up(N_{BF}, B+C))+T_{transfer}$$

where $T_{transfer}$ is the time for the longest transfer involving any of the BF memories. The longest transfer is from P.post and takes 24.432 msec. The maximum time for any of the BF processors to finish is 896.8 msec, leaving 103.2 msec of slack from the 1000 msec interval. For post-BF processors, finish time is 822.4+16.4+4*4.1=855.2 msec. The finish times for the pre-BF processors are 756.6, 756.6, and 912.8 msec., respectively. These finish times leave a worst case slack time of 87.2 msec., thereby insuring realtime operation.

It is enlightening to compare the interference predicted by this method and the method used in the low interaction case. The finish time of BF_{11} predicted by the low interference model used by the decomposition is approximately (ignoring the interference from the PRE, POST, and I/O nodes): 9*(46.5+10*46.0)=4558.5 msec. This far exceeds the one second allowed for the BF mode to run in realtime. That is, the linear estimate is far too conservative when used on the high interaction case.

CONCLUSIONS

The methodology of decomposition varied with the type of data flow graph. CPR LOFAR has little interaction between nodes of the data flow graph. A simple memory interference model for CPR LOFAR estimates interference as a linear function of the number of processors referencing a memory bank. When concurrent block transfers interfere with one another, this linear estimate of interference is appropriate and accurate. CR Beamforming has a highly interactive data flow graph which requires a more sophisticated queuing model of memory interference. To calculate effective processor cycle time from the queuing model, the memory cycle time is assumed to be exponentially distributed. This queuing model is solved using (1) simulation (constant access per unit time), (2) Buzen's method (exponentially distributed memory cycle time), and (3) the linear interference model. These three techniques lead to effective processor cycle times of 1.82, 2.11, and 10.9 microseconds, respectively.

The methodology presented may not lead to optimal decompositions. However, in the examples presented, the processing time required for each channel was large enough to preclude more economical decompositions (i.e., $[N/T]^+$ was the number of concurrent channels). This situation may often be true in practice.

All these methods are useful design tools. The Buzen method is useful when memory references are random; the linear interference model is useful when memory references are mainly block transfers; simulation is useful as a final validation of a decomposition. It is anticipated that more complicated signal processing modes will require more detailed queuing models and simulation to verify decomposition.

REFERENCES

1. Wulf, W. A. and C. G. Bell, "C.MMP—A Multi-Mini-Processor," *AFIPS Conference Proceedings*, Vol. 41, 1972, pp. 765-777.
2. Heart, F. E., S. M. Ornstein, W. R. Crowther, and W. B. Barker, "A new Minicomputer/multiprocessor for the ARPA network," *AFIPS Conference Proceedings*, Vol. 42, NCC 1973, pp. 529-537.
3. Baer, J. L., "A Survey of Some Theoretical Aspects of Multiprocessing,"*Computing Surveys*, Vol. 5, No. 1, March 1973.
4. Siewiorek, D. P., W. C. Brantley, and G. W. Leive, "Modeling Multi-

processor Implementations of Passive Sonar Signal Processing," Final Report—NRL Contract Number N00173-76-C-0048, Carnegie-Mellon University, Pittsburgh, Pa., October 1975.

5. Allen, Jonathan, "Computer Architecture for Signal Processing," *Proc. of IEEE*, Vol. 55, No. 12, December 1967.
6. Smith, H. H. and L. E. Russo, "Microprogrammed Benchmarks for the Signal Processing Arithmetic Unit for the AN/UYK-17(XB-1)(V) Signal Processing Element," Naval Research Laboratory Report No. 7831, January 29, 1975.
7. Froscher, Judith N., "Evaluation of Algorithms for Linear Beamforming on the AN/UYK-17 (XB-1)(V) Signal Processing Element," Naval Research Laboratory, January 1975.
8. Shay, Barry P., "Design Considerations of a Programmable Prediction Digital Signal Processor for Radar Application," Naval Research Laboratory, NRL-7455, December 1972.
9. Buzen, Jeffrey P., "Computational Algorithms for Closed Queuing Networks with Exponential Servers," *CACM,* Vol. 16, No. 9, September 1973, pp. 527-531.

Implementation and application of a function data type*

by MARK B. WELLS

Los Alamos
Scientific Laboratory
of the University of California
Los Alamos, New Mexico

ABSTRACT

The modularization construct of subroutine, function, or procedure is well established in the scientific programming languages of today. In most cases, however, the construct is static in that once a function is defined and named it remains unchanged throughout the scope of its definition. We are concerned in this paper with the generalization to a function data type, that is, to the situation where one may have variables of type function which assume different specific procedures as their value within their scope. Notationally, implementation of the concept is made feasible by use of a juxtaposition operator. Computationally, the concept derives its usefulness from use of the contour model which allows activation record retention. Examples are given which show that the function data type concept subsumes less general techniques such as "coroutines" and "stream functions."

INTRODUCTION

Certain concepts of high-level, scientific programming languages such as recursive procedures, block structuring, and the general *while* and *if-then-else* control structures are well accepted due to their generality and usefulness in writing readable, well-structured programs. Other concepts such as hierarchical data structuring and abstract data types, while of more recent vintage, are fast proving their worth in the context of high-level languages. In this paper, we discuss experience with a function data type, a concept which we believe has promise of becoming as important as those mentioned above. This belief is based on the fact that function variables significantly assist with the high-level implementation of "generation procedures" and "backtracking"[1] and a number of other less general organizational approaches such as "coroutines."[2] Furthermore, our implementation of the function data type helps resolve the "harmful global variable" problem.[3]

The particular programming language involved in this work is Madcap, for which documentation is fragmented and incomplete.[4-6] However, this paper is self-contained since all relevant notation is either self-explanatory or explained herein.

NOTATION AND IMPLEMENTATION

Data of a Madcap program are categorized into objects called "spaces." A space includes a description of the operations that can be applied to the associated data items and, if the space is defined by the user, may include a description of the representation of a typical item.[6] There are certain spaces, called base language spaces, which have a syntactic construct for constants from that space. For example, REAL is a base language space with constants written 5.15, −623.9, 17, · · · . This space contains various operations—add, divide, square root, etc.—some being predefined by the particular computer system and others being defined at the language level. Terminologically, we say "a variable is of type real" or "has data type real" when the values it may assume and the primary operations that may be applied to it are from the space REAL. The base language spaces and examples of their constant form are given in Figure 1.

Note with regard to functions that the formal parameters are listed first using a colon notation and that the value of the last evaluated expression is returned as the result of a function's evaluation.

Madcap is a block structured language in which the text of a function definition, i.e., a constant of type function, is a block. Identifiers are local to a block if they are underlined at their point of definition within the block; they are global otherwise. Since identifiers are used solely as names of variables and there are no explicit declaration statements in the language, this point of definition will either be on the left of an assignment statement as in

$$\underline{v} \leftarrow 7.1$$

* This work was sponsored by the Division of Physical Research/Molecular Sciences of the United States Energy Research and Development Administration.

space	constant_form
REAL	1.264, 372, -0.019
BOOLEAN	true, false
STRING	"dog", "::/A="
FUNCTION	«x:real; 3x-2»
SEQUENCE	⟨2.65,-1.98⟩
SET	{0,4,8,27}
HIERARCHY	⟨↓x:true; ↓y:17.7⟩
SPACE	$⟨↓form:p; ↓pls:«···»⟩$

Figure 1—Base language spaces

or in the range of a loop statement (or loop expression) as in

$$\text{for } 0\leq\underline{i}<I\text{: } A_i\leftarrow 2i-1$$

The data type of an underlined variable is derived from context, in the case of an assignment from the type of the expression on the right of the arrow. For instance, the assignment statement above serves to declare v to be a local variable of type real as well as indicating that when executed the value 7.1 is to be assigned to the variable v.

There exist well-defined rules of type propagation through expressions; and since there are no jump statements in the language, it is possible to compute the types of all variables at compile-time by a direct flow analysis.[7] (Of course, the program must be complete, including all referenced spaces, and must be properly ordered, with all variables defined before they are used. This last requirement does *not* preclude the use of recursive functions and data which are an integral part of the Madcap language.) The precise operation that is being specified by an expression depends on the types of the operands as well as on the operator. For instance,

$$x+y$$

indicates a real addition if x and y are of type real while it indicates a matrix addition if the operands are of type matrix. In the latter case, a function for actually performing the matrix addition would exist in the (user-defined) space named MATRIX and would be called upon to perform the operation.

There are about ninety operators in the Madcap language for which operations can be defined in various spaces.[6] Some of these are predefined for base language spaces. For instance, $+$, $-$, $\times$, $/$, $\sqrt{\ }$, etc., imply established operations in the space REAL, while $\neg$, $\vee$, and $\wedge$ specify base language operations available on boolean variables. The only predefined operation of the FUNCTION space is evaluation. If the function value has no arguments then this operation is specified by the dagger (†) prefix operator, for example

† proc.

If the function has arguments, the evaluation is usually specified by the juxtaposition operator (one of the ninety), for example

$$f(x+1) \quad g(x, y) \quad \text{exists } x \quad f(x) \quad (f)x.$$

Since juxtaposition for identifier formation takes precedence over juxtaposition as an operator, it is not possible to have the immediate juxtaposition of two letters of the same case imply function evaluation (or any other operation for that matter). However, this is not serious since spaces separate tokens as in the third example above, and the dot (·) infix operator also specifies function evaluation as in

$$f{\cdot}x \quad g{\cdot}(x,y) \quad f(a){\cdot}b$$

for example. (The juxtaposition operator has precedence over the dot operation.)

The contour computational model[8] is used by Madcap. This means that the environment of a function value (block) is retained if needed for use by all activations of that value. Therefore, a function value must be represented by a pair of pointers, a pointer to the code associated with the value (code pointer) and a pointer to the activation record of the environment of the function definition (static link). A ⟨code pointer, static link⟩ pair can be passed around arbitrarily as the value of various function variables; it will be used to set up an activation record when evaluation of a function variable having it as value is requested. The static link of an activation record, which derives from the static link of a function value, is used to access global variables according to the block structure (i.e., function definition structure) of the program. (Incidentally, the static link of an activation record should not be confused with the "dynamic link" of an activation record. Dynamic links chain together the activation records according to function calls rather than function environments.)

Use of the contour model and its associated activation record retention allows functions values to be used freely as inputs and outputs of other functions and we truly have a function data type. Indeed one can specify additional operations besides evaluation, e.g., composition, within the FUNCTION space (see Reference 6 and Figures 9 and 10). (It should be emphasized that at any point in time only those activation records "accessible" (potentially still needed) are truly retained.)

APPLICATIONS

We present in this section several examples of application of the function data type. In most cases the program pieces have been abstracted from existing programs. They are categorized according to the characteristic application we wish to illustrate.

Isolation of Global Variables: The program piece given in Figure 2 illustrates the basic use of activation record retention. The variable *trace* is assigned to the output of the evaluation of a constant function. Here that output is a function. When *trace* is called, that function will be evaluated in an environment which includes the variable *n*. Thus *n* is global to the *trace* function but hidden from all parts of the program which call *trace*. (This application is reminiscent of the own-variable concept of Algol 60 in that only the *trace* routine itself and not its calls need be aware of the existence of *n*.)

```
trace ← †«
 n ← 0
 «
   s: string; g: general
   n ← n+1
   display ← "::/"; display ← s
   display ← " " ; display ← g
   if n ≥ 43:
      keyboard
      display ← "::r"
      n ← 0
 »
»
```

Figure 2—Simple retention

A similar application but with more levels appears in Figure 3. Here a function is assigned as value of SPACE.FORMER upon evaluation of a constant function. The subsequent evaluation of SPACE.FORMER then produces a space which has two hidden levels of environment to work under. Computation involved in forming the outer environment need only be repeated when the original constant function is reevaluated, while the inner environment will be recomputed, making use of the outer environment, at each call to SPACE.FORMER.

The essential characteristics of the function data type in these examples are retention and the fact that a function can change by virtue of a change in its environment. Following examples illustrate the effect of changing the value of a function variable to a new constant value.

Stream Functions—Generation of Sequences: Burge[9] discusses an abstract scheme, called "streaming," for generating successive elements of an arbitrary sequence. The idea is that a function call returns one element of the sequence *and* a new "stream function" for generation of the next element and the next stream function. Successive calls to the sequence of stream functions successively produces the desired sequence. (This desired generation effect was accomplished by special "generation procedures" in an earlier version of Madcap.[1]) The program of Figure 4 is a concrete realization of streaming using the function data type concept. The stream functions are successive values of the function varable *new.composition.* Each call of *new.composition* in the while statement produces the next composition (a composition of N into M parts is a vector $\langle c_1, c_2, \ldots c_m\rangle$ such that $\Sigma_{1\leq i\leq M}c_i=N$) as the value of the global variable c and changes the value of the variable *new.composition.* Each function evaluation also returns a boolean value to control the iteration. (Recall that the value of the last evaluated expression is returned from a function call.) The value *true* is returned while the generation is active and *false* is returned when the generation is complete.

```
SPACE.FORMER ← †«
 .
 . (outer environment)
 .
 †«
  p:parameter
  .
  . (inner environment)
  .
  $( ··· )$    ¢this space is returned¢
 »
»
 .
 .
 .
···SPACE.FORMER(x)···
```

Figure 3—Two level retention

```
.
.
.
c ← composition
new.composition ← «
 n: real
 d: real
 new.composition ← «
  n: real
  d: real
  i ← 0;  while c_i = 0: i ← i+1
  if i = #c-1: false
  else:
      c ← (c_i -1; 0:1≤k≤i; c_i+1 +1; c_k :i+2≤k<#c)
      true
 »
 c ← (d; 0: n items)
 true
»
.
.
.
while new.composition(N,M):
     .
     . (use composition c)
     .
```

Figure 4—Stream generation of compositions

A somewhat more involved example of streaming appears in Figure 6. This is a program piece, taken from the Madcap compiler itself, which forms the sequence of characters representing an identifier; the syntax diagram for *identifier* appears in Figure 5.

In Figure 6 the stream functions are successive values of the function variable *stream.* As before, these functions (1) return the sequence element as the value of a global variable, here of c (c is known to be either an upper or lower case letter upon entry to the initial value of *stream*), (2) assure that the value of the stream function is correct for the next generation, and (3) return *true* or *false* according as the generation remains active or is complete. In this example, the successive values of c are accumulated in a sequence using the structure former notation of Madcap.[5]

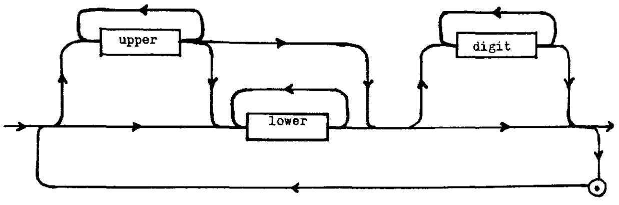


Figure 5—Syntax diagram for identifiers

```
α
c ← ↑next.character
if c is.upper ∨ c is.period:
    true
else:if c is.lower:
     stream ← lower
     true
else:if c is.digit:
     stream ← digit
     true
else: false
```

```
•
•
•
c ← undefined.character
stream ← «
 (upper,lower,digit) ← «boolean»
 upper ← «...α ↗ »
 lower ← «...β → »
 digit ← «...γ ↘ »
 c ← ↑character
 stream ←
      if s is.upper: upper
      else:          lower
 true
»
Identifier ← (c: while ↑stream)
•
•
•
```

```
β
c ← ↑next.character
if c is.lower:
    true
else:if c is.digit:
     stream ← digit
     true
else:if c is.period:
     stream ← upper
else: false
```

```
γ
c ← ↑next.character
if c is.digit:
    true
else:if c is.period:
     stream ← upper
     true
else: false
```

Figure 6—Identifier parsing using streaming

When complete this sequence is assigned as the value of *Identifier*.

Coroutines: The function data type allows a high level realization of the idea of coroutines[2,10] routines of equal stature that call each other and continue calculation at their previous point of departure when called themselves. A schematic illustration of this construction using Madcap notation appears in Figure 7.

In this scheme, two coroutines are implemented as the function variables *f* and *g*. A current evaluation of *f* changes *f* to a new value just prior to calling *g* and vice versa. As shown by the actual examples of this section, changing the value of a function may involve either changing the environment of a constant function value or assigning a new constant value to the function variable.

Backtrack Programming: The scanning of hierarchical structures, either actual or logical, is often called "backtrack programming." Backtrack programs usually take the form of a sequence of nested iterations, the nesting accom-

```
(f,g) ← («  ···  », «  ···  »)

•                                  •
•                                  •
•                                  •
f ← «                              g ← «
 •                                  •
 •                                  •
 •                                  •
 f ← «                              g ← «
  •                                  •
  •                                  •
  •                                  •
 »                                  »
  ↑g                                 ↑f
 »                                  »
↑g                                 ↑f
•                                  •
•                                  •
•                                  •
```

Figure 7—Coroutine outline

plished by recursive procedure calls. Using function variables, a different organization is possible that allows the handling of the terminal nodes to be extracted from the depths of the recursion. An example using this organization appears in Figure 8. (While perhaps not truly backtracking, the hierarchical search structure of this simple problem *is* characteristic of all backtrack programs.) This program collects the terminal nodes of a tree into a sequence called *A*. For instance, if the structure ⟨1, ⟨⟨2, 3, ⟨4⟩, 5⟩, 6, ⟨7, 8⟩, 9, 10⟩⟩ were used as input to *g*, then *A* would equal ⟨1, 2, 3, 4, 5, 6, 7, 8, 9, 10⟩ upon completion of the program. Each call to *g* produces a new environment for the successive activations of *f* at a deeper level of the tree, each activation being associated with one node of the tree. The parameters *h* save the function values to be used when backtracking. Note the function ≪false≫ being used in the initial call to *g*; this terminates the entire scan when called as the value of *f*.

```
f ← «true»
x ← terminal.type
g ← «
 s: (terminal.type: ? items)
 h: f
 i ← -1
 «
  i ← i+1
  if i < #s:
      a ← s
           i
      if terminal a:
          x ← a
          true
      else:
          f ← g(a,f)
          ↑f
  else:
      f ← h
      ↑f
 »
»
f ← g(S,«false»)
A ← (x: while ↑f)
```

Figure 8—Function variables in backtracking

The important feature of this example is that one terminal node (here, one value of *x*) is handled (or produced) by each call to the stream function (here, each evaluation of *f* during the formation of *A*).

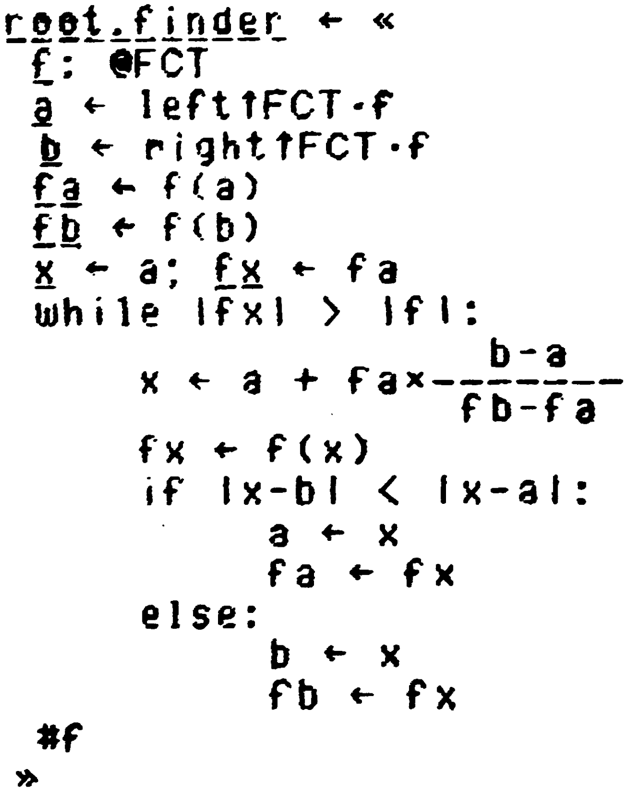
Function spaces: The function and abstract data type[6] concepts combine to permit the construction of spaces in which the elements are functions. A number of useful operations besides evaluation—composition, integration, differentiation—may be defined within such function spaces and referenced using natural notation—*f∘g*, *∫f*, *∂g*. In Figure 9 we illustrate partial function construction using the juxtaposition operator.

```
                  .
                  .
                  .
                  ↓jux: «
                   F: «(x,y):real; real»
                   a: real
                   b: real
                   if ¬(∃b):
                      «
                       y: real
                       F(a,y)
                      »
                   else:
                      «
                       x: real
                       F(x,b)
                      »
                  »
                  .
                  .
S ← $( ... )$     .

.
.
.
f ← «(x,y): real; ... »

.
.
.
h ← f(blank,a)
.
.
.
...h(3.2)...
```

Figure 9—Partially evaluated functions

In this example, *f* is an element of a space of real valued functions of two real variables. The juxtaposition operator is used to specify the partial function formation as illustrated in the assignment of *h*, a real valued function of one real variable. The formation operation itself (see the function labeled ↓jux) is a function of three formal parameters *F*, *a*, *b*; *F* has the same form as *f*, while *a* and *b* are real numbers. As shown, the value returned (last expression evaluated) is a function of one variable. The importance of retention is seen in this example, since when *h* itself is evaluated, it must do so within the environment of this formation in order to have access to the proper values of *a* or *b*.

Another potentially more important application of function spaces is suggested by the routine appearing in Figure 10. This is an abstract one-dimentional root finder routine. Its sole input is an arbitrary member, *f*, of a function space called FCT. All of the information needed to carry out the algorithm, left and right endpoints, error tolerance, etc., is extracted using operators (or function calls) of this space. The evaluation of *f* is specified using the natural juxtaposition operator, and the subtraction, multiplication, etc., operations of the argument space are also indicated with natural notation. Nevertheless, the precise form of the input function or type of its arguments are unspecified within this routine. That information is contained in the definition of the function space FCT and its environment. In a sense this algorithm is a higher level algorithm (a well-defined search) which is built upon lower level algorithms



```
root.finder ← «
 f: @FCT
 a ← left↑FCT·f
 b ← right↑FCT·f
 fa ← f(a)
 fb ← f(b)
 x ← a; fx ← fa
 while |fx| > |f|:
                         b-a
     x ← a + fax--------
                       fb-fa
     fx ← f(x)
     if |x-b| < |x-a|:
           a ← x
           fa ← fx
     else:
           b ← x
           fb ← fx
 #f
»
```

Figure 10—"Functional" algorithm

(e.g., evaluating particular functions over some domain) but which maintains its independence from those lower level algorithms. (In mathematical terminology such a routine might be called a "functional".) Indeed there are many other algorithms—Fibonacci search, adaptive quadrature, etc.—which benefit by having their input parameters be of a type defined by an external class of function spaces. Routines for these algorithms can be programmed once and for all efficiently, in a natural notation, and will remain unchanged over a wide variety of applications.

These examples are characteristic of the many already proven applications of the function data type. Investigations in its use are continuing. Rechard[11] has formulated a rather neat solution to the reader/writer problem of concurrent processes that uses function variables. This author has derived a reasonably natural decision table notation combining the function and set data types, an extension of the already useful Madcap sequence-of-functions notation used for "case" constructions.[5] Other applications are undoubtedly forthcoming as this simple yet powerful tool becomes more widely used.

CONCLUSION

The function data type concept appears in programming languages as early as Lisp[12] and Euler[13] and is one of the advanced characteristics of Algol 68.[14,15] It also appears unimplemented in the work of Burge[9] and, in a somewhat different form using sets, in SETL.[16] Nevertheless, partly because of notational questions, this concept has not received the attention befitting such an important unifying idea.

We believe that the use of activation record retention, a simple bracketed form for function constants, and the juxtaposition operator makes our scheme particularly natural and easy to use. Furthermore, our scheme benefits from and enhances other notational features of Madcap: (1) The type propagation scheme, which obviates explicit declarations, makes it more convenient to write algorithms that are completely independent of their data (Figure 10). (2) The structure former notation along with streaming provides a natural mechanism for constructing arbitrary sets, sequences and other structures (Figures 6, 8). (3) The value returned by a function being the last evaluated expression of the function body permits a concise yet readable form for deferred evaluation of expressions (Figure 8).

The existing Madcap compiler which implements the function data type described here does not possess a complete type-checking system. This deficiency is due both to expediency and to a lack of understanding with regard to type-checking for function values. However, we do now feel reasonably confident that the various values assigned to a function variable must at least have the same number of input parameters and output parameters. Also, corresponding parameters should, in some sense, have the same type. The question of global variables is less clear. We have determined that for certain optimizations to be applicable, the compiler must be able to determine all global variables referenced by any value of each function variable.

There is much work to be done with regard to compiler construction and optimization for languages with a function data type. Nevertheless, we consider the function data type to be a significant step in the quest for a very general, concise, and readable scientific programming language.

REFERENCES

1. Wells, M. B., *Elements of Combinatorial Computing*, Pergamon, Oxford, 1971.
2. Knuth, D. E., *The Art of Computer Programming, Volume 1: Fundamental Algorithms*, Addison-Wesley, Reading, MA, 1969.
3. Wulf, W. and M. Shaw, "Global Variable Considered Harmful," *SIGPLAN Notices*, 8:2, Feb. 1973, pp. 28-34.
4. Wells, M. B. and J. B. Morris, "The Unified Data Structure Capability in Madcap 6," *International Journal of Computer and Information Sciences*, 1:3, Sept. 1972, pp. 193-200.
5. Morris, J. B. and M. B. Wells, "The Specification of Program Flow in Madcap 6," *Proc. of 1972 ACM National Conf.* 2, 1972 Boston, MA, pp. 755-762.
6. Wells, M. B. and F. Cornwell, "A Data Type Encapsulation Scheme Utilizing Base Language Operators," *SIGPLAN Notices*, 11:1976 Special Issue, March 1976, pp. 170-178.
7. Wells, M. B., "The Madcap Programming Language," (in preparation).
8. Johnston, J. B., "The Contour Model of Block Structured Processes," *SIGPLAN Notices*, 6:2, Feb. 1971, pp. 55-82.
9. Burge, W. H., *Recursive Programming Techniques*, Addison-Wesley, Reading, MA, 1975.
10. Conway, M. E., "Design of a Separable Transition-Diagram Compiler," *CACM* 6, 1963, pp. 396-408.

11. Rechard, O., (private communication).
12. McCarthy, J., et al., *LISP 1 Programmer's Manual,* M.I.T. Computation Center and Research Lab. of Electronics, Cambridge, MA, 1960.
13. Wirth, N. and H. Weber, "Euler: A Generalization of ALGOL, and its Formal Definition: Part II," *CACM* 9:2, Feb. 1966, pp. 89-99.
14. van Wijngaarden, A., et al., "Report on the Algorithmic Language ALGOL 68," *Numerische Mathematik,* 14, 1969, pp. 79-218.
15. Tanenbaum, A. S., "A Tutorial on Algol 68," ACM Computing Surveys, 8:2, June 1976, pp. 155-190.
16. Mullish, H. and M. Goldstein, *A SETLB Primer,* Courant Institute of Mathematical Sciences, New York Univ. 1973.

A general purpose dialogue processor

by JAMES L. BLACK
*Science Applications, Inc.**
San Francisco, California

ABSTRACT

This paper describes work done on the Automatic Program Generator project at the Sperry Research Center. The overall aim of this research is to discover techniques that will make computers more directly accessible to non-technical users.

The approach taken is based on the use of interactive dialogue, using CRT terminals or work stations. A general purpose Dialogue Processor has been implemented whose function is to facilitate the creation and management of interactive dialogues. A high-level Dialogue Specification Language (DSL) is used, in which dialogues are represented as choice trees. Interactions with a user translate into selection or rejection of available paths plus any required data entry. Trees may be specified which present steps in the formulation of commands or transactions in the way that appears most natural to the user. User responses are then mapped into an appropriate format for output to the process that is to be dialogue driven. A Dialogue Editor enables the user to backtrack during a dialogue session, or at a later date to reformulate his responses.

The Dialogue Processor is a universal facility which has many potential uses; essentially, it can front-end any parameter or transaction driven system. It is currently being used as an interactive program specification technique in conjunction with an application customizer. Among other possible applications are its use as an aid to formulating complex command language statement (e.g., JCL) and as a query language interface for data bases.

We believe that the approach described in this paper, coupled with declining hardware costs and other technological advances, makes it possible to extract increasing benefits from interactive dialogue, while minimizing some of the traditional drawbacks.

IMPORTANCE OF EASE OF USE

It would be hard to overstate the importance that ease of use will assume in future computer products. This feature is given prominence in the list of design priorities for all new Sperry Univac products; and indeed, it is being stressed increasingly by the computer industry as a whole.

Ease of use becomes critical to the success of small business systems aimed at the first-time user market. For such customers, the cost of programming, if it were carried out in the traditional way by hiring application programmers, would easily overshadow the rental cost for the machine. At the lowest level, small business system customers have typically depended on the vendor, or a third party, to supply application programs. These generalized application packages have then to be tailored to the particular customer's business methods by the vendor's support personnel, either manually or in a more or less automated fashion through an application customizer. Such a situation arose when Sperry Univac OED decided to introduce the BC/7 small business computer. The Sperry Research Center participated in a joint effort with Sperry Univac to develop an interactive customizing facility for application packages, some details of which will be described later in this paper.

Ease of use considerations are not, of course, restricted to small machines; probably the most fertile area is to provide facilities for a whole range of users to gain access to distributed computer networks for a variety of purposes, with particular emphasis on shared data base applications.

STANDARD SOLUTIONS TO PROVIDING EASE OF USE

Easy to use languages

The traditional approach to providing solutions to the ease of use problem has been by designing languages. Because of the number of languages that have been created, and the lack of consensus on what are the criteria for deciding whether or not a language is easy to use, a discussion of computer languages will not be attempted here. While some general purpose languages make a claim to be easy to use, such as COBOL, BASIC or APL, another approach has been the design of special purpose languages. Very High Level Languages[1] could be characterized as languages in which the user specifies "what" he wants done rather than "how" to solve the problem; such

* Previously at Sperry Research Center, Sudbury, Massachusetts

an approach, almost by definition, limits the field of applicability of the language. Other examples of special purpose languages are data base "query" languages, such as Sperry Univac's UNIQUE and QLP languages which operate in IMS 90 and DMS 1100 environments respectively. These languages permit an end user to perform a restricted number of file operations, such as retrieval and/or update of data items which satisfy search criteria submitted by the user.

The two above mentioned, as well as many other languages, are designed to be used through terminals, in which environment the user both receives diagnostic and prompting messages at the time of submitting his program, and in many cases will receive output at the terminal on execution of the program. Interactive operation is one of the keys to ease of use.

Another approach to ease of use is to make the computer language approximate to natural language as closely as possible. Thus, there have been many efforts and proposals to use English as a programming language.[2] We are seeing much effort in the Artificial Intelligence community being directed towards English-understanding systems. However, even if it were conclusive that natural English was well adapted to expressing user needs (which is arguable), we are still a long way from being able to propose practical and economic systems using English as a programming language.

Other man-machine interfaces

Procedural languages are one form of man-machine interface. Another way we frequently communicate with the machine is through a command language (e.g., JCL). However, there are also many non-linguistic methods of getting the computer to execute the functions we desire. The most basic of these is typified by the computer operator's use of a console consisting of switches and lights. In some cases a standard terminal may be used in console-like fashion.

An off-line analogue to setting switches is the use of forms. The use of a check mark or one of a set of code symbols in a fixed position on a form is the equivalent to the setting of a switch, once the form is transcribed to a machine readable medium.

A third non-linguistic interface is interactive dialogue. In this case the operator responds to machine requests for decisions or values which will determine the succeeding course of events.

In general, non-linguistic interfaces are much less versatile, offering only a restricted range of possibilities, when compared to a general purpose programming language. However, generality is not always desirable or necessary when ease of use is the goal, and the amount of built-in prompting and control possible with non-linguistic systems makes them candidates for further development.

EASE OF USE THROUGH DIALOGUE

The use of computer dialogue is now, of course, commonplace; we see it used in a host of applications.[3] It is worthwhile, however, to reexamine some of the pros and cons of this technique.

Advantages

The advantages of dialogue driven systems may be summarized as follows.

- Learning is reduced or eliminated. The dialogue serves as a prompting mechanism or, depending on its verbosity, even as a tutorial, to guide the user's next response.
- User responses can be minimal. In a typical case an entire line of information might be displayed to the user, and his reply could consist of a single keystroke: Y for yes or N for no.
- User response can be validated immediately. Any unacceptable responses can be detected at the ideal time and the question either simply repeated or a diagnostic message displayed.
- The sequence of computer initiated questions is dynamically changeable, depending on the response to previous steps; thus, an appropriate subset of questions can be asked, depending on the problem, without the user's being exposed to irrelevant dialogue.
- As the dialogue proceeds, the user can save and review the previous interactions.

It is clear that dialogue is an invaluable tool in making computers easier to use. Well designed dialogue is "natural," if the computer's questions are phrased in terminology that is clear to the user. We can consider the computer as playing the role of an "automated consultant," asking all the necessary questions, and only the necessary questions, for some particular purpose. The function of the dialogue may be to control a process directly in real-time (such as in many time-sharing systems dialogues), or to extract information from the user. If information gathering is the purpose, the answers obtained from the user may represent the specification for some subsequent process; i.e., the responses could consist of language statements or a set of parameters.

Where a complex set of specifications is required, a form, or set of forms, is often employed. An example is the specification of a report using RPG, in which various forms describe the format of the output, the nature of the input and any calculations, summarizations or editing that is required. An alternative way of obtaining this information is through an interactive dialogue in which the user fills out the "form" step by step, by responding to questions about the problem he is specifying. One benefit of this approach over an actual form is apparent in situations where the consequence of indicating one choice implies that another section must be completed. No manual method can guarantee the user will complete the form correctly, but in an interactive environment, all relevant sections can automatically be presented as a dialogue sequence. Conversely, many columns, or even entire forms, may be unnecessary for specifying a simple problem; the equivalent questions would not be seen by the user responding to a dialogue; he is thus shielded from any more complexity than that needed to solve his problem.

Disadvantages

There have traditionally been several disadvantages to the dialogue approach to offset the advantages discussed above. Often, interactive dialogue mode has been considered simply as a crutch for beginners, to be discarded as soon as they have learned the series of steps through which the dialogue is taking them.

What are some of the problems that make people unwilling to tolerate dialogue once they have used the same sequence a few times? Some contributing factors are:

- Type of Device—Most dialogue up until now has been conducted on typewriter devices, in particular the Teletype Model 33. These devices are slow and noisy. In addition, they are linear devices and not well adapted to two-dimensional presentation.
- Speed of Display—This is related to the type of device; but even if a buffered CRT such as a Model 100 Uniscope ® terminal is used for dialogue, it will be limited by the throughput of the telecommunication line.
- Response Time—Most present-day dialogue driven programs reside on large, centralized systems which time-share the dialogue user with many other processes. Usually, each step in the dialogue incurs communications, processor and operating system overhead plus disc access time, all of which can amount to a noticeable delay between steps of the dialogue which may be irritating to an experienced user.
- Dialogue Design—Many dialogues appear to be poorly designed for their intended audience. It is common to encounter stylistic extremes ranging from verbose instructions intended to be helpful but ending up making the user feel the dialogue designer assumes he is an idiot, or alternately of cryptic messages and symbols that reinforce the user's feelings of confusion and mystery when using the computer. Other faults are a poor ordering of questions, designed more for the benefit of the machine than the user, and poor "sign posting," so that the user can easily lose his way in a series of questions, or suddenly find unexpected things happening.
- Cost—Typically, dialogue driven processes have required large systems, and have consumed large amounts of memory and other system resources. Telecommunication lines, particularly with high throughput, are another heavy expense. If one takes a narrow view and simply counts tangible costs, dialogue driven processes are not an "efficient" way of utilizing resources. Lastly, the development cost of software for on-line applications has been considerably greater than the cost of developing equivalent applications running in a non-interactive environment.

DIALOGUE APPROACH

Our approach to finding solutions to the ease of use requirement is essentially dialogue based. Many of the problems discussed above are tending to be diminished by current technological trends. For example, CRT terminals are no longer considered exotic, high cost items; similarly, communications technology is tending towards higher capacity at less cost. The possibilities offered by intelligent terminals suggest a new distribution of functions, outside of the host system. In such an environment it is clear that at least a part of the work associated with managing dialogue could be performed by the terminal itself.

While technological advances are tending to eliminate most of the disadvantages associated with the mechanics of computer dialogues cited above, we still see one serious problem remaining—that is the design and implementation of effective dialogues. Our approach has been to separate the dialogue activity from the system that is to be dialogue driven, by creating some specialized software: a Dialogue Specification Language, which encodes a dialogue from a high-level description, and the Dialogue Processor, which is a run-time interpreter, managing the interactive process of presenting the steps of the dialogue to the user and capturing his responses.

In a typical present-day situation, the dialogue is inextricably embedded in the application or system program. The form of the dialogue is entrusted to the software designer, who is frequently not an expert in human communications. Ideally, we believe, dialogues should be written by skilled communicators, such as technical writers, educators, industrial psychologists or specialists in the subject matter of the dialogue. Dialogue design should be made more responsive to human needs; the format, presentation and choice of words should reflect the user's familiarity with the interactive process. The terminology used should vary to suit different user communities, for example, people whose native tongue is not English.

The above requirements suggest that, for a given application, many dialogues could exist simultaneously, each turned to a different user need. The Dialogue Specification Language and Dialogue Processor were designed to meet these goals.

DIALOGUE PROCESSOR

The Dialogue Processor is a program that supervises an interactive process in which a succession of messages is displayed on a CRT screen. For each message the user's response is obtained, analyzed and used to control succeeding interactions. The Dialogue Processor is a general purpose program; it will operate with any dialogue that has been suitably encoded. In addition to the display output described above, the Dialogue Processor produces two other outputs: firstly an audit file from which a listing of the dialogue, expressed in natural language style, can be extracted for the user, and secondly a formatted output file that embodies the essential information of the user's responses.

The Dialogue Processor is essentially one component of a two-part system, the other component consisting of the process that is to be dialogue driven. Figure 1 is a block diagram of the Dialogue Processor; it depicts the output extracted from the user's responses as a string of parameters that might drive one of many different processes, such as an application customizer, operating system or file management system.

The questionnaire of Figure 1 is expressed in Dialogue Specification Language and encoded by the DSL compiler. It is worth noting that in general the writer of the dialogue will not be the same person as the interactive user, the former will need some specialized skills, while the latter is assumed to be non-technical.

Choice trees

Consider a dialogue as represented by a tree, with nodes corresponding to points at which the user has a choice and branches corresponding to messages and user input. The whole tree corresponds to the dialogue structure, and paths through the tree are permissible sequences of user responses. The dialogue can be conceptualized as having a syntax which can be represented with the following metalinguistic symbols:

{ } [ ] ‖ ‖ . . .

The following example illustrates the steps by which we can go from a syntactic structure to a dialogue. Figure 2 shows the syntax for the EXTENT statement in the JCL for Sperry Univac's OS/3 Operating System. Using the format and the accompanying rules as set out in the Reference Manual,[4] the syntax can be expressed rigorously in the format shown in Figure 3. Figure 4 shows the substitution of messages for mnemonics and codes. If we were to make a dialogue in which each group of parameters in turn were to be offered to the user, we could represent the syntax internally as the tree shown in Figure 5. Here the nodes exercise the following control over the branches which emanate from them:

K = Sequential—The Dialogue Processor follows the branches one by one in sequence (the user does not have control).

A = Mandatory Exclusive—The user must choose one and only one branch. The Dialogue Processor follows the path indicated.

B = Optional Exclusive—The user may choose one branch, which the dialogue Processor will follow; alternately, he is permitted to signify a null choice—"none of the above."

We emphasize it is the Dialogue Processor that enforces the choice rules at each node, making it impossible for the user to violate them. Thus, any path the user is able to trace through the tree results in a syntactically correct EXTENT statement.

As we will see later, the Dialogue Specification Language has features enabling us to write dialogues embodying the syntactic structures illustrated above, directly.

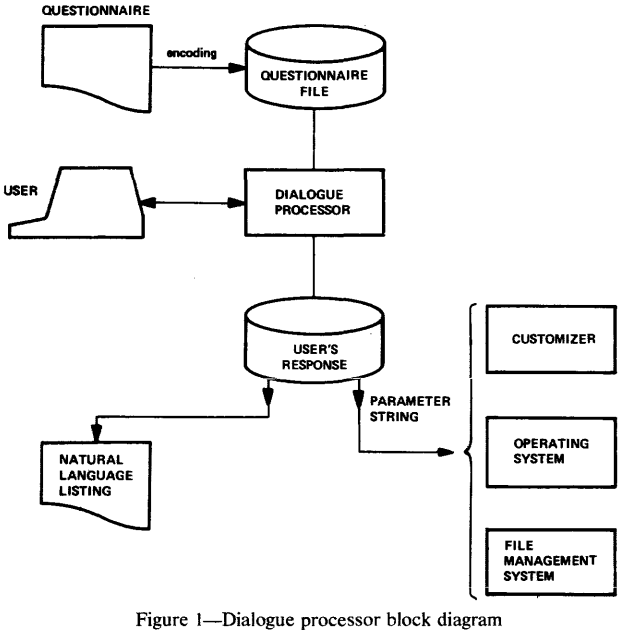


Figure 1—Dialogue processor block diagram

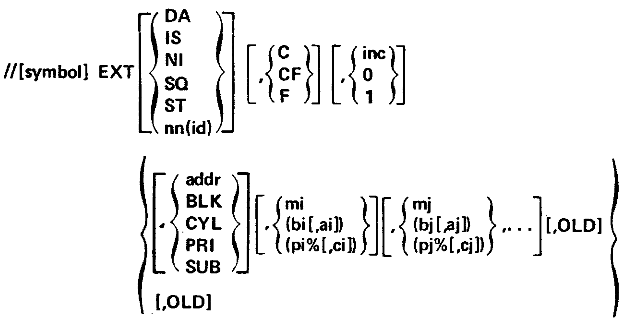


Figure 2—OS/3 job control EXTENT statement

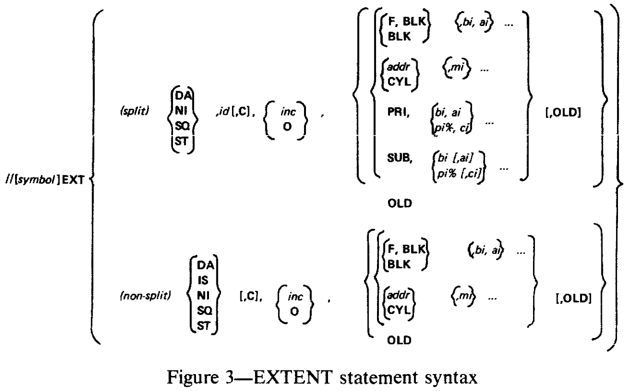


Figure 3—EXTENT statement syntax

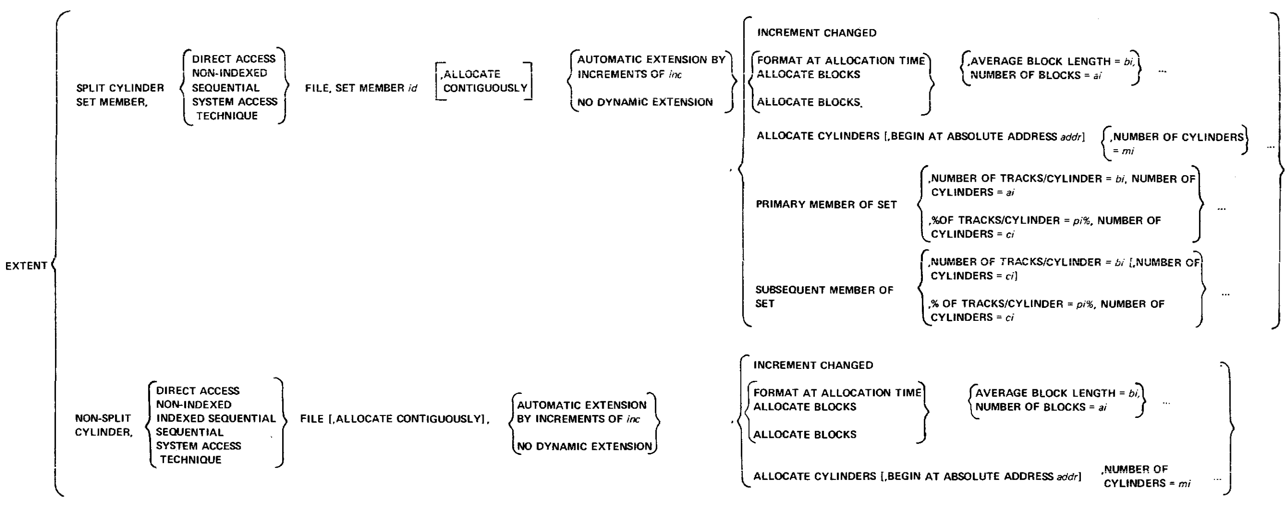

EXTENT
SPLIT CYLINDER SET MEMBER,
DIRECT ACCESS
NON-INDEXED
SEQUENTIAL
SYSTEM ACCESS TECHNIQUE
FILE, SET MEMBER id
,ALLOCATE CONTIGUOUSLY
AUTOMATIC EXTENSION BY INCREMENTS OF inc
NO DYNAMIC EXTENSION
INCREMENT CHANGED
FORMAT AT ALLOCATION TIME ALLOCATE BLOCKS
ALLOCATE BLOCKS,
,AVERAGE BLOCK LENGTH = bi,
NUMBER OF BLOCKS = ai
ALLOCATE CYLINDERS [,BEGIN AT ABSOLUTE ADDRESS addr]
,NUMBER OF CYLINDERS = mi
PRIMARY MEMBER OF SET
,NUMBER OF TRACKS/CYLINDER = bi, NUMBER OF CYLINDERS = ai
,%OF TRACKS/CYLINDER = pi%, NUMBER OF CYLINDERS = ci
SUBSEQUENT MEMBER OF SET
,NUMBER OF TRACKS/CYLINDER = bi [,NUMBER OF CYLINDERS = ci]
,% OF TRACKS/CYLINDER = pi%, NUMBER OF CYLINDERS = ci
NON-SPLIT CYLINDER,
DIRECT ACCESS
NON-INDEXED
INDEXED SEQUENTIAL
SEQUENTIAL
SYSTEM ACCESS TECHNIQUE
FILE [,ALLOCATE CONTIGUOUSLY],
AUTOMATIC EXTENSION BY INCREMENTS OF inc
NO DYNAMIC EXTENSION
INCREMENT CHANGED
FORMAT AT ALLOCATION TIME ALLOCATE BLOCKS
ALLOCATE BLOCKS
AVERAGE BLOCK LENGTH = bi,
NUMBER OF BLOCKS = ai
ALLOCATE CYLINDERS [,BEGIN AT ABSOLUTE ADDRESS addr]
,NUMBER OF CYLINDERS = mi
...

Figure 4—EXTENT statement re-expressed in English

Functions of the dialogue processor

The Dialogue Processor is responsible for the presentation and sequencing of dialogue elements. Dialogue elements consist of specific messages to the user (concerned with the subject of the dialogue); standard messages to the user (concerned with control, prompting, error notification); requests for the user to specify choice (menu selection) and request for the user to enter data. For each step in the dialogue, the Dialogue Processor provides the appropriate mechanism for the user's response. Thus, the user may be confronted with a list of elements from which to choose; accompanying the list will be instructions on how to signify his choice and what rules govern this particular choice. Choices may be mandatory or optional, mutually exclusive, or inclusive. In each case the Dialogue Processor automatically enforces the choice rule. Each interaction is terminated by the user signifying choice, acknowledgment, or by entering data. Following verification that the user's response was valid, the Dialogue Processor proceeds to the next interaction called for by the dialogue structure. Deciding where to go next is a Dialogue Processor function which will, in many cases, be dependent on the previous response. As the user follows a path through the dialogue, he may discover he has taken a "wrong turn" or may wish to review and change some earlier response; a feature of the Dialogue Processor is the ability to back up or assist the user to recover from an unintended response.

DIALOGUE SPECIFICATION LANGUAGE

DSL is a high level language especially designed for the creation of dialogues which will run under the control of the Dialogue Processor. It has facilities for specifying the dialogue structure, messages to be displayed, input to be entered by the user and the content, format, and mapping rules for both the formatted output file and the natural language report output. DSL source programs are compiled into a mixture of tables and interpretive code—the output of the DSL compiler is termed the encoded dialogue. The Dialogue Processor is the run-time interpreter for encoded dialogues.

A reference manual for DSL has been documented elsewhere.[5] In this paper we will only attempt to highlight some of the unusual features of the language.

Trees

DSL has the facility to describe trees which are used to represent dialogue structures. The language has a tree declaration statement, the syntax of which is shown in Figure 6. The DSL tree is essentially a specialized program structure consisting of an optional subroutine called the trunk, a control mechanism called the node type, and a collection of subroutines called the branches. DSL possesses a unique command

PRESENT *tree*

which, when executed, starts an interactive process in which the Dialogue Processor displays messages and elicits responses to enable it to traverse the tree under guidance from the user.

Associated with each branch are branch actions. These actions may display messages, request input or perform output and other processing functions. At the time the branch is initially displayed, all the branch actions are performed on a temporary basis; only after the branch has been selected are the actions made permanent. A branch may terminate with another tree, thereby providing a nesting structure.

Control of execution of the branch actions is governed by the node type of the tree. For an exclusive node, all the branches are executed temporarily, then the one selected by the user has its actions made permanent. An inclusive

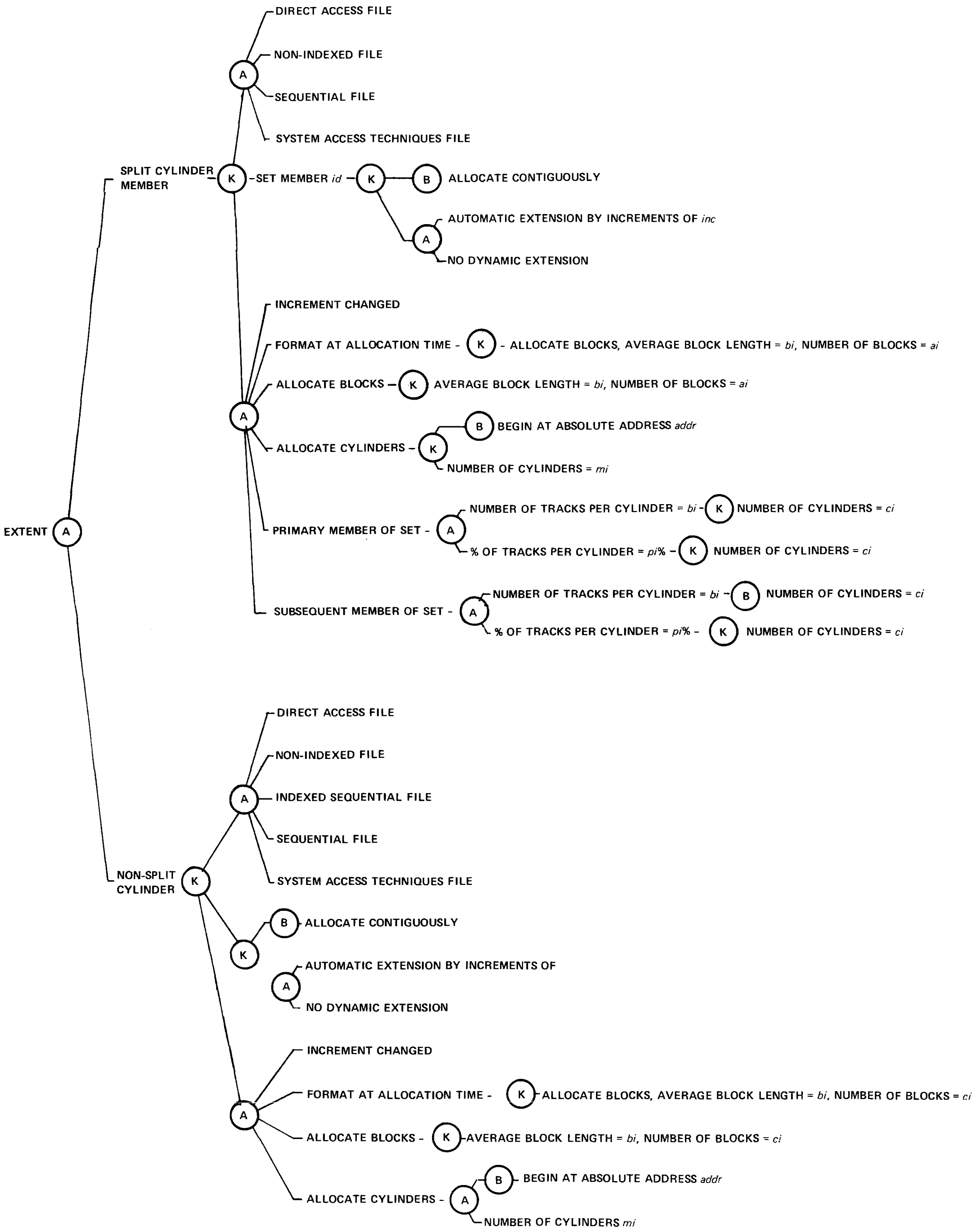

EXTENT
A
SPLIT CYLINDER MEMBER
K
SET MEMBER id
A
DIRECT ACCESS FILE
NON-INDEXED FILE
SEQUENTIAL FILE
SYSTEM ACCESS TECHNIQUES FILE
K
B
ALLOCATE CONTIGUOUSLY
A
AUTOMATIC EXTENSION BY INCREMENTS OF inc
NO DYNAMIC EXTENSION
A
INCREMENT CHANGED
FORMAT AT ALLOCATION TIME -
K
- ALLOCATE BLOCKS, AVERAGE BLOCK LENGTH = bi, NUMBER OF BLOCKS = ai
ALLOCATE BLOCKS
K
AVERAGE BLOCK LENGTH = bi, NUMBER OF BLOCKS = ai
ALLOCATE CYLINDERS -
K
B
BEGIN AT ABSOLUTE ADDRESS addr
NUMBER OF CYLINDERS = mi
PRIMARY MEMBER OF SET -
A
NUMBER OF TRACKS PER CYLINDER = bi -
K
NUMBER OF CYLINDERS = ci
% OF TRACKS PER CYLINDER = pi% -
K
NUMBER OF CYLINDERS = ci
SUBSEQUENT MEMBER OF SET -
A
NUMBER OF TRACKS PER CYLINDER = bi -
B
NUMBER OF CYLINDERS = ci
% OF TRACKS PER CYLINDER = pi% -
K
NUMBER OF CYLINDERS = ci
NON-SPLIT CYLINDER
K
A
DIRECT ACCESS FILE
NON-INDEXED FILE
INDEXED SEQUENTIAL FILE
SEQUENTIAL FILE
SYSTEM ACCESS TECHNIQUES FILE
K
B
ALLOCATE CONTIGUOUSLY
A
AUTOMATIC EXTENSION BY INCREMENTS OF
NO DYNAMIC EXTENSION
A
INCREMENT CHANGED
FORMAT AT ALLOCATION TIME -
K
ALLOCATE BLOCKS, AVERAGE BLOCK LENGTH = bi, NUMBER OF BLOCKS = ci
ALLOCATE BLOCKS -
K
AVERAGE BLOCK LENGTH = bi, NUMBER OF BLOCKS = ci
ALLOCATE CYLINDERS -
A
B
BEGIN AT ABSOLUTE ADDRESS addr
NUMBER OF CYLINDERS mi

Figure 5—Tree structure for EXTENT statement dialogue

$$\text{tree} ::= \textbf{TREE}\ [\text{trunk}] \left\{\begin{matrix} [\textbf{OPTIONAL}] \left\{\begin{matrix}\textbf{INCLUSIVE}\\ \textbf{EXCLUSIVE}\\ \textbf{SEQUENTIAL}\\ \textbf{PARALLEL}\end{matrix}\right] \\ \textbf{REJECT}\end{matrix}\right\} \left\{\begin{matrix}(\text{branch})\\ (\textit{branch name})\end{matrix}\right\}\ldots \left[\left\|\begin{matrix}\text{mask clause}\\ \textbf{SELECT}\ \{\textit{array name}\}\ \ldots\end{matrix}\right\|\right]$$

$$\text{mask clause} ::= \textbf{MASKED BY}\ \text{logical expression}$$

$$\text{branch} ::= [\text{branch element}]\ \ldots \left[\left\{\begin{matrix}\text{tree}\\ \textit{tree name}\end{matrix}\right\}\ [\text{mask clause}]\right]$$

$$\text{trunk} ::= \text{trunk element}\ \ldots$$

$$\text{branch element} ::= \left\{\begin{matrix}\text{assignment;}\\ \text{branch block}\\ \textbf{ENTER}\ \{\textit{array name}\}\ \ldots;\\ \left[\begin{matrix}\textbf{DISPLAY}\\ \textbf{OUTPUT}\\ \textbf{PRINT}\end{matrix}\right]\ \{\text{string expression}\}\ \ldots;\\ \text{conditional branch element}\end{matrix}\right\}$$

$$\text{trunk element} ::= \left\{\begin{matrix}\text{assignment;}\\ \text{trunk block}\\ \left[\begin{matrix}\textbf{DISPLAY}\\ \textbf{OUTPUT}\\ \textbf{PRINT}\end{matrix}\right]\ \{\text{string expression}\}\ \ldots;\\ \text{conditional trunk element}\end{matrix}\right\}$$

Figure 6—Syntax of DSL tree and branch declarations

node has a similar effect, except that the branches that remain unchosen after the first interaction are again presented to the user. In the case of a sequential node, only one branch at a time is executed. The purpose of a sequential node is to present branches in order. Parallel nodes are used for data entry sequences; neither parallel nor sequential nodes require a choice to be exercised by the user. The reject node type is the inverse of an inclusive choice—in this case it is those branches the user does not indicate that are executed. If a node type is qualified as optional, the user can indicate that he does not want to select any branch.

Branches and trees can be either written in line, or declared and named. If a tree is named, multiple references to it are possible, enabling a dialogue structure that is a network to be easily represented in DSL.

The trunk subroutine is executed once only, on presentation of the tree; its intended purpose is to display a heading for the user.

The optional MASKED BY and SELECT clauses will be explained following the discussion of DSL masks, below.

Data items, arrays and masks

DSL has facilities for declaring data items and arrays. Associated with every array is a mask with a bit corresponding to each element of the array. The effect of the mask is to only allow the DSL program to see those elements of the array for which the corresponding mask bit is set to 1.

If an array is referenced in a branch subroutine, it will have the effect of replicating the branch once for each element of the array; thus the array reference in the *n*th instance of a replicated branch is implicitly a reference to the *n*th element of the array. The above mechanism can be used to display all the elements of an array to the user as the branches of a tree, and, through the use of the select facility that exists in DSL, the array's mask bits can be set to mask off those elements that correspond to the branches not chosen.

Masks can also be used to condition the branches of a tree, this is the function of the MASKED BY clause. If this clause is used, the *n*th branch of a tree will only be presented to the user if the *n*th bit of the mask is set to 1.

Block structure

DSL uses a block structure to control looping, initialization and the scope of variables. The syntax for block declarations is shown in Figure 7. Looping may be controlled by a count, or continue while or until a certain condition is true.

Arrays can also be used to control looping. Using the DO FOR EACH *array-name* construction, a program block

```
                  block  :: =  DO [block control header] block body END

                             ⎧ ⎧ expression TIMES                                  ⎫                      ⎫
                             ⎪ ⎪       ⎧ ALL  ⎫                                    ⎪                      ⎪
                             ⎪ ⎪  FOR  ⎨      ⎬ {array name} . . . [mask clause]   ⎪                      ⎪
                             ⎪ ⎨       ⎩ EACH ⎭                                    ⎬  [INDEX index name]  ⎪
  block control header  :: = ⎨ ⎪ ⎧ UNTIL ⎫                                         ⎪                      ⎬
                             ⎪ ⎪ ⎨       ⎬ conditional expression                  ⎪                      ⎪
                             ⎪ ⎩ ⎩ WHILE ⎭                                         ⎭                      ⎪
                             ⎪ CASE expression                                                            ⎪
                             ⎩ CONTROLLED BY {array name} . . . [mask clause]                             ⎭

            block body  :: = {general statement} . . .
```

Figure 7—Syntax of DSL block declaration

controlled by an array will iterate through each element of the array for which the corresponding mask bit is set to 1. Any reference to the controlling array within the block is actually a reference to the current element of the array. This mechanism provides an implicit subscript feature in DSL.

The case statement has its conventional meaning, but the CONTROLLED BY mechanism provides a powerful extension of case—the effect of this construction is that for each 1 bit set in the mask of the controlling array, the corresponding general statement is executed.

The block mechanism is also used to control initialization. Variables are automatically initialized on entering, or re-entering the block in which they are declared. The scope of a variable is the block in which it is declared and all nested blocks.

Other features of DSL

There are powerful formatting facilities in DSL which can control the way data appear in the various outputs from the Dialogue Processor. Facilities include decimal placement and/or suppression, left or right justification, zero fill and centering.

DSL is also a general programming language with arithmetic, logical expression and string manipulation capabilities. The language has a simple macro facility.

The DSL programmer does not have to write explicit routines for handling user choices—the display of standard control and error messages, the positioning of headings and the tabulation of choices are all done automatically; similarly, the programmer does not have to concern himself with writing backtracking or error recovery procedures, as these are built in to the Dialogue Processor.

OUTPUTS OF THE DIALOGUE PROCESSOR

As the user progresses through a dialogue, he is building two distinct outputs. (We are not including, in this discussion, output messages to the CRT.)

Target file

As previously observed, the Dialogue Processor is one component of a two-part process; it is essentially the front-end to some dialogue driven process. The target file consists of exactly the input required by the complementary process. The Dialogue Processor is responsible for formatting the input correctly as well as ensuring a syntactically correct stream of data to the dialogue driven process.

The input requirements of many proposed target processes are machine-oriented rather than designed with good human factors (for instance, the cryptic codes and rigourous punctuation rules of JCL). One important function of the Dialogue Processor is to insulate the user from such problems, and to do so, it possesses a powerful mapping capability.

Figure 8 shows the syntax of a dialogue that is part of the information needed to specify a payroll. As the user responds to each element of this dialogue, he is specifying a rule that pertains to his payroll program; in this particular example, the rules for overtime pay. The use to which this information will be put is to furnish parameters to a customizer which will adapt a generalized payroll package to the user's particular methods. The input requirements of the customizer are, in fact, a collection of short procedural statements in a fixed assembly language-like format which are interpreted at run time. Figure 9 shows the paths selected during two passes through the tree, and beneath each statement is the code needed by the customizer to cause the processing to be carried out in the way described. The Dialogue Processor is able to map a human-oriented statement of requirements to the machine-oriented version required by the target process.

Report document

The other output created by the Dialogue Processor is an audit file from which a user oriented report document is extracted. Essentially the report summarizes, for the user's benefit, the dialogue that was transacted. Figure 10 is an

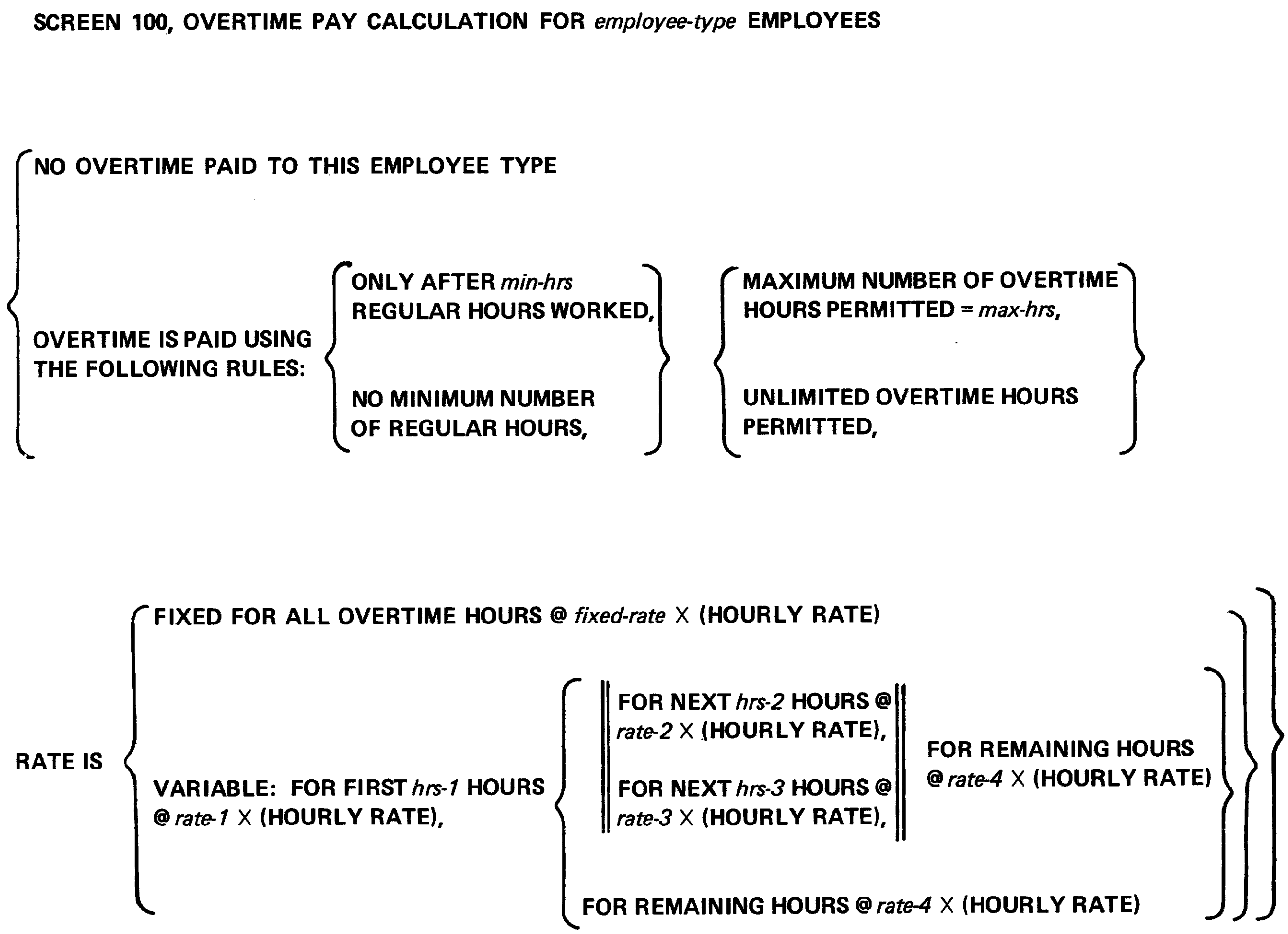

SCREEN 100, OVERTIME PAY CALCULATION FOR *employee-type* EMPLOYEES
NO OVERTIME PAID TO THIS EMPLOYEE TYPE
OVERTIME IS PAID USING THE FOLLOWING RULES:
ONLY AFTER *min-hrs* REGULAR HOURS WORKED,
NO MINIMUM NUMBER OF REGULAR HOURS,
MAXIMUM NUMBER OF OVERTIME HOURS PERMITTED = *max-hrs*,
UNLIMITED OVERTIME HOURS PERMITTED,
RATE IS
FIXED FOR ALL OVERTIME HOURS @ *fixed-rate* X (HOURLY RATE)
VARIABLE: FOR FIRST *hrs-1* HOURS @ *rate-1* X (HOURLY RATE),
FOR NEXT *hrs-2* HOURS @ *rate-2* X (HOURLY RATE),
FOR NEXT *hrs-3* HOURS @ *rate-3* X (HOURLY RATE),
FOR REMAINING HOURS @ *rate-4* X (HOURLY RATE)
FOR REMAINING HOURS @ *rate-4* X (HOURLY RATE)

Figure 8—Syntax for overtime rules in payroll dialogue

example of part of the report produced in specifying the payroll that was discussed in the previous example. The text of this report can be identical with the messages associated with each path selected during the interactive process (as is the case with Figure 10), or it can be separately specified to provide any suitable documentation. It should be noted that the style of the report can be close to a natural English description of the problem and thus be readily understandable to a non-technical person.

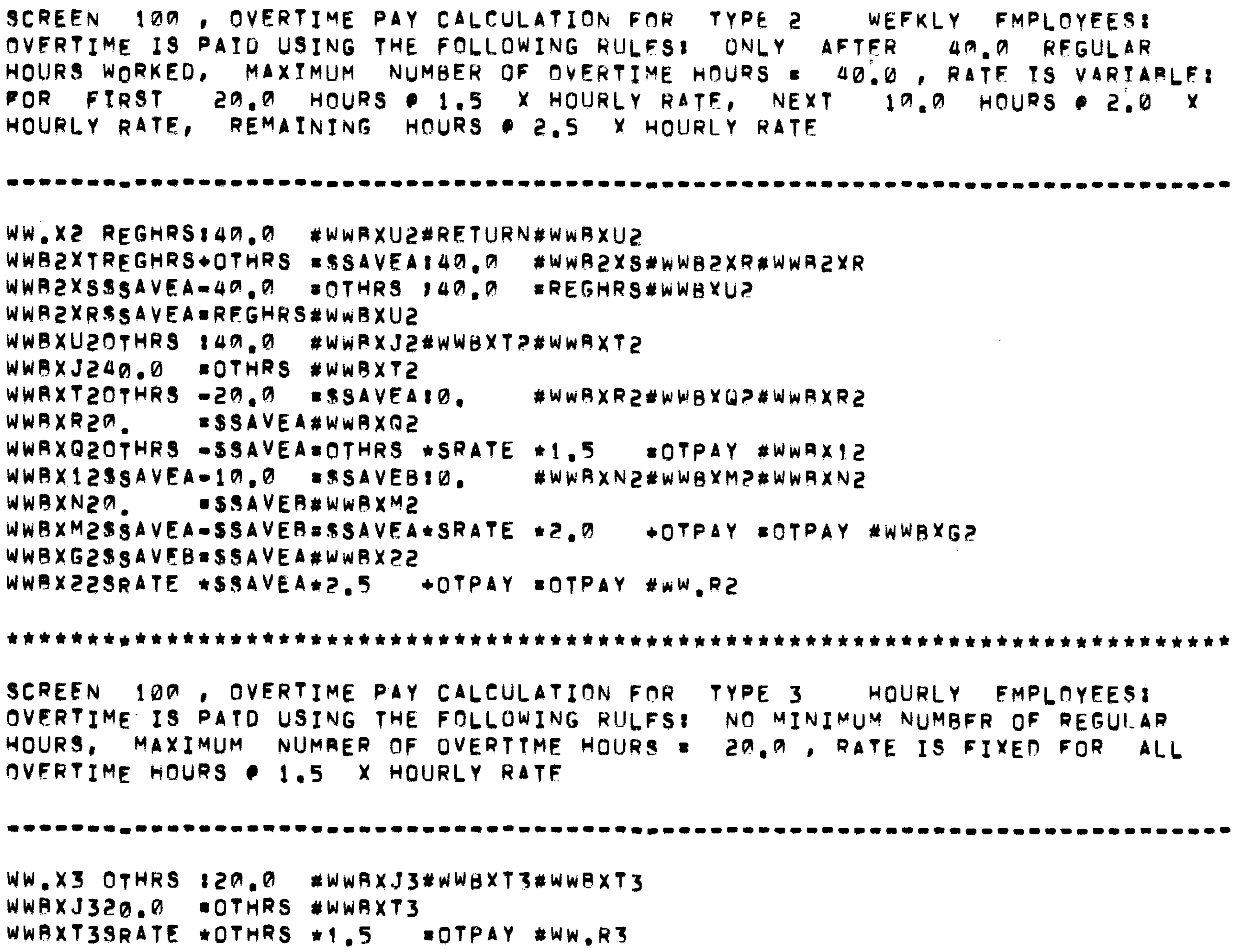
DIALOGUE EDITOR

The purpose of the Dialogue Editor is to enable the user to change his responses to a dialogue that has already taken place. The user may wish to change the data that he previously submitted in response to a data entry request, or he may wish to change a previously made choice selection.

The editor is intelligent in the following sense: Certain changes that the user can make will affect the subsequent dialogue, e.g., the deletion of a branch should result in all nested branches also being deleted. The editor will attempt to effect all consequent changes automatically, however, where the result of an earlier change lead to a situation where the user must provide more information, the editor solicits this via the corresponding dialogue.

Audit file

The editing mechanism is based on an audit file which is produced during a dialogue session and which contains essentially two kinds of information: firstly, the data that goes into the Report Document, which is user visible and is the means through which the user signifies any changes, and secondly, control information which is not user visible, but which is a trail of everything the user did during the session. The Dialogue Professor always operates with four files, Figure 11. The user indicates which paragraphs he wishes to change by writing the paragraph numbers to the change file. During an editing session, the Dialogue Processor reads unchanged paragraphs from the old audit file, produced at the previous session, interpreting the control information in the same way as if it had come from the keyboard. When it reaches a change paragraph, it displays the old version on the CRT screen, and waits for the user to



```
SCREEN  10Ø , OVERTIME PAY CALCULATION FOR  TYPE 2   WEFKLY  FMPLOYEES:
OVFRTIME IS PAID USING THE FOLLOWING RULFS:  ONLY  AFTER  4Ø.Ø  RFGULAR
HOURS WORKED,  MAXIMUM  NUMBER OF OVERTIME HOURS =  40.0 , RATE IS VARIABLE:
FOR  FIRST   2Ø.Ø  HOURS @ 1.5  X HOURLY RATE,  NEXT   1Ø.Ø  HOURS @ 2.0  X
HOURLY RATE,  REMAINING  HOURS @ 2.5  X HOURLY RATE

----------------------------------------------------------------------------

WW.X2 REGHRS:4Ø.Ø  #WWBXU2#RETURN#WWBXU2
WWB2XTREGHRS+OTHRS =$SAVEA:4Ø.Ø  #WWB2XS#WWB2XR#WWB2XR
WWB2XS$SAVEA-4Ø.Ø  =OTHRS :4Ø.Ø  =REGHRS#WWBXU2
WWB2XR$SAVEA=REGHRS#WWBXU2
WWBXU2OTHRS :4Ø.Ø  #WWBXJ2#WWBXT2#WWBXT2
WWBXJ24Ø.Ø  =OTHRS #WWBXT2
WWBXT2OTHRS -2Ø.Ø  =$SAVEA:Ø.    #WWBXR2#WWBXQ2#WWBXR2
WWBXR2Ø.    =$SAVEA#WWBXQ2
WWBXQ2OTHRS -$SAVEA=OTHRS *SRATE *1.5   =OTPAY #WWBX12
WWBX12$SAVEA-1Ø.Ø  =$SAVEB:Ø.    #WWBXN2#WWBXM2#WWBXN2
WWBXN2Ø.    =$SAVEB#WWBXM2
WWBXM2$SAVEA-$SAVEB=$SAVEA*SRATE *2.Ø   +OTPAY =OTPAY #WWBXG2
WWBXG2$SAVEB=$SAVEA#WWBX22
WWBX22SRATE *$SAVEA*2.5   +OTPAY =OTPAY #WW.R2

****************************************************************************

SCREEN  10Ø , OVERTIME PAY CALCULATION FOR  TYPE 3   HOURLY  EMPLOYEES:
OVERTIME IS PATD USING THE FOLLOWING RULFS:  NO MINIMUM NUMBFR OF REGULAR
HOURS,  MAXIMUM  NUMBER OF OVERTTME HOURS =  2Ø.Ø , RATE IS FIXED FOR  ALL
OVERTIME HOURS @ 1.5  X HOURLY RATE

----------------------------------------------------------------------------

WW.X3 OTHRS :2Ø.Ø  #WWBXJ3#WWBXT3#WWBXT3
WWBXJ32Ø.Ø  =OTHRS #WWBXT3
WWBXT3SRATE *OTHRS *1.5   =OTPAY #WW.R3
```

Figure 9—Customizer code for two sets of overtime rules

indicate the desired change. If a choice change is requested, the user is shown the original menu at that point, and the Dialogue Processor takes further input from the keyboard until the user terminates that particular request. Following a change, the information on the old audit file may or may not be valid. Whenever a conflict is detected, the Dialogue Processor returns to the user for responses that will resolve the inconsistency.

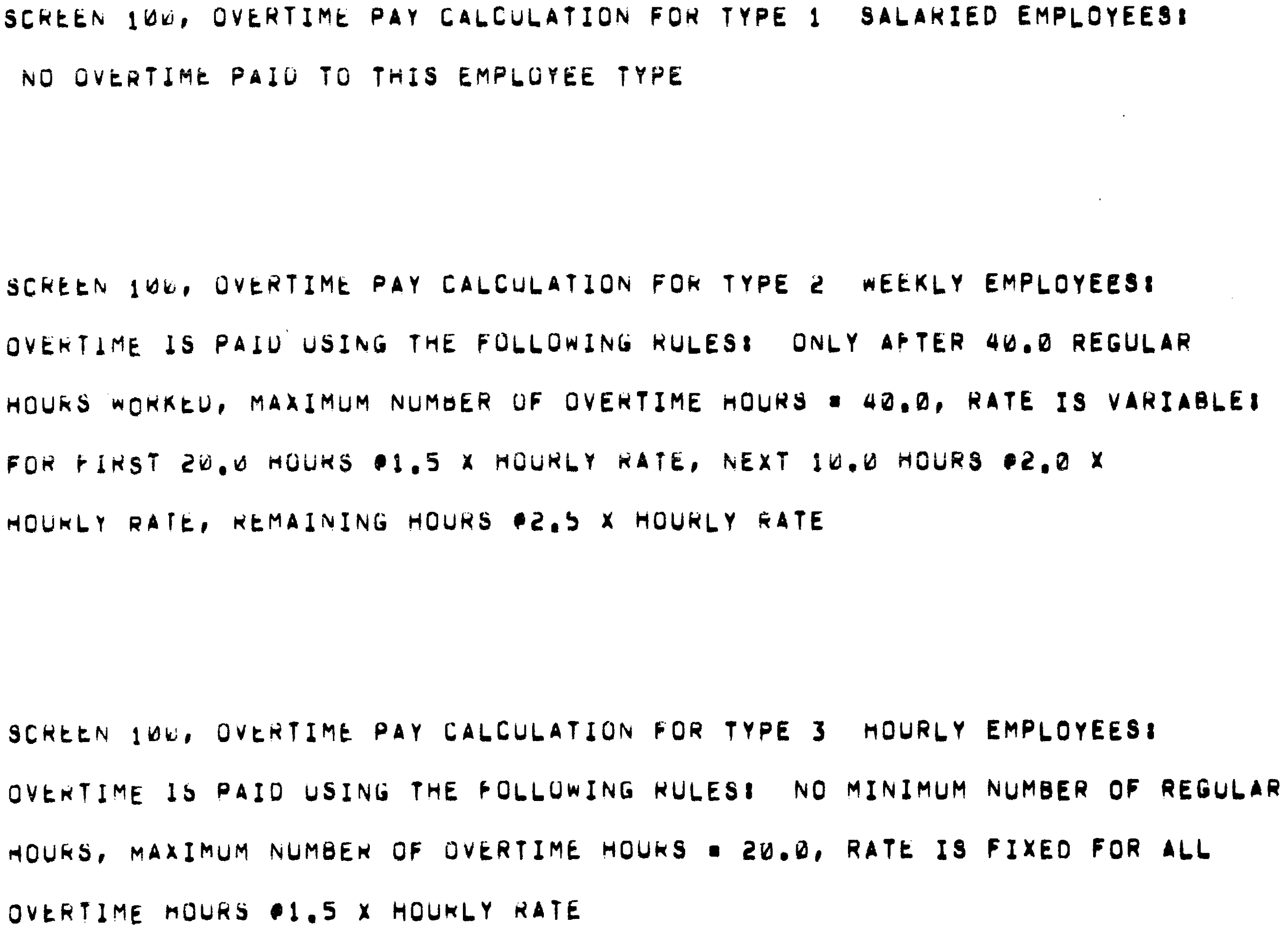
APPLICATIONS FOR THE DIALOGUE PROCESSOR

The initial application for the Dialogue Processor is in conjunction with a customizer that has been developed for use with packages for the small business machine market. The Dialogue Processor will replace the questionnaire forms that have been used with previous customizers.

Both the Dialogue Processor and customizer can run in the user's own machine; thus, it will be possible for the users to create their own applications without knowledge of a computer language. The automatic documentation feature, in conjunction with the Dialogue Editor, will also enable users to modify and maintain their own applications.

We are currently investigating other areas where the Dialogue Processor should prove effective. Some of these applications are:

- Formulation of commands by users in the traditionally difficult command language areas such as Job Control, System Generation, Network Specification and Control, and other instances where the user is required to submit complex specifications to a process.
- Creation of on-line applications (such as financial, medical, etc.) where the dialogue for receiving user requests and transactions has traditionally been an integral part of the application. We will attempt to demonstrate the advantages of flexibility and lower development costs the Dialogue Processor will provide.



```
SCREEN 1ØØ, OVERTIME PAY CALCULATION FOR TYPE 1  SALARIED EMPLOYEES:
 NO OVERTIME PAID TO THIS EMPLOYEE TYPE




SCREEN 1ØØ, OVERTIME PAY CALCULATION FOR TYPE 2  WEEKLY EMPLOYEES:
OVERTIME IS PAID USING THE FOLLOWING RULES:  ONLY AFTER 4Ø.Ø REGULAR
HOURS WORKED, MAXIMUM NUMBER OF OVERTIME HOURS = 4Ø.Ø, RATE IS VARIABLE:
FOR FIRST 2Ø.Ø HOURS @1.5 X HOURLY RATE, NEXT 1Ø.Ø HOURS @2.Ø X
HOURLY RATE, REMAINING HOURS @2.5 X HOURLY RATE




SCREEN 1ØØ, OVERTIME PAY CALCULATION FOR TYPE 3  HOURLY EMPLOYEES:
OVERTIME IS PAID USING THE FOLLOWING RULES:  NO MINIMUM NUMBER OF REGULAR
HOURS, MAXIMUM NUMBER OF OVERTIME HOURS = 2Ø.Ø, RATE IS FIXED FOR ALL
OVERTIME HOURS @1.5 X HOURLY RATE
```

Figure 10—Part of the payroll report output

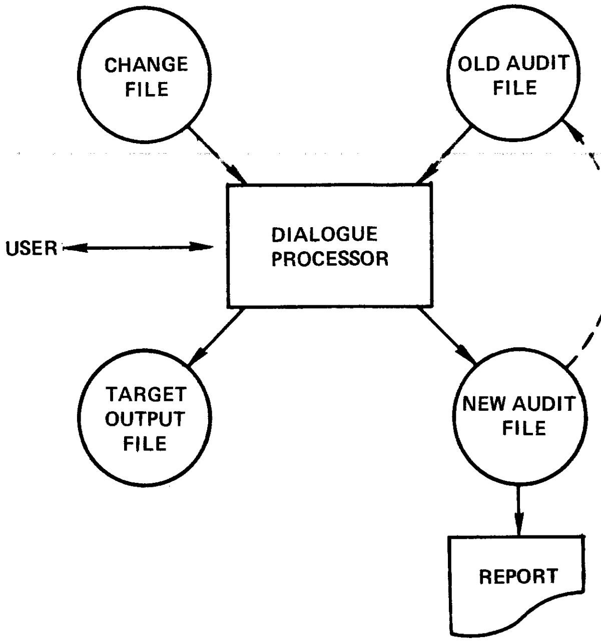


Figure 11—Dialogue editor

- Creation of end user facilities for interfacing with data base management systems; in particular, using dialogues to enable non-technical users to generate valid queries or transactions against data bases having complex structures and interrelationships. We plan to experiment with a variety of styles of interaction appropriate to different users, which the Dialogue Processor will map to a standard interface with the DBMS.
- Creation of dialogues for engineering diagnostics, which would enable the Dialogue Processor to be used by the customer to step through a series of tests and produce a detailed report of the fault, prior to calling the customer engineer.

We have already started to investigate some of the above areas. It is obvious there are many other applications for the Dialogue Processor.

CONCLUSION

In the first part of this paper we have identified some of the issues that affect ease of use. Particularly in the area of business applications, we can see the emergence of certain trends: Application generators, problem oriented languages, interactive systems and some serious effort to develop systems that employ natural language.

To aid in the development of easy-to-use systems we have created the Dialogue Processor and Dialogue Specification Language. With these tools we believe the utility of interactive dialogues can be greatly extended.

The Dialogue Processor is seen as one component of a two-part system. Since many different processes can be dialogue driven, the Dialogue Processor is truly a general purpose facility. Initially, the Dialogue Processor will interface with existing systems; however, once proven, we can expect the development of new systems especially designed to operate with a dialogue front-end.

With the Dialogue Processor, a user can express complex commands and specifications without being burdened with the effort of learning a programming language. All communications can be conducted in natural language; and furthermore, syntactic considerations are handled automatically.

DSL will encourage the development of effective dialogues tuned to users' needs. Dialogues can be developed as a separate activity from the processes they drive.

The Dialogue Processor has been designed with very modest processor and memory requirements; it is, therefore, feasible for it to reside in a programmable terminal. With this arrangement, the user will obtain very quick response between successive steps in the dialogue so it may be acceptable to use dialogue as the regular mode of expressing commands, and not simply to regard it as a training aid.

We expect the Dialogue Processor not only to make computers more accessible to non-programmers but also to help computer professionals become familiar with new facilities more quickly and use them more effectively.

ACKNOWLEDGMENTS

The author is greatly indebted to Mr. Jerome Holland for many discussions and insights, without which the work described in this paper would never have become a reality. Acknowledgment is also due for the efforts of Messrs. David Monk and William Overholt of Sperry Univac, who first saw the relevance of the Dialogue Processor to customizing small business applications, and who were responsible for supporting the research effort and its implementation in the Sperry Univac BC/7. Lastly, a large debt of thanks is due to Mr. Theodore Bonn for his encouragement and support and to numerous colleagues at the Sperry Research Center for their invaluable technical assistance.

REFERENCES

1. Leavenworth, B. M. and J. E. Sammett, "Overview of Nonprocedural Languages," *SIGPLAN Notices,* Vol. 9, No. 4, 1974.
2. Petrick, S. R., "On Natural Language Based Computer Systems," *I.B.M. Journal of Research and Development,* pp. 314-325, July 1976.
3. Martin, J., *Design of Man-Computer Dialogues,* Prentice-Hall, 1973, pp. 25-34.
4. Sperry-Univac, *Operating System/3 (OS/3) Job Control User Guide,* UP-8065 Rev. 1, 1975.
5. Black, J. L., "Dialogue Specification Language Reference Manual," Sperry Research Center Working Paper CM18-25, 1976.

A study in man-machine interaction*

by LAWRENCE H. MILLER

University of Southern California
Marina del Rey, California

ABSTRACT

The performance of users in man-machine interaction (MMI) is described in terms of a number of user- and machine-oriented parameters. The general linear model for experimental design is used as a model of the interaction. Performance measures are selected and a questionnaire developed to gauge user attitudes toward the man-machine system (MMS) and its environment. The interface parameters selected are hypothesized to have a significant effect on the performance and attitude measures.

The effects of varying CRT display rates and output delays upon user performance and attitudes in a series of message retrieval tasks were evaluated experimentally. The results support the somewhat surprising conclusion that doubling the display rate from 1200 to 2400 baud produces *no* significant performance or attitude changes; increasing the *variability* of the output display rate produces both significantly decreased user performance and a poorer attitude towards system and interactive environment. The generally held notion that increasing output display rates is associated with better user performance is not supported.

INTRODUCTION

Within computer science, it is now possible to divert attention away from fundamental theoretical issues toward refinements in systems and applications design for the greater satisfaction of end users. To this end, the previous ad hoc methods of refining systems to users' needs—based on the intuition of the designer or programmer—ought to give way to the more rigorous and reliable techniques of controlled observation, experimentation and development. The research reported here demonstrates that certain parameters of the man-machine interaction environment are manipulatable as a means of improving *user* performance.

An interactive message processing system now used at the University of Southern California's Information Sciences Institute (ISI) and other locations was used for this research. This program has been modified to provide a useful means of examining the relationship between the performance of the user and the variables influencing that performance.

Later sections discuss in greater detail the parameters of the interaction, which are shown to influence performance. Appropriate measurements are developed for evaluating performance in these kinds of interactive tasks. The rationale for performing research on the effects of changes in the system parameters upon user performance is to make possible the development of interactive computer systems that are more compatible with the needs, expectations, motivations, limitations and abilities of the potential users of that system.

A number of broad-ranging questions occur as one studies the ways in which people interact with computer systems: what are suitable input languages, keyboard and terminal designs, format and intensity of displays, amount of material, content of responses, speed and variability of display rates, etc. A complete theory of man-machine interaction (MMI) would take these factors as well as those relating to user differences into consideration in attempting to predict user performance for a given man-machine system (MMS). The theory would also include parameters relating directly to the individual system, and perhaps as well factors which relate to the supporting computer system—CPU speed, memory capacity, etc. A preliminary step in developing a model or theory of MMI includes the selection of useful performance measures.

The set of MMI parameters may be divided into those which represent the man (user) and those which represent the machine. The machine parameters, in turn, may be divided into those which represent the particular interactive program, those which represent the interactive environment (the terminal, display and input language form), and those which represent the background processor. By fixing all of the parameters except the display ones, the research reported here explores the effects of changes in the display upon user performance in a given (though not untypical) MMS. In fixing the user population, the MMS, the input language form and the background computer system, we still tacitly assume that the levels at which these have been fixed are representative of a broad class of interactive systems and users. It is this assumption of the generalizability of the research which makes the results of potential interest outside the immediate system and subject sample.

* This research was supported by the Advanced Research Projects Agency under contract No. DAHC 15 72 C 0308, ARPA order No. 2223. Views and conclusions contained in this study are the author's.

BACKGROUND

The number of controlled studies—either of specific systems and their user population, or broader theoretical studies of MMI—is extremely limited, although a number of authors express the opinion that these studies are needed. Willmorth[1] states:

> *Designing an information system for human use implies task analyses to determine the human actions to be performed, the decisions to be made, and the information required to be displayed to the human and expected from him, followed by the optimal design of the man/system interface. . . Time and effort must be devoted to designing a well-human-engineered system.*

Willmorth goes on to note that there is virtually no verified human engineering data for software and suggests an experimental methodology for examining the relationships between various versions of on-line planning systems and a set of (unnamed) performance measures or characteristics.

Bennett[2] concludes that "After a careful search of the major human factors and applied psychology journals . . . there is remarkably little evidence of research undertaken for the express purpose either of increasing our understanding of man-computer interaction or of providing information that will be useful in the development of systems that are optimally suited to user's needs." He identifies three areas that would benefit from human-engineering expertise: (1) conversational languages, (2) the effects of computer system characteristics on user behavior, and (3) the problem of describing, or modeling, man-computer interaction. He notes that early work with interactive facilities had computer efficiency as the paramount consideration, and that "the experience that makes optimum usage patterns obvious to the designer rests on a computer-oriented lore unknown to people who are not computer professionals." His final remark clearly expresses the need of a discipline of MMI design:

> *Because the theoretical basis for incorporating user problem-solving characteristics into analytical models is so rudimentary, the resulting user interface technology will take the form of procedural rules used by designers to guide their creative judgment. Indeed, the challenge for research is to transform the current art of design into an engineering discipline by developing an agreement on ways for characterizing user tasks, for allocating interface resources to meet task requirements, and for evaluating user effectiveness in task performance.*

Specific examples of research designs, methodology and results in the MMI literature include Walther and O'Neil,[3] who studied the effects of both user characteristics (for example, evaluative attitude [i.e., prior attitude toward computers], experience with on-line systems), and program and terminal characteristics (TTY vs. CRT, flexible vs. inflexible command recognizer). They found significant effects for terminal type and interface flexibility, as hypothesized, but there were often significant interactions with user experience or evaluative attitude. Since some of their results were counter-intuitive, they add evidence that there is a need for carefully designed, well controlled experiments on the relationship between user and system characteristics and *user* performance.

Hansen[4] examined differences in performance of groups of users in solving complex problems in on-line vs. batch environments. He concludes his work by noting "It is not necessary to predict accurately and in detail in order to be useful. Man-machine research may be effective if it serves only to help the designer to organize his thinking about how [users] perform, to enable him to distinguish those variables which are likely to be important, and to design ad hoc experiments to answer specific questions."

The few researchers who have performed experimental research on interactive systems feel that the user's view of the system is as important to the success of a man-machine system as more objective performance measures such as time to complete tasks, errors, cost, etc. Sterling[5] discusses the need for "humanizing" computerized information systems and the difficulty in deciding just what that term means. Martin, Carlisle and Treu,[6] in examining the man-machine interface in a number of interactive bibliographic systems, note that there is a lack of "knowledge about the blend of ingredients that produces a comfortable man-machine interface."

Thus it is reasonable to optimize systems in terms of user-oriented performance measures. A questionnaire (see Appendix 3) was administered to the subjects of this study as a means of eliciting their attitudes toward the system as they had just experienced it. The post-test questionnaire data was further analyzed to determine whether differences in the versions of the system of this study are associated with differences in user attitudes toward the system.

The study of time and delays in interactive systems is presented by Miller,[7] who lists a number of interaction modes (from first logon through requests for lengthy compilations) and discusses reasonable time delays for system response. The reasonableness of a delay is based upon user expectation and psychological closure. He does not discuss the possible effects of continuous excessive or unanticipated delays in response upon the user over a period of time. The effects of repeated delays upon the user's performance and attitudes form the foundation of the research reported here.

METHODOLOGY

The experimental design used to test the effects of the variables of this study on the performance measures was a 2×2×2 factorial design, with repeated measures on each subject across the two output volumes. A diagram of the factorial design is presented in Figure 1. The levels of the

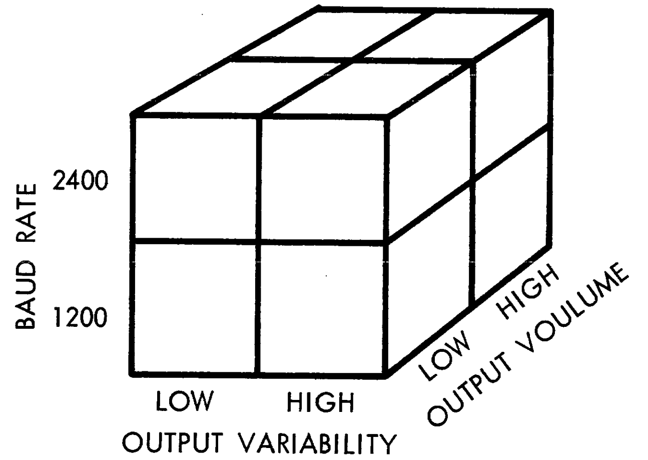


Figure 1—2×2×2 factorial design

independent variables selected for this study were:

Output baud rate	1200	2400
Output rate variability	Low	High
Output volume	<1000 Chars.	>1000 Chars.

Performance measures included the total time to complete a series of message search and retrieval tasks (see Appendix 2 for the tasks performed) and the number of functions used (keystrokes) in completing the tasks; CPU time used in completing the tasks was measured and gives a gross measure of computer performance. A questionnaire (see Appendix 3) was administered to the subjects after finishing all of the tasks, as a means of eliciting their attitude toward a number of system features and the overall interactive environment. The questionnaire consisted of 18 questions which required the subject to rate on a five-point scale his attitudes toward four different groups of system properties: input language and sufficiency of commands; physical characteristics of the display, the CRT and keyboard; the speed and variation in speed of the system; and the overall utility of the message processing system.

THE SYSTEM

The system used to test the influence of the independent variables on the performance measures is an interactive message retrieval system in use at ISI and other locations. It works on text files which conform to a standard message format. The program is routinely used by all of the subjects of these experiments; the subjects required essentially no additional training. This system has been modified to permit performance measurements to be taken on-line. Further modifications were made to the program in order to mold it to the simulated travel department environment of this study. The data base consisted of approximately 200 travel request messages (see Appendix 1 for examples of messages from the data base).

The usual result of a search through the data base is a listing of just the headers of the messages satisfying the search request. From these headers, the user may specify the exact message or messages to be displayed on the screen. If the user has reason to believe that the selected messages will not be too numerous, or knows that he will want to read all of the selected messages, he may have the system immediately begin typing them on the screen rather than first displaying the headers.

THE SUBJECTS

Subjects for this study represent a population of experienced interactive message search and retrieval system users. Such individuals might include, but are not limited to, librarians or other users of interactive bibliographic systems, airline reservations personnel, hotel reservations service personnel, users of data base management or retrieval systems, etc.

Actual subjects used for these experiments were members of the professional and secretarial staff of ISI. In order to minimize training time and to better represent a population of experienced interactive computer users, subjects were taken from those who have had some experience in using the system. Though most users of interactive computer systems do require training and experience with a particular system in order to reach maximum efficiency, the learning and adaptive phase represents only a very small fraction of the total time in which they will be interacting with the system. Subjects were randomly selected from the entire ISI staff. The number of subjects used was 9 per cell, a total of 36. Each subject was randomly assigned to one of the four cells of the factorial design.

EXPERIMENTAL SETTING

The experiments were conducted in an office at ISI (see Figure 2). The room contained the usual ISI office furniture, including a Hewlett-Packard 2640A CRT and keyboard (HP), the computer terminal which all participants in

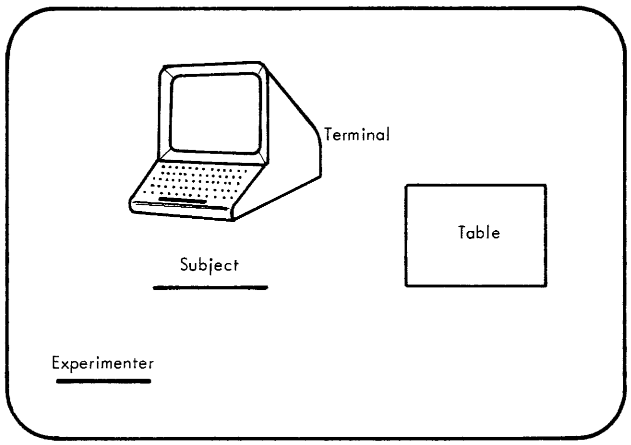


Figure 2—Experimental arrangement

the experiments ordinarily use for a large part of their daily activities, and a table with answer sheets for writing responses to the series of tasks to be performed. The HP includes a 24 line by 80 character (5 inch by 10 inch) rectangular CRT display and keyboard. The normal display rate is switch-selectable from 110 baud to 2400 baud. At ISI, terminals are used at the 2400 baud rate (approximately 240 characters per second). To simulate the 1200 baud display rate used in these experiments, each line of output was interleaved with "null" characters, which produce no output on the screen, but have a bit string and are handled as an ordinary character, in order to increase the display time for a given output string by a factor of two.

For each of the tasks, the task question appeared in a reserved area at the top of the screen, where it would remain until the subject pressed the "N" key to go to the next task. All requested output would then appear below the reserved area and would scroll in the normal manner. Figure 3 indicates the way the screen looked to the subject with the first sample task description in the reserved area, and sample output in the working area.

Each task required the subject to read or count a number of messages and write the appropriate information on the answer sheets provided. After all of the tasks were completed, the instructions for the post-test questionnaire appeared on the screen. The subject completed the questionnaire, including an open-ended question which allowed him to express his general comments on the system and, in particular, to comment on any areas about which he might have felt strongly but which were not adequately covered by the previous questions. Each subject's total time in the experimental session varied with the particular combination of independent variables experienced; the average was about 1 to 1½ hours.

DATA ANALYSIS

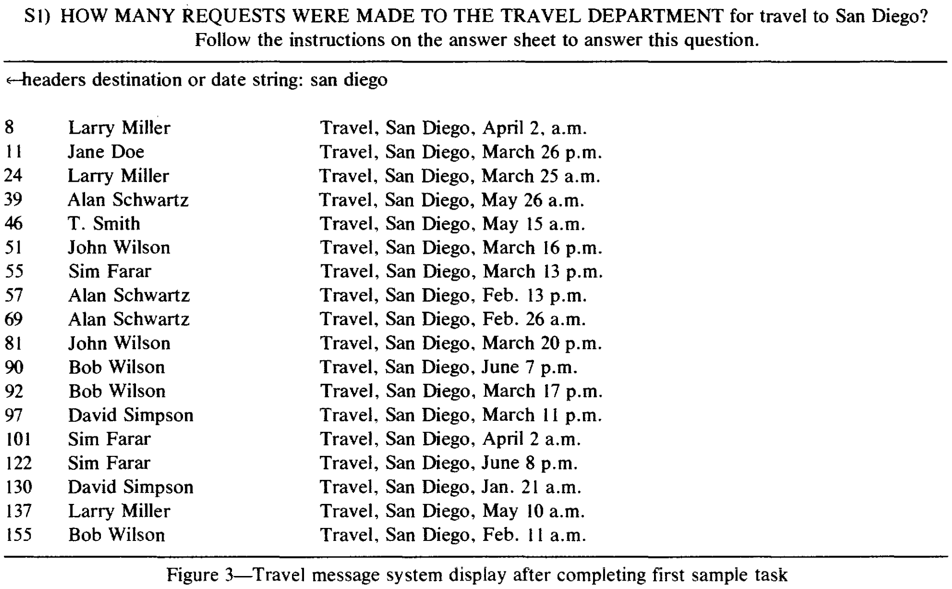
Classical analysis of variance[8] provides the framework for testing the significance of the observed performance differences between the various treatment conditions of the factorial design. For the post-test questionnaire, the experimental design is simply a 2×2 factorial design, two levels of the nominal output rate (1200 baud vs. 2400 baud), and two levels of output rate variability (see Figure 4). Analysis of variance was used for testing for significance the differences in responses to the questionnaire items. Initially, each question was analyzed independently. Then an index was constructed which was just the average response to the 18 questions. Finally, indices were constructed giving the average response to the questions within the four groupings described above. The results of the task and questionnaire data are presented below.

RESULTS

Data was pooled across all 11 tasks. Additionally, tasks were divided into ones requiring low output volume and those requiring high output volume. The task with the median output volume was eliminated in order to further enforce the high-low dichotomy.

The main conclusion, that there is a significant effect for output variability on user performance, is supported, p<.05, across all volume levels. Analysis of variance summary tables for the repeated measures design, with repeated measures across the two volume levels (Table I(a)-(c)), appear below, and support this conclusion.

In Table I(a), note that the effects for baud, var x baud,



S1) HOW MANY REQUESTS WERE MADE TO THE TRAVEL DEPARTMENT for travel to San Diego?
Follow the instructions on the answer sheet to answer this question.

```
←headers destination or date string: san diego

8      Larry Miller       Travel, San Diego, April 2, a.m.
11     Jane Doe           Travel, San Diego, March 26 p.m.
24     Larry Miller       Travel, San Diego, March 25 a.m.
39     Alan Schwartz      Travel, San Diego, May 26 a.m.
46     T. Smith           Travel, San Diego, May 15 a.m.
51     John Wilson        Travel, San Diego, March 16 p.m.
55     Sim Farar          Travel, San Diego, March 13 p.m.
57     Alan Schwartz      Travel, San Diego, Feb. 13 p.m.
69     Alan Schwartz      Travel, San Diego, Feb. 26 a.m.
81     John Wilson        Travel, San Diego, March 20 p.m.
90     Bob Wilson         Travel, San Diego, June 7 p.m.
92     Bob Wilson         Travel, San Diego, March 17 p.m.
97     David Simpson      Travel, San Diego, March 11 p.m.
101    Sim Farar          Travel, San Diego, April 2 a.m.
122    Sim Farar          Travel, San Diego, June 8 p.m.
130    David Simpson      Travel, San Diego, Jan. 21 a.m.
137    Larry Miller       Travel, San Diego, May 10 a.m.
155    Bob Wilson         Travel, San Diego, Feb. 11 a.m.
```

Figure 3—Travel message system display after completing first sample task

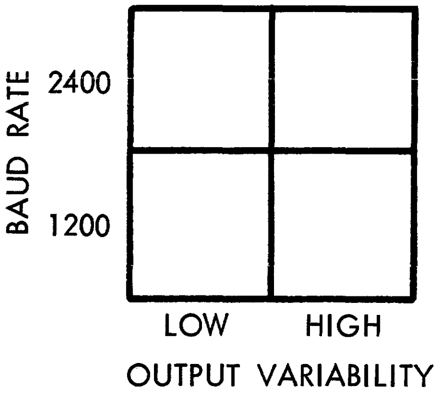


Figure 4—2×2 factorial design

vol x baud, and the triple interaction are not significant (p>.05).

It is instructive at this point to view the data as a 2×2 factorial design combining low and high Baud groups. The results are plotted below (Figure 5). This more clearly shows the effects of increased output variability and increased output volume.

The nominal baud rates for the low and high variability conditions and the low and high baud rate conditions are indicated in Figure 6. The numbers in brackets indicate the average baud rate over the row or column, as appropriate.

Since the high variability conditions yielded a nominal baud rate of 1800, and the low variability conditions one of 900, we might expect differences in performance to accom-

TABLE I(a)—Analysis of Variance Summary Table

Independent Variable: Total time in seconds to complete tasks

SOURCE	SS	df	MS	F	p
Variability	1370340	1	1370340	6.10	<.05
Baud	398	1	398	<1.00	N.S.
Var×Baud	54	1	54	<1.00	N.S.
Error (bet)	7196608	32	224894		
Vol	17743917	1	17743917	233.30	<.01
Var×Vol	583020	1	583020	7.67	<.01
Baud×Vol	3160	1	3160	<1.00	N.S.
Var×Vol×Baud	2962	1	3296	<1.00	N.S.
Error (w/in)	2434000	32	76063		

TABLE I(b)—Analysis of Variance Summary Table

Independent Variable: CPU time used

SOURCE	SS	df	MS	F	p
Variability	9.90	1	9.90	<1.00	N.S.
Baud	1245.80	1	1245.80	19.60	<.01
Var×Baud	4.7	1	4.70	<1.00	N.S.
Error (bet)	2042.60	32	63.80		
Vol	1099.00	1	1099.00	17.90	<.01
Var×Vol	26.00	1	26.00	<1.00	N.S.
Baud×Vol	599.20	1	599.20	9.80	<.01
Var×Baud×Vol	3.30	1	3.30	<1.00	N.S.
Error (w/in)	1962.00	32	61.30		

TABLE I(c)—Analysis of Variance Summary Table

Independent Variable: Keystrokes used

SOURCE	SS	df	MS	F	p
Variability	80.20	1	80.20	<1.00	N.S.
Baud	288.00	1	288.0	<1.00	N.S.
Var×Baud	312.50	1	312.50	<1.00	N.S.
Error (bet)	14095.60	32	440.5		
Vol	53.40	1	53.40	<1.00	N.S.
Var×Vol	450.00	1	450.00	1.50	N.S.
Baud×Vol	555.60	1	555.60	1.90	N.S.
Var×Baud×Vol	501.40	1	501.40	1.70	N.S.
Error (w/in)	9570.00	32	299.10		

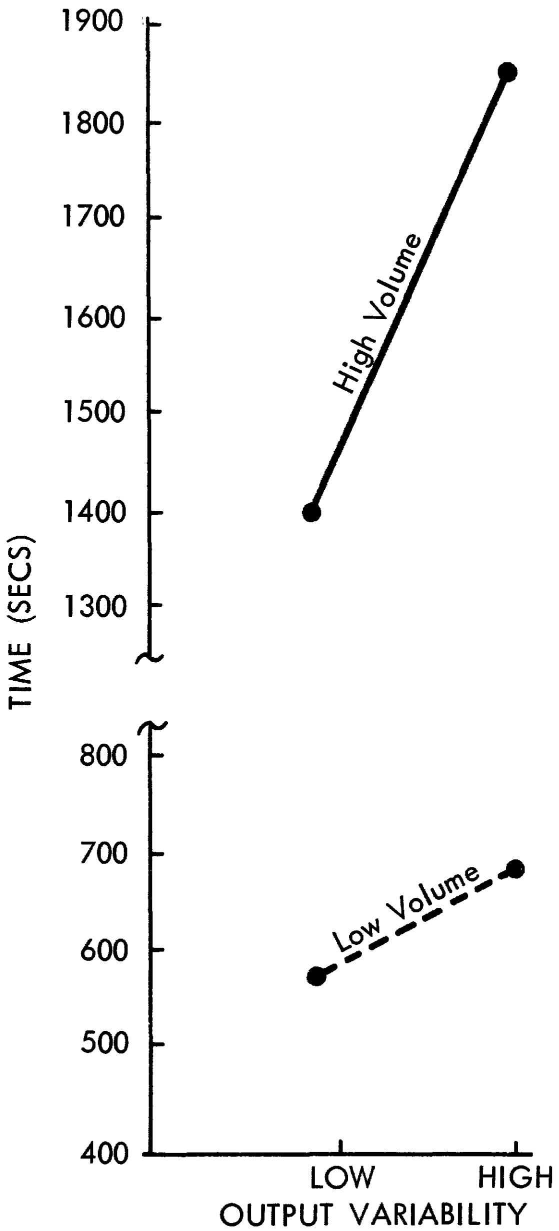


Figure 5—Graph of time to complete tasks vs. output variability for low volume and high volume

pany these differences. Similarly, the average difference in baud rate is also 900 vs. 1800. But here, no significant performance differences are observed, p>.05. Thus the significant performance difference between the low and high output variability groups can *not* be accounted for by differences in nominal output rates.

In Figure 6, note that there are two cells with identical nominal output rates—2400, high variability, and 1200, low variability. Each yields a nominal 1200 baud output rate. It seems clear that any performance differences between these two cells can be attributed to output variability differences only. A t-test was performed comparing these two cells of the factorial design. The mean difference in time to complete the tasks between cells was significant: t=4.28, p<.01.

POST-TEST QUESTIONNAIRE

The 18 questions making up the post test questionnaire may be thought of as comprising an "index of satisfaction" of the user with the system. If we simply average the responses of each subject this average may be considered the satisfaction index. Figure 7 presents the average response for the subjects in each of the four cells of the factorial design graphically.

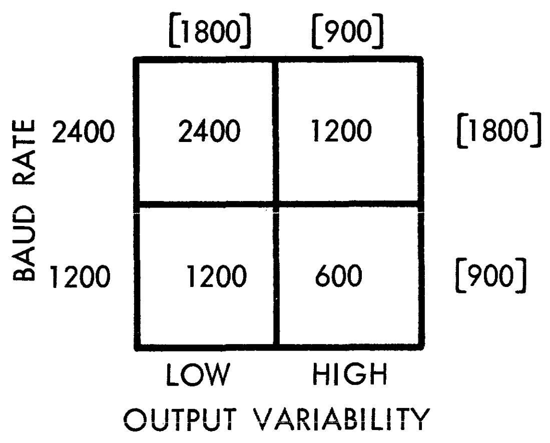
An analysis of variance was performed using this index as the dependent measure. The results of the analysis of variance are presented in Table II. We note that there is a significant effect for output variability on the average response to the questionnaire, F(1,24)=19.4, p<.01. The effects of baud rate and var x baud are not significant.

This analysis allows an unequivocal view that users experiencing the high variability versions of the system expressed a significantly lower view of the system, its commands, its display, its speed and its overall utility than those experiencing the low variability versions of the system.

It is useful to examine the average response of the questions in each of the four question groups. For ease of later reference, the groupings of questions will be denoted [C] for 1-4 (commands), [D] for 5-8 (display), [S] for 9-13, (speed), and [U] for 14-18 (utility). Table III presents the analysis of variance summary tables for the average response within groups.

When the answers are viewed as groups, each representing a common component of the interactive environment, we find that there are significant differences between subjects in low vs. high variability groups in the average response within the three groups [C], [S] and [U], but not for [D].

Figure 8 presents graphical results of the average response to the questions within the four groups ([C], [D], [S] and [U]), vs. output variability, for 1200 and 2400 baud.



		[1800]	[900]	
BAUD RATE	2400	2400	1200	[1800]
	1200	1200	600	[900]
		LOW	HIGH	
		OUTPUT VARIABILITY		

Figure 6—Nominal baud rates

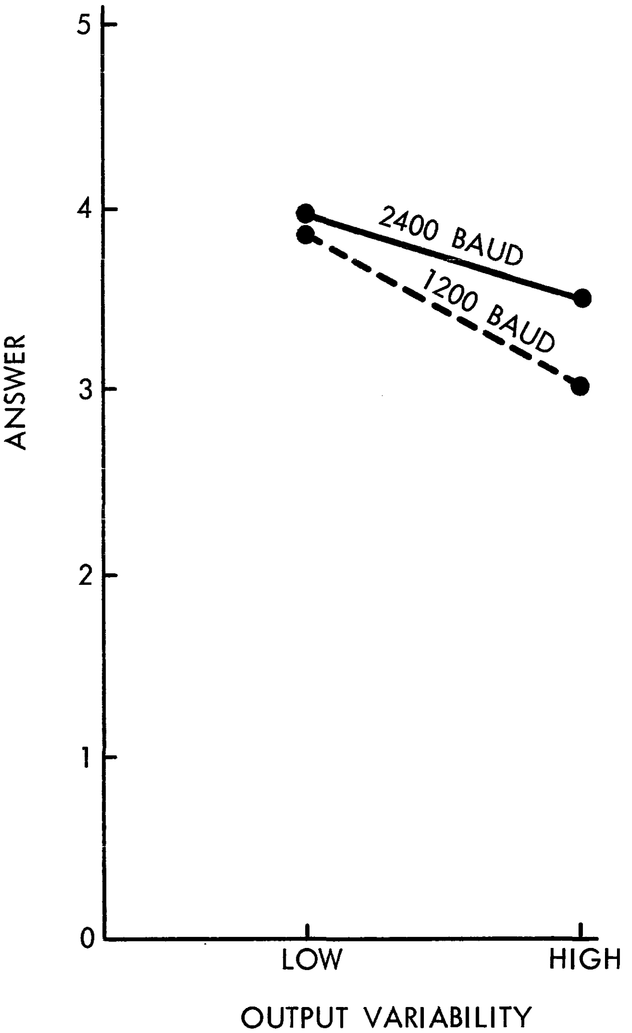


Figure 7—Graph of average response to post-test questionnaire vs. output variability for 1200 and 2400 baud

TABLE III—Anlaysis of Variance summary tables for the Post-Test Questionnaire by groups: [C]—questions 1-3; [D]—questions 4-8; [S]—questions 9-13; [U]—questions 14-18

SOURCE	SS	df	MS	F	p
[C]					
Variability	1.38	1	1.38	3.93	~.05
Baud	0.14	1	0.14	0.40	N.S.
Var×Baud	0.39	1	0.39	1.1	N.S.
Error	8.45	24	0.35		
[D]					
Variability	0.93	1	0.93	2.45	N.S.
Baud	2.04	1	2.04	5.41	<.05
Var×Baud	0.13	1	0.13	0.35	N.S.
Error	9.06	24	0.38		
[S]					
Variability	7.26	1	7.26	16.24	<.01
Baud	0.04	1	0.04	0.09	N.S.
Var×Baud	0.04	1	0.04	0.09	N.S.
Error	10.73	24	0.45		
[U]					
Variability	2.64	1	2.34	13.63	<.01
Baud	0.83	1	0.83	4.31	<.05
Var×Baud	1.02	1	1.02	5.26	<.05
Error	4.65	24	0.19		

DISCUSSION AND ANALYSIS OF TASK RESULTS

Variability effects

There is a significant difference in time to complete the tasks, across the output variability. Further comparisons of the two conditions, 2400 baud/high variability vs. 1200 baud/low variability were done in order to ascertain the possibility of confounding effects. These two cells produced equivalent nominal output rates: 1200 baud (see Figure 6). The variability algorithm was designed such that the total time to display N characters on the screen would be approximately double the amount of time to display the same N characters without the variability. Therefore, any performance differences between these two conditions can reasonably be ascribed to variability of output differences rather than total output time differences. The result of the comparison of performance differences between these two conditions is that there is a significant difference in the time to complete the tasks. The amount of CPU time used also varied but this may be attributed to the additional processing needed to implement the 1200 baud display rate, and the high output variability. There was no significant difference, however, in the number of keystrokes used in performing the tasks.

TABLE II—Analysis of Variance Summary Table

Independent Variable: Average answer to 18 questions of post-test questionnaire

SOURCE	SS	df	MS	F	p
VARIABILITY	2.77	1	2.77	19.4	<.01
BAUD	0.43	1	0.43	3.02	N.S.
VAR×BAUD	0.29	1	0.29	2.05	N.S.
ERROR	3.40	24	0.14		

The conclusion is that increasing the variability of computer output is associated with significantly decreased user performance in the interactive tasks. To the extent that the test population for these experiments represents a broader category of potential interactive system users, and the test system is representative of a broader class of interactive systems, we may conclude that variability in display rate has per se a detrimental effect on user performance.

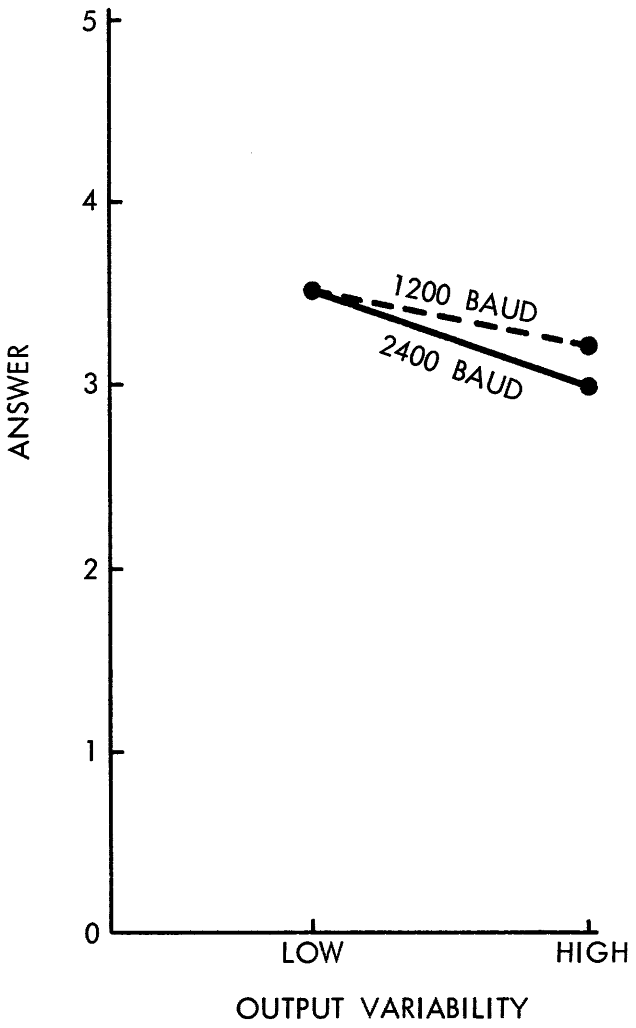


Figure 8(a)—Graph of average response to the (C) questions vs. output variability

Baud rate effects

Presenting the requested information at 1200 baud vs. 2400 baud produced *no* significant differences in performance. In particular, even though the nominal baud rate was 1800 for the 2400 baud groups and 900 for the 1200 baud groups (see Figure 6), there was *no* significant difference in time to complete the tasks for the two groups. This result holds across high volume as well as low volume tasks (i.e., the baud × volume interaction was not significant). It appears that both 1200 and 2400 baud display rates are faster than the typical subject can read, so that time to read a page of material depends on the individual's reading speed rather than system display rates. One would then conjecture a plateau in the curve of time to read a screen of text vs. display rate, somewhat like the one below (Figure 9). Apparently the plateau is reached at display rates of less than 1200 baud. In fact, 1200 baud corresponds to approximately 1200 words per minute, a rate faster than the rate at which the average person reads.

However, the result is still somewhat curious. A number of tasks required the subject to visually search message bodies for particular names. While it would seem reasonable to expect that doubling the display rate should lead to shorter times in completing those kinds of tasks, this was not observed to occur. We may conclude that doubling the display rate from 1200 to 2400 baud does not produce improved performance for the subjects and system of these experiments. It should also be noted that the total time to present the typical amount of material was only about seven percent of the total time subjects needed to perform the individual tasks.

This result makes the one involving output variability seem that much stronger. Examining Figure 6 again, we note that there was a difference in average display rates across the two variability conditions. It is not immediately clear whether the effect for variability might be confounded with the average display rate. That there is no significant difference across baud rate leads us to reject that possibility.

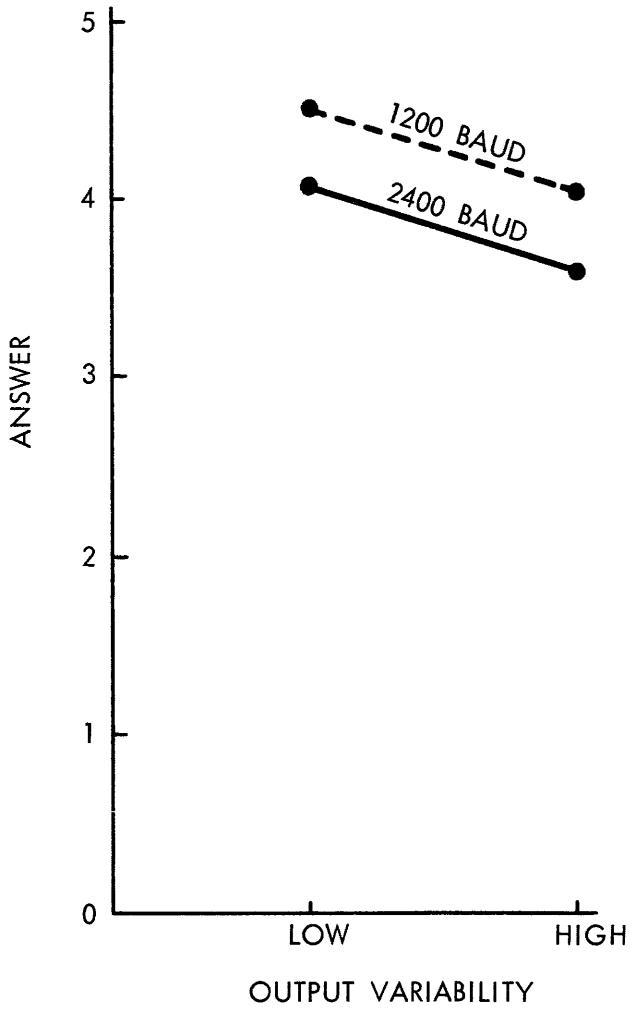


Figure 8(b)—Graph of average response to the (D) questions vs. output variability

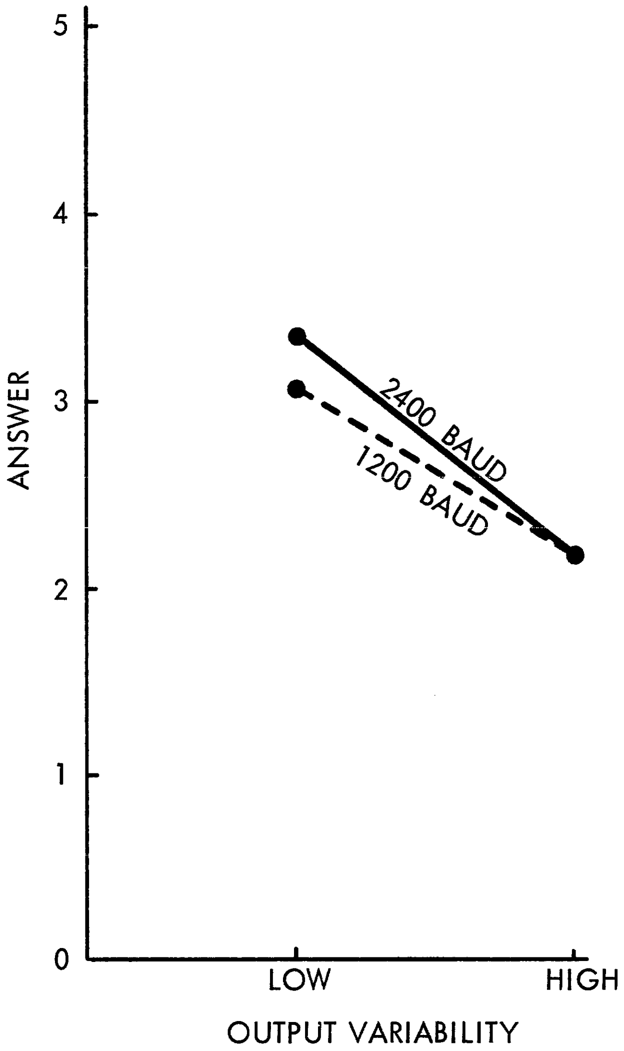


Figure 8(c)—Graph of average response to the (S) questions vs. output variability

DISCUSSION AND ANALYSIS OF POST-TEST QUESTIONNAIRE

A fundamental conclusion, which is supported both by the task data and the questionnaire data, is that the nominal output baud rate, at 1200 baud vs. 2400 baud, has at best a very weak effect upon the user's performance and attitude towards the system. Specifically, over a number of tasks, involving both low and high output volumes, there was *no* significant performance difference between those receiving the 1200 baud version and those receiving the 2400 baud version. This result is observed across the low and high volume tasks, and across the low and high output variability groups.

Examining the questions by groups, we notice a strong effect of output variability upon the attitudes of the users toward the system on three of the four question groups. Subjects experiencing the high variability conditions had a lower response index to the questions in the [C], [S] and [U] groups. Examining more closely the meaning of the [C], [D], [S] and [U] indices, the following are concluded:

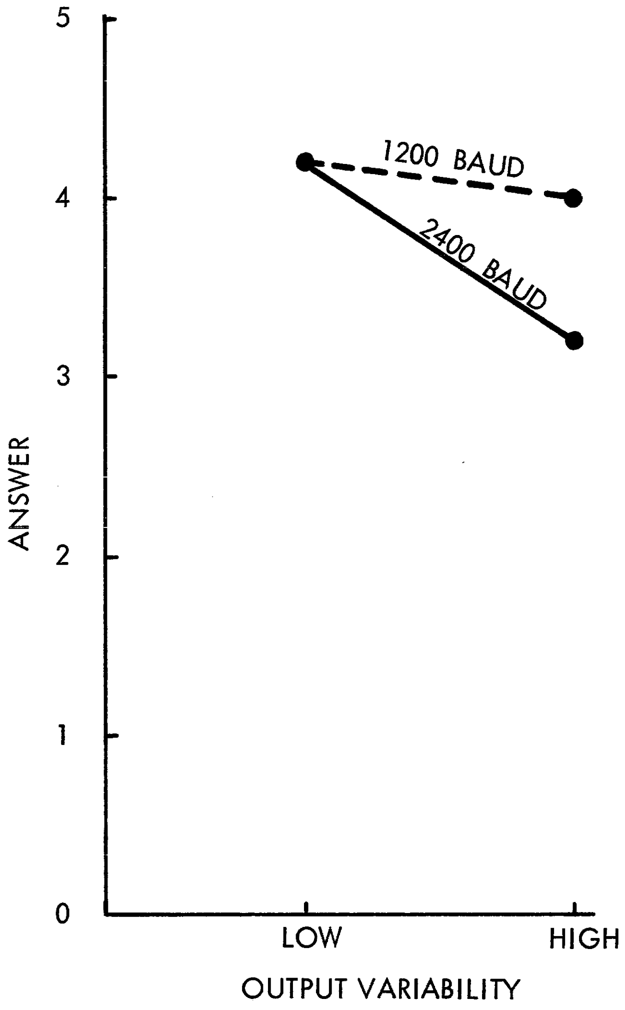


Figure 8(d)—Graph of average response to the (U) questions vs. output variability

(1) High output variability subjects perceived the command structure as less adequate to their needs. As

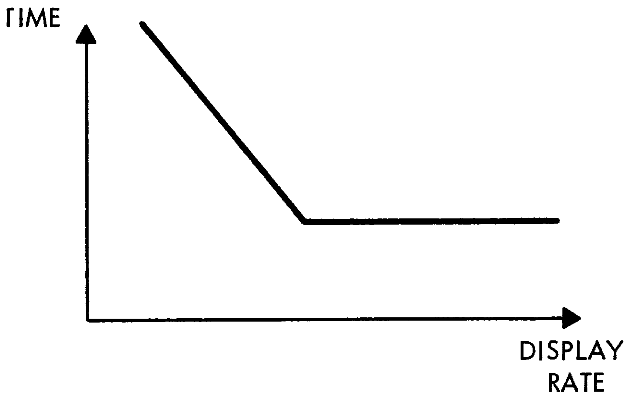


Figure 9—Graph of time to read screenful of material vs. display rate

indicated earlier, the experimental design assumed a fixed terminal type with its own display characteristics. The HP terminal used in these studies works on a scrolling method where each new line of output is presented on the bottom of the screen and all lines above scroll up. The top-most line is lost as each new line is appended at the bottom. Some of the apparent dissatisfaction with the command language (and the terminal itself) may be associated with the scrolling typical of the HP and other terminals.

(2) High output variability subjects were less satisfied with the physical display. Though this result is somewhat ambiguous, the general conclusion is that increasing the variability of the output display rate reduces the user's overall image of the system. Put another way, users with low variability of output seemed more likely to find the particular display satisfactory.

(3) Users who experienced the high output variability version were bothered by the slowness of the system, and noticed the reduced speed and the increased variability of the output rate. However, merely cutting the output rate in half (from 2400 baud to 1200 baud) did not produce a noticeable reduction in the answers for users in either the high or low variability groups.

(4) In general, subjects experiencing the high variability versions of the system had a lower view of the overall utility of the system, as evidenced by their average response to questions concerning the usefulness of the system, the desirability of using the system vs. performing the tasks by hand, need for more materials, and their overall rating of input to and output from the computer.

Examining the intercorrelations between the answers to the questionnaire provides a useful insight into those parts of the system which are viewed as a whole. For example, looking at those questions which correlate significantly with the [S] questions, 9-13 (Table IV) allows one to identify those aspects of the interactive system with which a user is least satisfied as the system becomes more stressful. It would be expected, of course, that there would be significant correlations between questions within the [S] grouping, and this is observed. Identifying those questions outside of the [S] group which correlate with questions within the group, the following are concluded:

TABLE IV

Question	Questions with which it correlates significantly ($p<.05$)
9	10,11,12,13
10	4,5,8,9,12
11	9,12,13,14,18
12	1,5,9,10,13
13	9,11,12,16,18

(1) Those who perceived the system as being slower had a significantly poorer view of the ease of using the commands and the overall utility of the system, felt a need for more materials on the system, and generally had a lower overall view of system output, than those who perceived the system as being relatively faster.

(2) Those who perceived the system as being relatively high in variability of output and processing speed had a significantly lower view of the ease of using the commands, found the brightness and size of the display screen less satisfactory and found that the data presented was less sufficient for their needs, than those who perceived the system as having relatively little variability in output or processing speed.

CONCLUSIONS AND RECOMMENDATIONS

The major emphasis of the research presented in this report is that there are a number of parameters of the man-machine interaction which affect the performance of the user. Specifically, it was hypothesized that changes in the nominal display rate of the presentation of computer output, and the variability in the display rate, would have significant effects on the performance and attitudes of the users of the man-machine system.

This research has found that doubling the display rate of system output to the user of an interactive message processing system does not improve performance, nor does it lead to an improved view of the system or attitude toward the system on the part of the user. What then are the effects of increasing system output, and what ways might be effective in both improving user performance and improving attitude? At this point, a confounding problem occurs. It has been a (seemingly not unreasonable) assumption on the part of system designers that increasing the display rates leads to better performance in interactive systems. If we could guarantee that the variability in the display rate were held constant as the display rate was increased, the results of these experiments allow us to conclude that performance and attitude are not diminished. They may even be improved, though this does not appear to be the case in this research. So there is certainly no immediately apparent drawback to providing faster displays. However, as systems become heavily loaded, increased display rates are associated with increases in the variability. The actual display rate may not be improved, and the results of this research strongly demonstrate that performance is decreased and that user attitudes towards the system deteriorate. These conclusions are so strongly supported by the data presented that a general recommendation to system designers would have to be that increasing output display rates should not be attempted without a corresponding increase in CPU power in order to guarantee consistency in the output display rate.

APPENDIX 1—SAMPLE MESSAGES FROM THE DATA BASE

—

To: TRAVEL DEPT.
From: Alan Schwartz
Subject: Travel, San Francisco, Feb. 2 a.m.
Date: 31 JAN 76 1303-PST
Message:
Please reserve 2 seats to San Francisco on Feb. 2 a.m. for me and Arnold Serkin
Return: OPEN
Thanks

—

To: TRAVEL DEPT.
From: David Simpson
Subject: Travel, Des Moines, Jan. 4 p.m.
Date: 31 JAN 76 1303-PST
Message:
Please reserve 4 seats to Des Moines on Jan. 4 p.m. for me and Jane Doe
Arnold Serkin
John Wilson
Return: Jan. 8
Thanks

—

To: TRAVEL DEPT.
From: Arnold Serkin
Subject: Travel, Miami, April 23 a.m.
Date: 31 JAN 76 1303-PST
Message:
Please reserve 3 seats to Miami on April 23 a.m. for me and Sim Farar
Alan Schwartz
Return: April 27
Thanks

—

To: TRAVEL DEPT.
From: Larry Miller
Subject: Travel, San Diego, April 2 a.m.
Date: 31 JAN 76 1303-PST
Message:
Please reserve 3 seats to San Diego on April 2 a.m. for me and
David Simpson
John Wilson
Return: April 6
Thanks

—

APPENDIX 2—TASKS TO BE PERFORMED

S1) HOW MANY REQUESTS WERE MADE TO THE TRAVEL DEPARTMENT for travel to San Diego? Follow the instructions on the answer sheet to answer this question.

S2) WHO WANTED TO GO TO DES MOINES DURING THE MONTH OF JANUARY? Again, follow instructions on the answer sheet to answer this question.

1) WHO WANTED TO GO TO LONDON IN MARCH? If there is more than one person who wanted to go, write down all of their names. If nobody wanted to travel to London in March, write "NONE."

2) WHO WANTED TO GO TO KANSAS CITY DURING THE MONTH OF FEB? Answer this question similarly to the previous question.

3) WHO WANTED TO GO TO PORTLAND DURING THE MONTH OF FEB? Answer this question similarly to the previous questions.

4) WHO WANTED TO GO TO MIAMI ON FEB. 2? Answer this question similarly to the previous questions.

5) WHO WANTED TO GO TO SAN DIEGO ON APRIL 2? Answer this question similarly to the previous questions.

6) HOW MANY REQUESTS FOR TRIPS TO SEATTLE ARE THERE IN THE DATA BASE?

7) WHO TOOK THOSE TRIPS, and how many trips did each of these people take to Seattle.

8) FOR THOSE WHO TOOK FIVE OR MORE TRIPS TO SEATTLE, to which other cities did they REQUEST travel?

9) LIST THE LOCATIONS AND REQUESTED DATES OF TRAVEL, that Alan Schwartz made, where he requested Sim Farar to also travel. Similarly, list locations and dates of travel where Sim Farar requested travel with Alan Schwartz.

10) LIST THE DATES WHEN ALAN SCHWARTZ AND SIM FARAR BOTH TRAVELED TOGETHER TO SEATTLE.

11) ON THOSE OCCASIONS WHERE BOTH SIM FARAR AND ALAN SCHWARTZ TRAVELED TOGETHER TO SEATTLE, list those who also traveled with them.

APPENDIX 3—POST-TEST QUESTIONNAIRE

You are now requested to answer a brief series of questions concerning your opinions of the computerized system you've just been using. It is important that you answer these questions with the answer that best represents your attitude to the particular area of the question. Specifically, the questions require you to numerically rate certain aspects of the computerized system. Even though you may not have a strong feeling one way or the other, please select one of the numerical ratings that best characterizes your attitude to that particular area.

You will note that the questions are answered using the computer. Please be careful that you select the correct number for your answer. If you make an error, press the "DEL" key. When you are satisfied with your answer for that particular question, press the "RETURN" key.

Please answer the following series of questions with a numerical rating in the range of 1-5. 1 = Very Poor, Unacceptable, etc. 5 = Excellent, Completely Acceptable, Easy to Use, etc. (1-2 implies a generally negative response, 4-5 a generally positive one.) However, a specific numerical scale will be given for each question.

1) COMMANDS: EASE OF USE—
1=Difficult to use
3=Easy to use, but somewhat confusing
5=Easy to use, no confusion as to meaning

2) COMMANDS: CLEAR AND MEANINGFUL FUNCTIONS—
1=Commands produced results completely different from what was expected.
3=Some commands were clear and simple, others were very confusing.
5=All commands were completely clear.

3) COMMANDS—
1=Would liked to have had a number of additional commands available to make the tasks easier to accomplish.
3=Some additional commands would have been useful.
5=Available commands were completely adequate to accomplish tasks.

4) SCREEN: BRIGHT ENOUGH?
1=Too dim, completely unreadable.
3=Too dim, but readable
5=Brightness just right.

5) SCREEN: LARGE ENOUGH?
1=Screen size too small, completely unreadable
3=Screen size too small, but readable
5=Screen size just right

6) CHARACTERS: LEGIBLE, ADEQUATE SIZE, ETC.—
1=Characters too small or awkwardly shaped
3=Character size and shape adequate, but some difficulty in reading
5=Character size and shape just right

7) PRINTING FORMAT: READABLE?
1=Format unclear, jumbled, etc. Unreadable.
3=Format readable, but not outstanding.
5=Format excellently arranged and completely readable.

8) PRINTING FORMAT: SUFFICIENT DATA?
1=Completely insufficient data to adequately complete tasks.
3=Just barely sufficient data, but would have been able to utilize more.
5=Data presented was completely adequate to complete tasks.

9) COMPUTER SYSTEM SPEED—
1=Too slow
3=Just right
5=Too fast

10) VARIATION IN COMPUTER SYSTEM SPEED—
1=So much variation in computer and printing speed that system was difficult and bothersome.
3=Some variation in computer speed and printing speed, but not enough to be bothersome.
5=Little or no variation in the speed of the computer system.

11) PRINTING SPEED—
1=Too slow
3=Just right
5=Too fast

12) VARIATION IN PRINTING SPEED—
1=Far too much variation for easy reading of output
3=Some variation, but no great difficulty in reading
5=Output was smooth and easy to read

13) PROCESSING TIME—
1=System took way too long to do what should have been simple tasks.
3=System took about the time you would have expected.
5=Too fast, felt rushed, etc.

14) WAS THE TRAVEL MESSAGE PROCESSING SYSTEM USEFUL IN ANSWERING THESE QUESTIONS?
1=Completely useless, confusing, etc. Answering the questions was an exercise in futility.
3=Found the system marginally useful, but some aspects were difficult to use, too slow, confusing, etc.
5=Completely useful, no confusion in the use of the system. Speed of system was just right, easy to adapt to.

15) SUPPOSE YOU HAD TO ACTUALLY ANSWER THE TRAVEL QUESTIONS BY GOING THROUGH THE MESSAGES BY HAND. HOW MUCH IS THE TRAVEL MESSAGES COMPUTER PROCESSING SYSTEM WORTH TO YOU IN ORDER TO SAVE YOU THE EFFORT OF DOING THIS BY HAND?
1=No advantage seen in using the computer system. Would much prefer to perform these tasks by hand.
3=No strong feeling one way or the other.
5=Much prefer using the computer system rather than having to answer these questions by going through the messages by hand.

16) DID YOU FEEL A NEED FOR MORE MATERIALS ON THE FUNCTIONS AVAILABLE IN THE SYSTEM?
1=Available materials were completely useless.
3=What was available was useful, but more information was needed.
5=All available material was useful, no more information was needed.

17) YOUR OVERALL RATING OF INPUT TO THE COMPUTER—
[Use a 1-5 scale as explained in the top portion of the screen.]

18) YOUR OVERALL RATING OF OUTPUT FROM THE COMPUTER—
[Use a 1-5 scale as explained in the top portion of the screen.]

Please type your general comments on the functions provided, their ease of use, and your general feelings of frustration or satisfaction in the use of the system. Be certain to address yourself to your feelings in regards to the delays in output, and the general speed of the system, particularly if the load average was high and you noted unacceptable delays in system performance.

REFERENCES

1. Willmorth, N. E., "Human Factors Experimentation in Interactive Planning," in Sackman, H. & Ronald L. Citrenbaum (eds.), *ONLINE PLANNING, Towards Creative Problem Solving,* Prentice Hall, 1972, pp. 281-313.
2. Bennett, John L., "The User Interface in Interactive Systems," in Cuadra, Carlos A. (ed.), *Annual Review of Information Science and Technology,* Vol. 7, 1972, ASIS, Washington, D.C.
3. Walther, George H. and Harold F. O'Neil, Jr., "On-line User-Computer Interface—The Effects of Interface Flexibility, Terminal Type, and Experience on Performance," in AFIPS Conference Proceedings, Vol. 43, 1974, AFIPS Press, Montvale, N. J.
4. Hansen, James V., "Man-Machine Communication: An Experimental Analysis of Heuristic Problem-Solving Under On-Line and Batch-Processing Conditions," in *IEEE Trans. Systems, Man, and Cybernetics,* Vol. SMC-6, No. 11, November, 1976, pp. 746-752.
5. Sterling, Theodor, D., "Guidelines for Humanizing Computerized Information Systems: A Report from Stanley House," in *C. ACM,* Vol. 17, No. 11, November, 1974.
6. Martin, Thomas H., James Carlisle and Siegfried Treu, "The User Interface for Interactive Bibliographic Searching: An Analysis of the Attitudes of Nineteen Information Scientists," in *J. ASIS,* March-April, 1973, pp. 142-147.
7. Miller, Robert B., "Response Time in Man-Computer Conversational Transactions," in *AFIPS Conference Proceedings,* Vol. 33, Part 1, AFIPS Press, Montvale, N.J., 1968, pp. 267-277.
8. Winer, B. J., *Statistical Principles in Experimental Design,* McGraw-Hill, New York, 1971.

Responsive environments

by MYRON W. KRUEGER
The University of Wisconsin
Madison, Wisconsin

ABSTRACT

This paper introduces the concept of a responsive environment which perceives human behavior and responds with intelligent auditory and visual feedback. Several exhibits of responsive environments, implemented by the author, combining computer graphics, video projection and two-way video communication are described. VIDEOPLACE, an evolving exhibit which defines a conceptual telecommunication environment uniting geographically separated people in a common visual experience, is discussed at some length. Based on these examples a new art form of composed man-machine interaction is defined. Finally, practical applications are suggested for the fields of education, psychology and psychotherapy.

INTRODUCTION

Man-machine interaction is usually limited to a seated man poking at a machine with his fingers or perhaps waving a wand over a data tablet. Seven years ago, I was dissatisfied with such a restricted dialogue and embarked on research exploring more interesting ways for men and machines to relate. The result was the concept of a responsive environment in which a computer perceives the actions of those who enter and responds intelligently through complex visual and auditory displays.

Over a period of time the computer's displays establish a context within which the interaction occurs. It is within this context that the participant chooses his next action and anticipates the environment's response. If the response is unexpected, the environment has changed the context and the participant must reexamine his expectations. The experience is controlled by a composition which anticipates the participant's actions and flirts with his expectations.

This paper describes the evolution of these concepts from their primitive beginnings to my current project, VIDEOPLACE, which provides a general tool for devising many interactions. Based on these examples an interactive art form is defined and its promise identified. While the environments described were presented with aesthetic intent, their implications go beyond art. In the final section, applications in education, psychology and psychotherapy are suggested.

GLOWFLOW

In 1969, I became involved in the development of GLOWFLOW, a computer art project conceived by Dan Sandin, Jerry Erdman and Richard Venezsky at the University of Wisconsin. It was designed in an atmosphere of encounter between art and technology. The viewer entered a darkened room in which glowing lines of light defined an illusory space (Figure 1). The display was accomplished by pumping phosphorescent particles through transparent tubes attached to the gallery walls. These tubes passed through opaque columns concealing lights which excited the phosphors. A pressure sensitive pad in front of each of the six columns enabled the computer to respond to footsteps by lighting different tubes or changing the sounds generated by a Moog synthesizer or the origin of these sounds. However, the artists' attitude toward the capacity for response was ambivalent. They felt that it was important that the environment respond, but not that the audience be aware of it. Delays were introduced between the detection of a participant and the computer's response so that the contemplative mood of the environment would not be destroyed by frantic attempts to elicit more responses.

While GLOWFLOW was quite successful visually, it succeeded more as a kinetic sculpture than as a responsive environment. However, the GLOWFLOW experience led me to a number of decisions:

1. Interactive art is potentially a richly composable medium quite distinct from the concerns of sculpture, graphic art or music.
2. In order to respond intelligently the computer should perceive as much as possible about the participant's behavior.
3. In order to focus on the relationships between the environment and the participants, rather than among participants, only a small number of people should be involved at a time.
4. The participants should be aware of how the environment is responding to them.
5. The choice of sound and visual response systems should be dictated by their ability to convey a wide variety of conceptual relationships.
6. The visual responses should not be judged as art nor

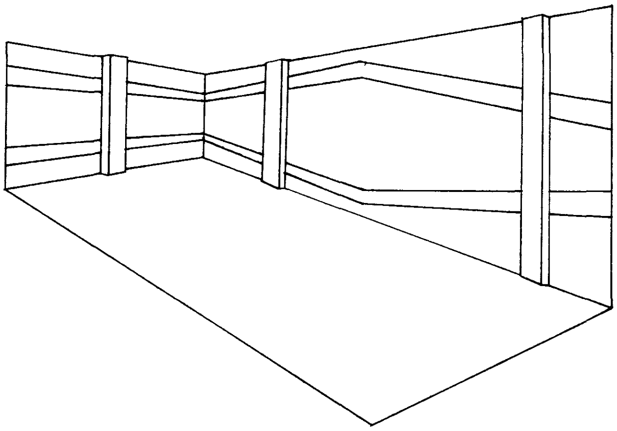


Figure 1—Glowflow tubes on gallery wall

the sounds as music. The only aesthetic concern is the quality of the interaction.

METAPLAY

Following the GLOWFLOW experience, I conceived and directed METAPLAY which was exhibited in the Memorial Union Gallery of the University of Wisconsin for a month in 1970. It was supported by the National Science Foundation, the Computer Science Department, the Graduate School and the loan of a PDP-12 by Digital Equipment Corporation.

METAPLAY'S focus reflected my reactions to GLOWFLOW. Interaction between the participants and the environment was emphasized; the computer was used to facilitate a unique real-time relationship between the artist and the participant. An 8′ by 10′ rear-projection video screen dominated the gallery. The live video image of the viewer and a computer graphic image drawn by an artist, who was in another building, were superimposed on this screen. Both the viewer and the artist could respond to the resulting image.

Hardware

The image communications (Figure 2) started with an analogue data tablet which enabled the artist to draw or write on the computer screen. The person doing the drawing did not have to be an artist, but the term is used for convenience. One video camera, in the Computer Center, was aimed at the display screen of the Adage Graphic Display Computer. A second camera, a mile away in the gallery, picked up the live image of people in the room. A television cable transmitted the video computer image from the Computer Center to the gallery and the two signals were mixed so that the computer image overlayed the live image. The composite image was projected on the 8′ × 10′ screen in the gallery and was simultaneously transmitted back to the Computer Center where it was displayed on a video monitor providing feedback for the artist.

The artist could draw on the Adage screen using a data tablet. By using function switches, potentiometers and the teletype keyboard the pictures could be rapidly modified or the mode of drawing itself altered. In addition to the effects of simple drawings, the image could be moved around the screen, image size could be controlled and the picture could be repeated up to ten times on the screen displaced by variable X, Y and size increments. A tail of a fixed number of line segments could be drawn allowing the removal of a segment at one end while another was added at the opposite end. An image could be rotated in 3-space under control of the pen. Although this was not true rotation, the visual effect was similar. A simple set of transformations under potentiometer and tablet control yielded apparent animation of people's outlines. Finally, previously defined images could be recalled or exploded. While it might seem that the drawing could be done without a computer, the ability to rapidly erase, recall and transform images required considerable processing and created a far more powerful means of expression than pencil and paper could provide.

Interaction

These facilities provided a rich repertoire for an unusual dialogue. The artist could draw pictures on the participants' images or communicate directly by writing words on the screen (Figure 3). He could induce people to play a game like Tic-Tac-Toe or play with the act of drawing, starting to draw one kind of picture only to have it transformed into another by interpolation.

Live graffiti

One interaction derived from the artist's ability to draw on the image of the audience. He could add graffiti-like features or animate a drawn outline of a person so that it appeared to dance to the music in the gallery. The artist tried various approaches to involve people in the interaction. Failing to engage one person, he would seek someone more responsive.

It was important to involve the participants in the act of drawing. However, the electronic wand designed for this purpose did not work reliably. What evolved was a serendipitous solution. One day as I was trying to draw on a student's hand, he became confused and moved it. When I erased my scribblings and started over, he moved his hand again. He did this repeatedly until it became a game. Finally, it degenerated to the point where I was simply tracking the image of his hand with the computer line. In effect, by moving his hand he could draw on the screen before him.

The relationship established with this participant was developed as one of the major themes of METAPLAY. It was repeated and varied until it became an aesthetic medium in iteslf. With each person we involved in this way, we tried to preserve the pleasure of the original discovery. After playing some graffiti games with each group that

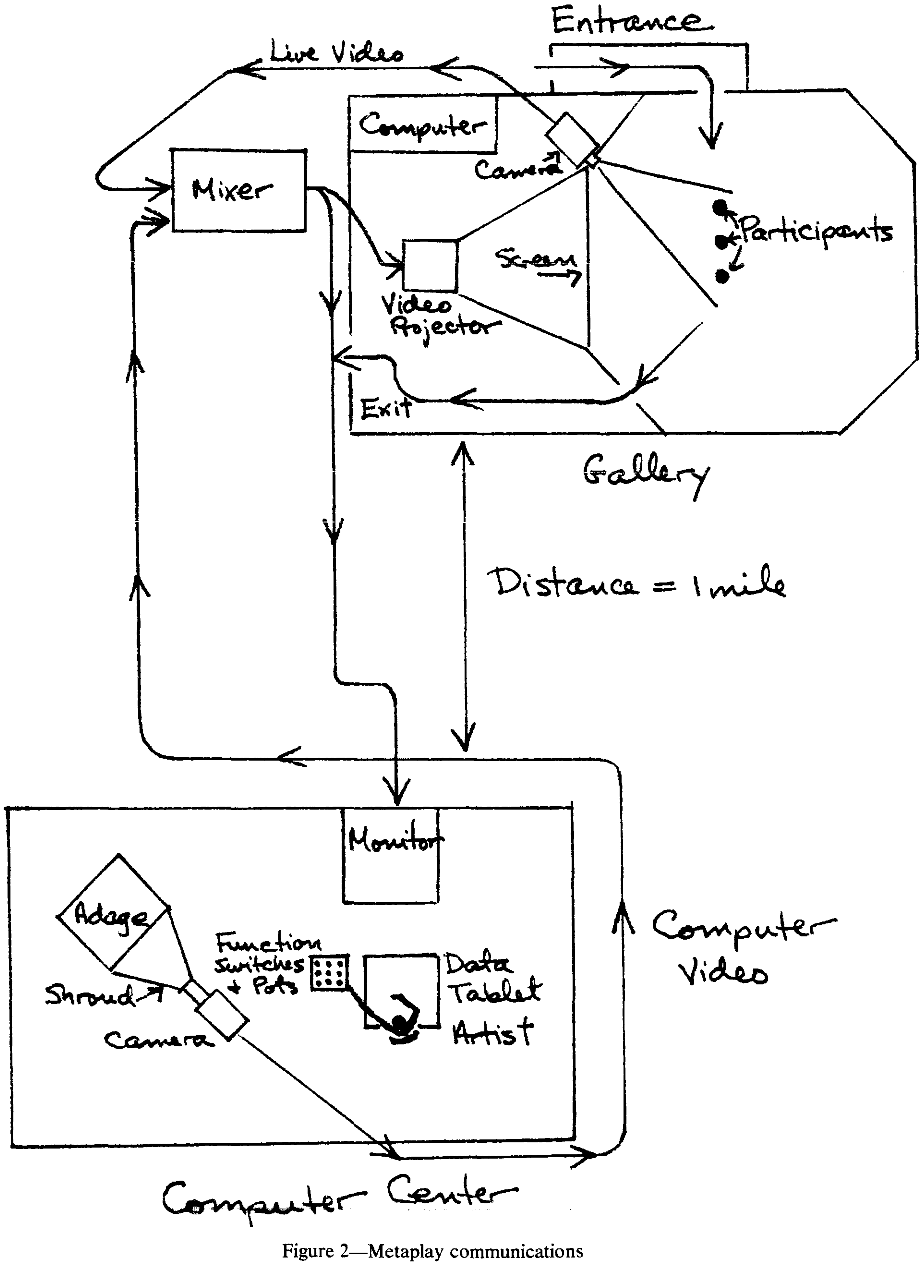


Figure 2—Metaplay communications

entered, we would focus on a single individual and draw around the image of his hand. After an initial reaction of blank bewilderment, the self-conscious person would make a nervous gesture. The computer line traced the gesture. A second gesture, followed by the line was the key to discovery. One could draw on the video screen with his finger! Others in the group, observing this phenomenon, would want to try it too. The line could be passed from one person's finger to another's. Literally hundreds of interactive vignettes developed within this simple communication channel.

Drawing by this method was a rough process. Pictures of any but the simplest shapes were unattainable. This was mainly because of the difficulty of tracking a person's finger. Happily, neither the artist nor the audience were concerned about the quality of the drawings. What was exciting was interacting in this novel way through a man-computer-video link spanning a mile.

PSYCHIC SPACE

The next step in the evolution of the responsive environment was PSYCHIC SPACE, which I designed and exhibited in the Memorial Union Gallery during May and June of 1971. It was implemented with the help of my students, the Computer Science Department and a National Science Foundation grant in Complex Information Processing.

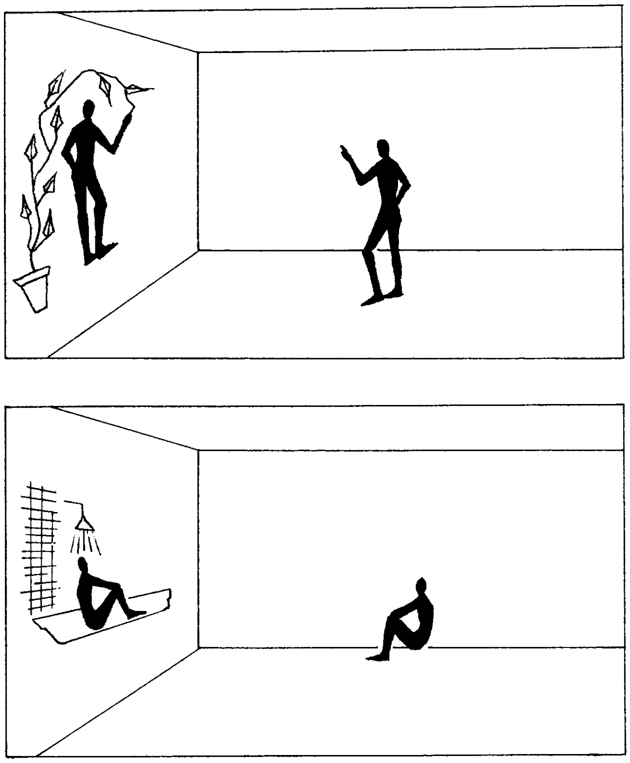


Figure 3—Metaplay drawing

PSYCHIC SPACE was both an instrument for musical expression and a richly composed, interactive, visual experience. Participants could become involved in a softshoe duet with the environment, or they could attempt to match wits with the computer by walking an unpredictable maze projected on an 8′ × 10′ video screen.

Hardware

A PDP-11 had direct control of all sensing and sound in the gallery. In addition, it communicated with the Adage AGT-10 Graphic Display Computer at the Computer Center (Figure 4). The Adage image was transmitted over video cable to the gallery where it was rear-projected on the 8′ × 10′ screen. The participant's position on the floor was the basis for each of the interactions. The sensing was done by a 16′ × 24′ grid of pressure switches, constructed in 2′ × 4′ modules, each containing eight switches (Figure 5). Since they were electronically independent, the system was able to discriminate among individuals if several were present. This independence made it easy for the programming to ignore a faulty switch until its module was replaced or repaired. Since there were 16 bits in the input words of the PDP-11, it was natural to read the 16 switches in each row across the room in parallel (Figure 6). Digital circuitry was then used to scan the 24 rows under computer control.

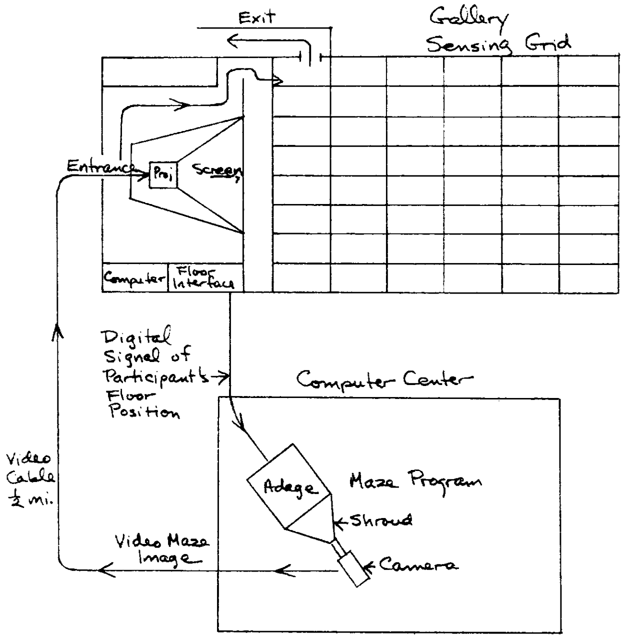


Figure 4—Data and video communication for psychic space

Input and interaction

Since the goal was to encourage the participants to express themselves through the environment, the program automatically responded to the footsteps of people entering the room with electronic sound. We experimented with a number of different schemes for actually generating the sounds based on an analysis of peoples' footsteps. In sampling the floor 60 times per second we discovered that a single footstep consisted of as many as four discrete events: lifting the heel, lifting the toe, putting the heel down and putting the ball of the foot down. The first two were dubbed the "unfootstep." We could respond to each footstep or unfootstep as it occurred, or we could respond to the person's average position. A number of response schemes were tried, but the most pleasing was to start each

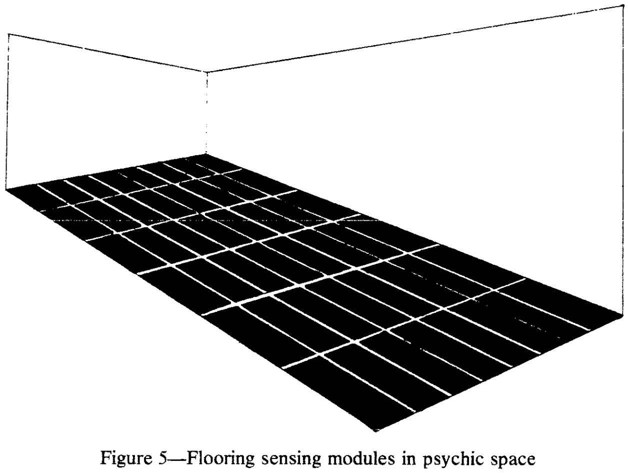
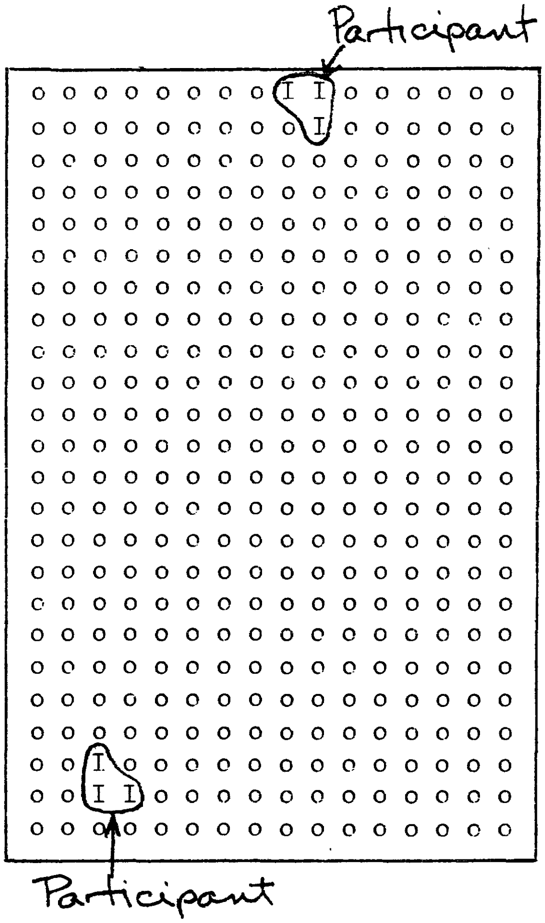


Figure 5—Flooring sensing modules in psychic space



```
o o o o o o o o I I o o o o o o
o o o o o o o o o I o o o o o o
o o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o o
o o I o o o o o o o o o o o o o
o o I I o o o o o o o o o o o o
o o o o o o o o o o o o o o o o
```

Figure 6—Participants' feet are seen by the computer as ones in a field of zeroes

tone only when a new switch was stepped on and then to terminate it on the next "unfootstep." Thus it was possible to get silence by jumping, or by lifting one foot, or by putting both feet on the same switch.

Typical reaction to the sounds was instant understanding, followed by a rapid-fire sequence of steps, jumps and rolls. This phase was followed by a slower more thoughtful exploration of the environment in which more subtle and interesting relationships could be developed. In the second phase, the participant would discover that the room was organized with high notes at one end and low notes at the other. After a while, the keyboard was abruptly rotated by 90 degrees.

After a longer period of time an additional feature came into play. If the computer discovered that a person's behavior was characterized by a short series of steps punctuated by relatively long pauses, it would use the pause to establish a new kind of relationship. The sequence of steps was responded to with a series of notes as before; however, during the pause the computer would repeat these notes again. If the person remained still during the pause, the computer assumed that the relationship was understood. The next sequence of steps was echoed at a noticeably higher pitch. Subsequent sequences were repeated several times with variations each time. This interaction was experimental and extremely difficult to introduce clearly with feedback alone, i.e., without explicit instructions. The desire was for a man-machine dialogue resembling the guitar duel in the film "Deliverance."

MAZE—A COMPOSED ENVIRONMENT

The maze program focused on the interaction between one individual and the environment. The participant was lured into attempting to navigate a projected maze. The intrigue derived from the maze's responses, a carefully composed sequence of relations designed to constitute a unique and coherent experience.

Hardware

The maze itself was not programmed on the PDP-11, but on the Adage located a mile away in the Computer Center. The PDP-11 transmitted the participant's floor coordinates across an audio cable to the Adage. The data was transmitted asynchronously as a serial bit stream of varying pulse widths. The Adage generated the maze image which was picked up by a TV camera and transmitted via a video cable back to the Union where it was rear-screen projected to a size of 8′ × 10′.

Interaction

The first problem was simply to educate the person to the relationships between the floor and the screen. Initially, a diamond with a cross in it representing the person's position appeared on the screen. Physical movement in the room caused the symbol to move correspondingly on the screen. As the participant approached the screen, the symbol moved up. As he moved away, it moved down. The next step was to induce the person to move to the starting point of the maze, which had not yet appeared on the screen (Figure 7). To this end, another object was placed on the screen at the position which would be the starting point of the maze. The viewer unavoidably wondered what would happen if he walked his symbol to the object. The arrival of his symbol at the starting point caused the object to vanish and the maze to appear. Thus confronted with the maze, no one questioned the inevitability of walking it.

Software boundaries

Since there was no physical constraints in the gallery, the boundaries of the maze had to be enforced by the computer. Each attempt to violate a boundary was foiled by one of many responses in the computer's repertoire. The computer could move the line, stretch it elastically, or move the whole maze. The line could disappear, seemingly removing the barrier, except that the rest of the maze would change

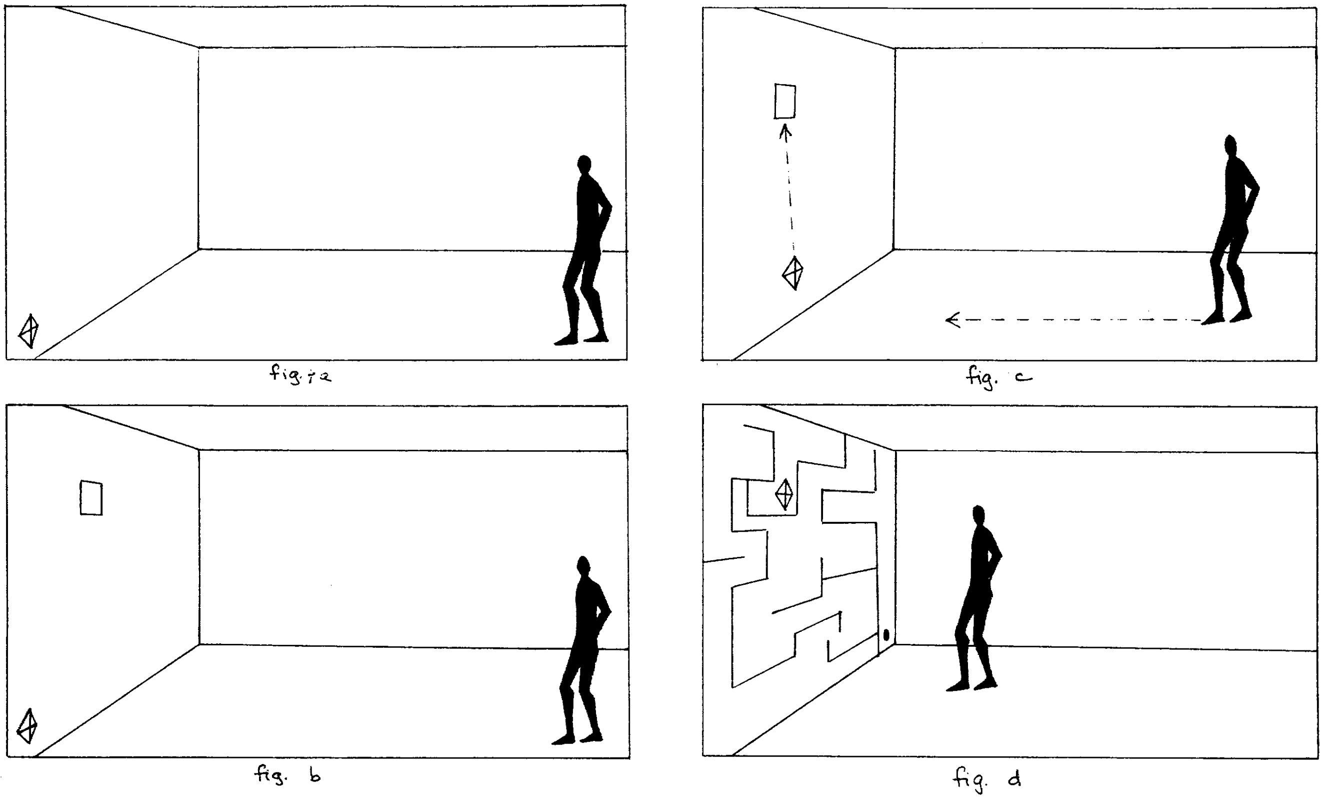


Figure 7—Composed environment-Maze

simultaneously so no advantage was gained. In addition, the symbol representing the person could split in half at the violated boundary, with one half held stationary while the other half, the alter ego, continued to track movement. However, no progress could be made until the halves of the symbol were reunited at the violated boundary.

Even when the participant was moving legally, there were changes in the program contingent upon his position. Several times, as the goal was approached, the maze changed to thwart immediate success. Or, the relationship between the floor and the maze was altered so that movements that once resulted in vertical motion, now resulted in horizontal motion. Alternatively, the symbol representing the participant could remain stationary while the maze moved.

Ultimately, success was not allowed. When reaching the goal seemed imminent, additional boundaries appeared in front of and behind the symbol, boxing it in. At this point, the maze slowly shrank to nothing. While the goal could not be reached, the composed frustration made the route interesting.

Experience

The maze experience conveyed a unique set of feelings. The video display space created a sense of detachment enhanced by the displaced feedback; movement on the horizontal plane of the floor translated onto the vertical plane of the screen. The popular stereotype of dehumanizing technology seemed fulfilled. However, the maze idea was engaging and people became involved willingly. The lack of any other sensation focused attention completely on this interaction. As the experience progressed, their perception of the maze changed. From the initial impression that it was a problem to solve, they moved to the realization that the maze was a vehicle for whimsy, playing with the concept of a maze and poking fun at their compulsion to walk it.

VIDEOPLACE

For the past two years I have been working on a project called VIDEOPLACE, under the aegis of the Space Science and Engineering Center of the University of Wisconsin. This work is funded by the National Endowment for the Arts and the Wisconsin Arts Board. A preliminary version was exhibited at the Milwaukee Art Center for six weeks beginning in October 1975. The development of VIDEOPLACE is still under way and several more years will be required before its potential is fully realized both in terms of implementing the enabling hardware and exploring its compositional possibilities.

VIDEOPLACE is a conceptual environment with no physical existence. It unites people in separate locations in a common visual experience, allowing them to interact in unexpected ways through the video medium. The term VIDEOPLACE is based on the premise that the act of communication creates a place that consists of all the information that the participants share at that moment. When people are in the same room, the physical and communication places are the same. When the communicants are separated by distance, as in a telephone conversation, there is still a sense of being together although sight and touch are not possible. By using television instead of telephone, VIDEOPLACE seeks to augment this sense of place by including vision, physical dimension and a new interpretation of touch.

VIDEOPLACE consists of two or more identical environments which can be adjacent or hundreds of miles apart. In each environment , a single person walks into a darkened room where he finds himself confronted by an 8′ × 10′ rear-view projection screen. On the screen he sees his own life-size image and the image of one or more other people. This is surprising in itself, since he is alone in the room (Figure 8). The other images are of people in the other environments. They see the same composite image on their screens. The visual effect is of several people in the same room. By moving around their respective rooms, thus moving their images, the participants can interact within the limitations of the video medium.

It is these apparent limitations that I am currently working to overcome. When people are physically together, they can talk, move around the same space, manipulate the same objects and touch each other. All of these actions would appear to be impossible within the VIDEOPLACE. However, the opposite is true. The video medium has the potential of being more rich and variable in some ways, than reality itself.

It would be easy to allow the participants to talk, although I usually preclude this, to force people to focus on the less familiar kinds of interaction that the video medium provides. A sense of dimension can be created with the help of computer graphics, which can define a room or another spatial context within which the participants appear to move around. Graphics can also furnish this space with artificial objects and inhabit it with imaginary organisms. The sense of touch would seem to be impossible to duplicate. However, since the cameras see each person's image in contrast to a neutral background, it is easy to digitize the outline and to determine its orientation on the screen (Figure 9). It is also easy to tell if one person's image touches another's, or if someone touches a computer graphic object. Given this information the computer can make the sense of touch effective. It can currently respond with sounds when two images touch and will ultimately allow a person's image to pick up a graphic object and move it about the screen.

While the participants' bodies are bound by physical laws such as gravity, their images could be moved around the screen, shrunk, rotated, colorized and keyed together in arbitrary ways. Thus, the full power of video processing

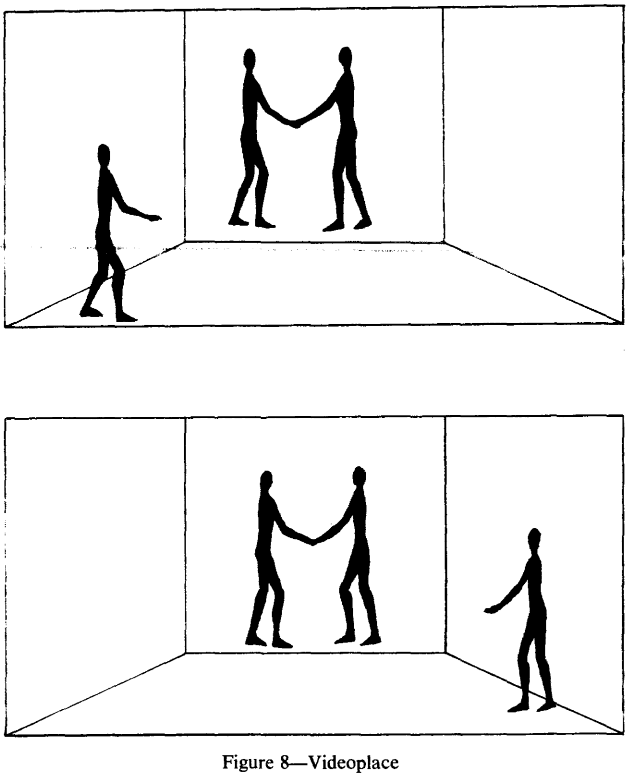


Figure 8—Videoplace

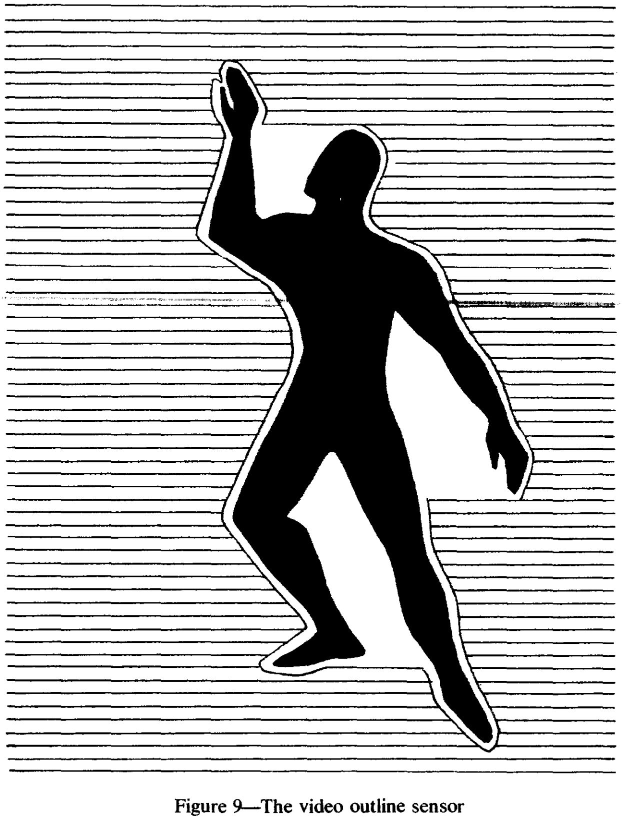


Figure 9—The video outline sensor

could be used to mediate the interaction and the usual laws of cause and effect replaced with alternatives composed by the artist.

The impact of the experience will derive from the fact that each person has a very proprietary feeling towards his own image. What happens to his image happens to him. In fact, when one person's image overlaps another's, there is a psychological sensation akin to touch. In VIDEOPLACE, this sensation can be enhanced in a number of ways. One image can occlude the other. Both images can disappear where they intersect. Both images can disappear except where they intersect. The intersection of two images can be used to form a window into another scene so two participants have to cooperate to see a third.

VIDEOPLACE need not involve more than one participant. It is quite possible to create a compelling experience for one person by projecting him into this imaginary domain alone. In fact the hardware/software system underlying VIDEOPLACE is not conceived as a single work but as a general facility for exploring all the possibilities of the medium to be described next.

RESPONSE IS THE MEDIUM

The environments described suggest a new art medium based on a commitment to real-time interaction between men and machines. The medium is comprised of sensing, display and control systems. It accepts inputs from or about the participant and then outputs in a way he can recognize as corresponding to his behavior. The relationship between inputs and outputs is arbitrary and variable, allowing the artist to intervene between the participant's action and the results perceived. Thus, for example, the participant's physical movement can cause sounds or his voice can be used to navigate a computer defined visual space. It is the composition of these relationships between action and response that is important. The beauty of the visual and aural response is secondary. Response is the medium!

The distinguishing aspect of the medium is, of course, the fact that it responds to the viewer in an interesting way. In order to do this, it must know as much as possible about what the participant is doing. It cannot respond intelligently if it is unable to distinguish various kinds of behavior as they occur.

The environment might be able to respond to the participant's position, voice volume or pitch, position relative to prior position or the time elapsed since the last movement. It could also respond to every third movement, the rate of movement, posture, height, colors of clothing or time elapsed since the person entered the room. If there were several people in the room, it might respond to the distance separating them, the average of their positions or the computer's ability to resolve them, i.e., respond differently when they are very close together.

In more complex interactions like the maze, the computer can create a context within which the interaction occurs. This context is an artificial reality within which the artist has complete control of the laws of cause and effect. Thus the actions perceived by the hardware sensors are tested for significance within the current context. The computer asks if the person has crossed the boundary in the maze or has touched the image of a particular object. At a higher level the machine can learn about the individual and judge from its past experience with similar individuals just which responses would be most effective.

Currently, these systems are constrained by the total inability of the computer to make certain very useful and for the human, very simple perceptual judgments, such as whether a given individual is a man or a woman or is young or old. The perceptual system will define the limits of meaningful interaction, for the environment cannot respond to what it cannot perceive. To date the sensing systems have included pressure pads, ultrasonics and video digitizing.

As mentioned before, the actual means of output are not as important in this medium as they would be if the form were conceived as solely visual or auditory. In fact, it may be desirable that the output not qualify as beautiful in any sense, for that would distract from the central theme: the relationship established between the observer and the environment. Artists are fully capable of producing effective displays in a number of media. This fact is well known and to duplicate it produces nothing new. What is not known and remains to be tested is the validity of a responsive aesthetic.

It is necessary that the output media be capable of displaying intelligent, or at least composed reactions, so that the participant knows which of his actions provoked it and what the relationship of the response is to his action. The purpose of the displays is to communicate the relationships that the environment is trying to establish. They must be capable of great variation and fine control. The response can be expressed in light, sound mechanical movement, or through any means that can be perceived. So far computer graphics, video generators, light arrays and sound synthesizers have been used.

CONTROL AND COMPOSITION

The control system includes hardware and software control of all inputs and outputs as well as processing for decisions that are programmed by the artist. He must balance his desire for interesting relationships against the commitment to respond in real-time. The simplest responses are little more than direct feedback of the participant's behavior, allowing the environment to show off its perceptual system. But far more sophisticated results are possible. In fact, a given aggregation of hardware sensors, displays and processors can be viewed as an instrument which can be programmed by artists with differing sensitivities to create completely different experiences. The environment can be thought of in the following ways:

1. An entity which engages the participant in a dialogue. The environment expresses itself through light and sound while the participant communicates with physi-

cal motion. Since the experience is an encounter between individuals, it might legitimately include greetings, introductions and farewells—all in an abstract rather than literal way. The problem is to provide an interesting personality for the environment.

2. A personal amplifier. One individual uses the environment to enhance his ability to interact with those within it. To the participants the interaction might appear similar to that described above. The result would be limited by the speed of the artist's response but improved by his sensitivity to the participants' moods. The live drawing interaction in METAPLAY could be considered an example of this approach.
3. An environment which has sub-environments with different response relationships. This space could be inhabited by artificial organisms defined either visually or with sound. These creatures can interact with the participants as they move about the room.
4. An amplifier of physical position in a real or artificially generated space. Movements around the environment would result in much larger apparent movements in the visually represented space. A graphic display computer can be used to generate a perspective view of a modelled space as it would appear if the participant were within it. Movements in the room would result in changes in the display, so that by moving only five feet within the environment, the participant would appear to have moved fifty feet in the display. The rules of the modelled space can be totally arbitrary and physically impossible, e.g. a space where objects recede when you approach them.
5. An instrument which the participants play by moving about the space. In PSYCHIC SPACE the floor was used as a keyboard for a simple musical instrument.
6. A means of turning the participant's body into an instrument. His physical posture would be determined from a digitized video image and the orientation of the limbs would be used to control lights and sounds.
7. A game between the computer and the participant. This variation is really a far more involving extension of the pinball machine, already the most commercially successful interactive environment.
8. An experimental parable where the theme is illustrated by the things that happen to the protagonist—the participant. Viewed from this perspective, the maze in PSYCHIC SPACE becomes pregnant with meaning. It was impossible to succeed, to solve the maze. This could be a frustrating experience if one were trying to reach the goal. If, on the other hand, the participant maintained an active curiosity about how the maze would thwart him next, the experience was entertaining. Such poetic composition of experience is one of the most promising lines of development to be pursued with the environments.

IMPLICATIONS OF THE ART FORM

For the artist the environment augurs new relationships with his audience and his art. He operates at a metalevel. The participant provides the direct performance of the experience. The environmental hardware is the instrument. The computer acts much as an orchestra conductor controlling the broad relationships while the artist provides the score to which both performer and conductor are bound. This relationship may be a familiar one for the musical composer, although even he is accustomed to being able to recognize one of his pieces, no matter who is interpreting it. But the artist's responsibilities here become even broader than those of a composer who typically defines a detailed sequence of events. He is composing a sequence of possibilities, many of which will not be realized for any given participant who fails to take the particular path along which they lie.

Since the artist is not dedicated to the idea that his entire piece be experienced he can deal with contingencies. He can try different approaches, different ways of trying to elicit participation. He can take into account the differences among people. In the past, art has often been a one-shot, hit-or-miss proposition. A painting could accept any attention paid it, but could do little to maintain interest once it had started to wane. In an environment the loss of attention can be sensed as a person walks away. The medium can try to regain attention and upon failure, try again. The piece has a second strike capability. In fact it can learn to improve its performance, responding not only to the moment but also to the entire history of its experience.

In the environment, the participant is confronted with a completely new kind of experience. He is stripped of his informed expectations and forced to deal with the moment in its own terms. He is actively involved, discovering that his limbs have been given new meaning and that he can express himself in new ways. He does not simply admire the work of the artist; he shares in its creation. The experience he achieves will be unique to his movements and may go beyond the intentions of the artist or his understanding of the possibilities of the piece.

Finally, in an exciting and frightening way, the environments dramatize the extent to which we are savages in a world of our own creation. The layman has extremely little ability to define the limits of what is possible with current technology and so will accept all sorts of cues as representing relationships which in fact do not exist. The constant birth of such superstitions indicates how much we have already accomplished in mastering our natural environment and how difficult the initial discoveries must have been.

APPLICATIONS

The responsive environment is not limited to aesthetic expression. It is a potent tool with applications in many fields. VIDEOPLACE clearly generalizes the act of telecommunication. It creates a form of communication so powerful that two people might choose to meet visually, even if it were possible for them to meet physically. While it is not immediately obvious that VIDEOPLACE is the optimum means of telecommunication, it is reasonably fair to say that it provides an infinitely richer interaction than

Picturephone allows. It broadens the range of possibilities beyond current efforts at teleconferencing. Even in its fetal stage, VIDEOPLACE is far more flexible than the telephone is after one hundred years of development. At a time when the cost of transportation is increasing and fiber optics promise to reduce the cost of communication, it seems appropriate to research the act of communication in an intuitive sense as well as in the strictly scientific and problem-solving approaches that prevail today.

EDUCATION

Responsive environments have tremendous potential for education. Our entire educational system is based on the assumption that thirty children will sit still in the same room for six hours a day and learn. This phenomenon has never been observed in nature and it's the exception in the classroom where teachers are pitted against children's natural desire to be active. The responsive environments offer a learning situation in which physical activity is encouraged. It is part of the process. An environment like VIDEOPLACE has an additional advantage. It gives the child a life-size physically identical alter ego who takes part in composed learning adventures on the video screen. In a fully developed VIDEOPLACE the size and position of the child's image on the screen would be independent of actual location in the room. In an interactive Sesame Street a child would be mesmerized as his own miniaturized image was picked up by a giant Big Bird (Figure 10). Conversely he would be delighted if the scales were reversed and he were able to pick up the image of a tiny adult teacher who spoke to him from his hand. The most overworked educational cliché, "experience is the best teacher," would have new meaning in this context. The environments provide an exerience which can be composed and condensed to demonstrate an educational point.

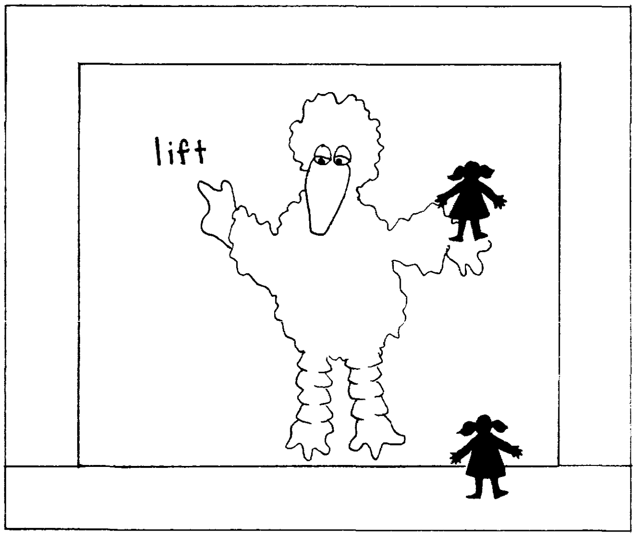


Figure 10—Interactive sesame street

While it is easy to generate examples of how the environments can be used to teach traditional subjects, their significance does not lie only in their ability to automate traditional teaching. More important, they may revolutionize what we teach as well as how we teach. Since the environments can define interesting relationships and change them in complex ways, it should be possible to create interactions which enrich the child's conceptual experience. This would provide the child with more powerful intellectual structures within which to organize the specific information he will acquire later. The goal would be to sophisticate the child, not to feed him facts.

PSYCHOLOGY

Since the environments can monitor the participants' actions and respond with visual and auditory feedback, it is natural to consider their application to the study of human behavior. The use of the computer allows an experimenter to generate patterns and rhythms of stimuli and reinforcers. In addition, the ability to deal with gross physical behavior would suggest new experimental directions. For instance, perception could be studied as part of physical behavior and not as a sedentary activity distinct from it. Also, an environment like VIDEOPLACE is very general. The same aggregate of hardware and software could be programmed to control a broad range of experiments. The scheduling of different experiments could be interspersed because only the software would have to be changed.

Since the university students used as subjects in many experiments are quite sophisticated about the concerns of psychologists, what is often being studied is the self-conscious behavior of people who know they are in an experiment and are trying to second-guess it. On the other hand, environments open to the public offer a source of spontaneous behavior. It is quite easy for the computer to take statistics without interfering with the experience. Or, interactions can be composed to test specific experimental hypotheses.

PSYCHOTHERAPY

It is also worth considering the application of responsive environments to psychotherapy. Perhaps most important for a psychotherapist is the ability of the environment to evoke and expand behavior. We have found in the past that people alone in a dark room often become very playful and flamboyant—far more so than they are in almost any other situation. Since the environment is kept dark, the patient has a sense of anonymity; he can do things that he might not do otherwise. The fact that he is alone in the dark serves to protect him both from his image of himself and from his fear of other people. The darkness also is a form of sensory deprivation which might prevent a patient from withdrawing. If he is to receive any stimulation at all, it must be from acting within the environment. Once he acts, he can be reinforced for continuing to act.

In the event that the subject refuses to act, the environment can focus on motions so small as to be unavoidable and respond to these and as time goes by encourage them, slowly expanding them into larger behavior, ultimately leading the patient to extreme or cathartic action.

In certain situations the therapist essentially programs himself to become mechanical and predictable, providing a structure that the patient can accept which can be expanded slowly beyond the original contract. It is possible that it would be easier to get a patient to trust a mechanical environment and completely mechanized therapy. Once the patient was acting and trusting within the environment, it would be possible to slowly phase in some elements of change, to generalize his confidence. As time went by, human images and finally human beings might be added. At this point, the patient could venture from his responsive womb, returning to it as often as needed.

CONCLUSION

The responsive environment has been presented as the basis for a new aesthetic medium based on real-time interaction between men and machines. In the long range it augurs a new realm of human experience, artificial realities which seek not to simulate the physical world but to define arbitrary, abstract and otherwise impossible relationships between action and result. In addition, it has been suggested that the concepts and tools of the responsive environments can be fruitfully applied in a number of fields.

What perhaps has been obscured is that these concepts are the result of a personal need to understand and express the essence of the computer in humanistic terms. An earlier project to teach people how to use the computer was abandoned in favor of exhibits which taught people about the computer by letting them experience it. METAPLAY, PSYCHIC SPACE and VIDEOPLACE were designed to communicate an affirmative vision of technology to the lay public. This level of education is important, for our culture cannot continue if a large proportion of our population is hostile to the tools that define it.

We are incredibly attuned to the idea that the sole purpose of our technology is to solve problems. It also creates concepts and philosophy. We must more fully explore these aspects of our inventions, because the next generation of technology will speak to us, understand us, and perceive our behavior. It will enter every home and office and intercede between us and much of the information and experience we receive. The design of such intimate technology is an aesthetic issue as much as an engineering one. We must recognize this if we are to understand and choose what we become as a result of what we have made.

Computer technology in data-base publishing

by D. B. BAKER and R. E. O'DETTE

Chemical Abstracts Service
Columbus, Ohio

ABSTRACT

The experience and the expectations of a large scientific society publisher of chemical and chemical engineering information tools provide a broad context for considering the history and probable future impact of computer technology on information dissemination. Chemical Abstracts Service (CAS) was in good health for half a century before computer technology was available in practical application to the production of information services. It is now in satisfactory technical and economic health largely because computer technology is available.

Early applications of computers to practical word and chemical structure manipulation were for the most part probes to determine how, or even if, the machines could be useful. Today the computer is the heart of the CAS production system in the same sense as the reactors are the heart of a chemical plant. Regardless of the forms or media of the output, the computer is essential to their production.

The future sees computers as the means by which all scientific and technical information dissemination activities can be helped to coordinate their functions and improve their services to information users.

INTRODUCTION

In considering "the impact of computer technology on information dissemination," the information-accessing services such as Chemical Abstracts Service (CAS) can be pardoned for wondering where the question-mark is. To us, this impact is like the impact of a cooling breeze upon a sweaty brow: in part, we did not realize how hot and sweaty we were until the breeze struck; in part we know that we need the breeze to continue our work.

Chemical Abstracts (CA) was, or course, in very good health for half a century before computer technology was available in practical application. It is now in good health—technically and economically—*because* computer technology is available. The organization and the services it provides could not exist without computer technology. And many other organizations share our experience.

Long gone are the days when we wondered about "the role" of "the computer" in our business. Gone also are the days when we seriously asked ourselves questions about whether "the computer" would, or could improve upon manual processes without degrading the quality of our services.

We no longer think about "the computer." We do accept "computer technology," and we understand this to mean "machines supporting people," not "people adapting to machines." At this point in time, computers are effectively cooling our sweaty brows—we could not exist without them. We don't intend to try.

INFORMATION-ACCESSING SERVICES

In discussing computer technology as applied to information dissemination, this paper emphasizes, almost exclusively, the use of computers in data base manufacture. The bodies of information, large and small, from which dissemination takes place are routinely called "data bases." And it is as a data base manufacturer that CAS is best able to speak.

Figure 1 is a drawing that CAS has used for ten years. It is a useful aid in the discussion of an important fundamental concept in information transfer. The figure shows the information user as the center of a universe, not only of different services but of different kinds of services. The information user needs different kinds of information support at different times; sometimes he needs different kinds in combination at the same time.

As we survey what has been happening in information dissemination through the years, we recognize that while the names and the forms may change, the functions depicted in Figure 1 have not changed very much, and they are unlikely to change conceptually for some time to come.

To begin at the beginning, there must be some means for capturing and communicating the first disclosure of new information. At one time, letters between scientists served as those means. Later, more formal means—the primary journals—became pre-eminent. Today, the primary journals of the world are still pre-eminent as the basic means for the first disclosure of new information. The traditional primary journal is being augmented by some new forms and media, but regardless of the form or medium, serious information dissemination will continue to begin with the creation of a record of disclosure. That record, whether or not it is a

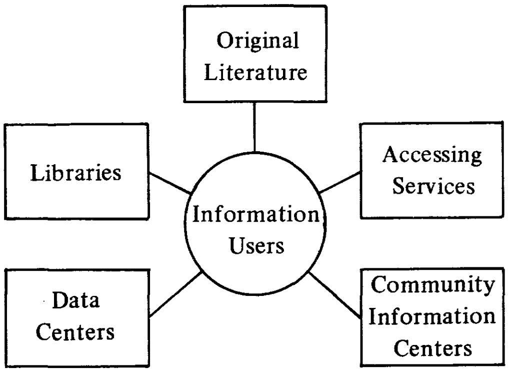


Figure 1

primary journal article, becomes, when it is made public, the "primary" source of the information.

Because even the people who are most interested cannot expect to collect, or even to be aware of, all pertinent recorded original information, past as well as current, they must depend on two other functions of information dissemination. They need libraries to keep large, organized collections of recorded information. They need accessing services such as CAS to help them gain an impression of the contents of those collections and what is being added to them. Libraries and accessing services existed for many years without thought of computers, but today both are becoming more and more heavily involved with automated retrieval.

The remaining two components of the information universe shown in Figure 1 are less firmly established than the first three, but, in one form or another, will assume major roles in the future. This transition will come about because of the need for the services and because computer systems will make them feasible and effective. The *community information center* is specifically a computer-based information center. It may be on-line or batch-oriented, commercial, academic, or not-for-profit, but its function is to serve an inquiring public by manipulating a collection of computer-readable files obtained from a variety of producers. The *data center* shown sharing equal status with the other functions does not truly occupy that position at present, but will undoubtedly do so in the future. Computers will play a major role in that transition, owing to the tremendous power of computational technology to organize data.

Computer technology is being introduced into each component of the user's information universe, but a breakthrough is needed to achieve closer coordination and interlinkage among the components of the universe.

EVOLUTION OF USE OF COMPUTER TECHNOLOGY AT CAS

With these concepts of an information universe as a starting point, it is appropriate to trace the evolution of the use of computer technology at CAS to help explain the present and clarify a perspective on the future.

Figure 2 depicts the logical arrangement of the CAS computer system today.[1] That system consists of foundation systems which support data entry and validation, data base management, and basic output functions. Separate production sub-systems are responsible for determining the subset of information to be selected from the data base for inclusion in a particular output package, for specifying the medium and format in which it is to be produced, and for driving the appropriate output devices.

THE ONSET OF MECHANIZATION

As noted, Chemical Abstracts had been published for more than 50 years before the first computer started working for the organization. As an abstracting and indexing service first in chemistry and later also in chemical engineering as that newer technology took shape, we were modeled after our German forebears who started such a service in the 1830's. Even as late as 1960 mechanization was limited. Our scientific staff used pencils, pens, and paper, and our clerical staff used typewriters to translate the handwriting. In the mid-1950's our indexers had begun to dictate their entries into magnetic wire recorders as a step forward from handwriting. Even though our typists had to learn to transcribe the complex dictation, this early application of modern information processing technology was found to be cost-effective.

In 1961, CAS made the first production scale application of Hans Peter Luhn's keyword-in-context index[2] programs by producing a periodical, Chemical Titles, completely from punched card input and computer processing. Four years later, in 1965, CAS produced a specialized abstracting publication, with indexes, called Chemical-Biological Activities (CBAC). That publication embodied a number of significant advances. Abstracts were input to an IBM 1410. Several kinds of indexes were derived by a combination of machine and manual means, and the journal was composed in a highly formatted output using a 120-character print chain created for this purpose. It not only had a much more attractive and readable typeface than the common 48-character chain, but it produced lower and uppercase

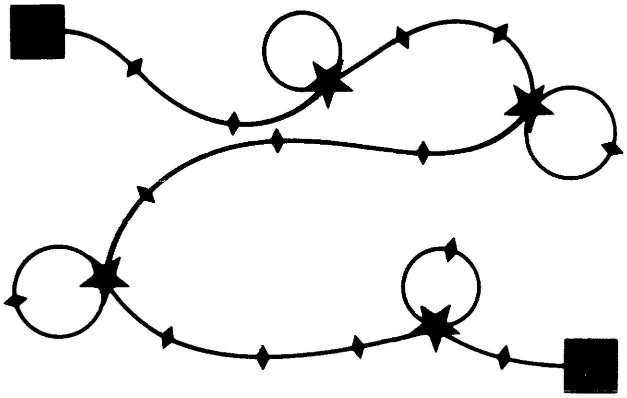


Figure 2

Roman and Greek letters, exponents, and other typography that was, at that time, sophisticated for computers. In fact, the problem of typography has been a key to many practical problems over the years.

In parallel with the CBAC development, the beginnings of the CAS Chemical Registry were established. The Registry concept was not generally understood at the time, although it did attract attention because the President's Science Advisor personally announced coordinated federal funding of the initial test of the Registry concept with support by the National Science Foundation (NSF), the National Institutes of Health, and the Department of Defense.[3] The Registry System is a pre-eminent example of the power of computer application in information processing.

These beginnings must be stressed in order to underscore how recently it was that such simple matters were looked on as being difficult. A flow diagram of CAS at that time would have shown many parallel, simultaneous production streams. Each stream involved the selection of source documents appropriate to the files being built. A computer was used in at least three of the production streams, for CBAC, Chemical Titles, and Polymer Science & Technology, a service analogous to CBAC, but in another scientific field.

In 1966, there was a breakthrough. The IBM 2280 Film Recorder was announced, and we negotiated with IBM to make some special modifications on a 2280 for character generation. That modified 2280 was the first fully operational, production-line photocomposer that had all the necessary attributes to bring photocomposition to fruition in helping solve the problems of computer production of publications such as ours. Our modified 2280 was computer driven; it was very fast. Individual characters and full-column formatting were program-generated and output speedily by the device, and the visual product was of satisfactory graphic-arts quality. In summary, we had a high-speed photocomposer which produced output that was visually comparable to the hot-type composition we had been doing.

One important technique that the 2280 enabled us to use was the output of the same information in two radically different forms—a proof sheet produced in high-speed, low quality output for internal proofreading, and a final, high quality output from which to prepare printed information services.

The reason that the 2280 must be stressed in our computer-related history is that its capabilities were the catalyst that enabled us to conceptualize a complete redesign of our procedures and really begin to experience profoundly the impact of computer processing on our kind of information dissemination. It was then that we could begin to visualize the reality of data base publishing.

A NEW MANUFACTURING SYSTEM

Prior to the use of computer technology and even prior to the photocomposer of 1966, the CAS manufacturing system depended on stepwise improvement in the analysis of each source document intended for Chemical Abstracts, our principal output. An abstract was prepared, transcribed, proofed, *recopied,* edited, *recopied,* proofed, *recopied,* and sent to the printer. While galleys were proofed, indexing entries were prepared, transcribed, edited, assembled, edited, and so on. Other similar procedures were used for recording bibliographic information, and still other procedures for producing five-year collective indexes by reassembling, and re-editing the entries created for the semiannual indexes. A flow diagram for the creation of pre-1966 issues of CA and their corresponding indexes would look like Figure 3. There would be many steps, many loops for proofing and copying, and hardly any interlinks.

The present CAS production system looks like Figure 4. Instead of the dozens of sequential steps of the "traditional" system, we have large blocks of integrated procedures. The intellectual processes that were necessarily distributed throughout the manual system are now telescoped at the beginning of the computer-based system. Instead of abstractors, abstract editors, indexers, and index editors, we now have document analysts. This change is much more than semantic. Over a period of several years, it was necessary to gradually retrain the entire editorial staff of subject matter and language specialists so that they could do both abstracting and indexing. As the training of small subject-related groups was completed, part of CAS coverage was converted to the newly integrated analysis procedure.

The new system subdivides the bibliographic identifications and analysis of incoming source documents into a large number of relatively small data elements. This accomplishes two purposes. Automated error detection becomes easier to design and program, and the system—in both its people and its machine functions—is freed of the former restraints of dealing with large, complex blocks of associated information.

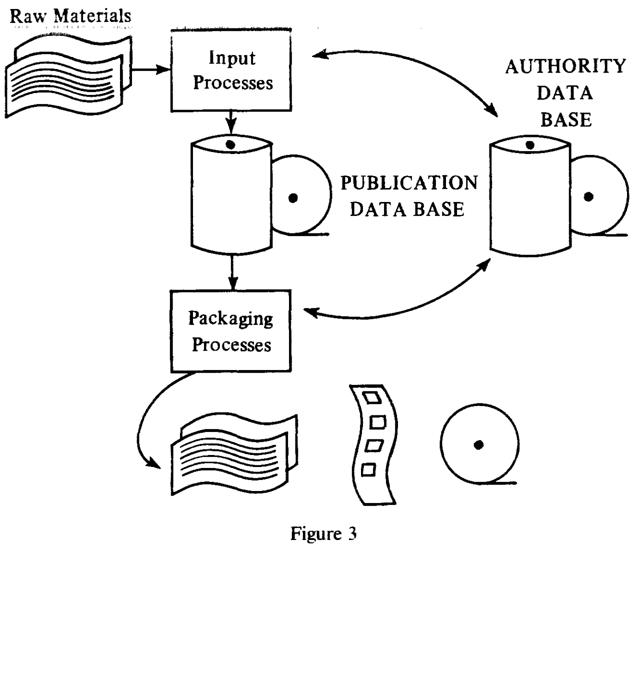


Figure 3

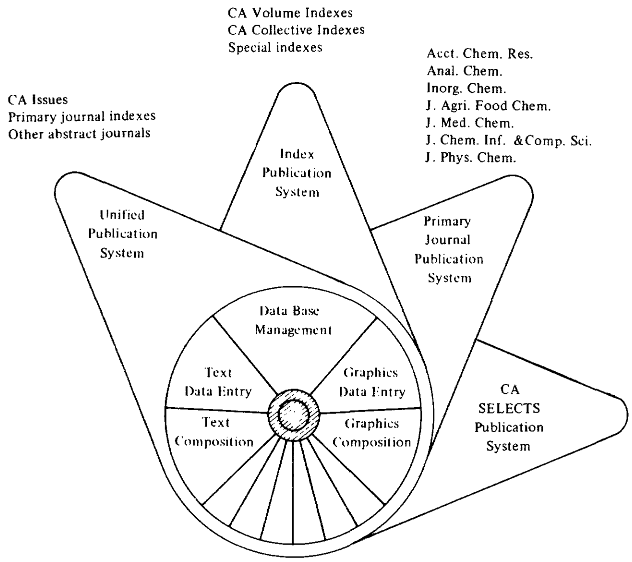


Figure 4

TAILORED INFORMATION SERVICES

Our user audience has always wanted to receive, and we have always wanted to be able to produce, whatever cut of the data base was needed. Now that all of the CAS data is processed through the computer system of Figure 4, we can think freely about non-traditional combinations of bits of subject matter, aspects of index entries, chemical structures, trade names, chemical names, and so on, into services precisely defined for a specific user audience. To illustrate, CA is divided into 80 sections representing meaningful subdivisions of chemically-related science and technology. The 80 sections are a basic, but by no means the only possible way to structure the science and technology of chemistry and chemical engineering. Towards the end of last year, for the first time we were able to take advantage of our completely computerized processing system to produce the first of six topic-oriented alerting services which we expect to be followed by many more. The six new printed services contain abstracts that are the same as abstracts that appear in CA, but the set is selected individually from the data base. This selection can be precise: in the specialized field of high-speed liquid chromotography, 18 abstracts were selected from nine CA sections in a recent issue.

To achieve the freedom of output options that computer processing has brought requires that details of format and typography not be imposed on the information until the output stage. That is, data elements comprising bibliographic information, abstracts, and index entries are input free of typographic or format coding. Details of typography and format are resident in output programs only. Similarly, output programs to reproduce computer-readable files can select data elements, ignoring questions of typography and printing format, and simply arrange the data elements in the Standard Distribution Format in which all CAS computer-readable files are distributed.

INFORMATION CENTERS

There are 90 computer-based information centers licensed to provide services to the public from various CAS computer-readable files. Most of these centers also process files from a number of data base producers. In the case of Lockheed and Systems Development Corporation, for example, the number of different data bases offered is large.

It was not long ago that the prospect of interacting with a computerized file of information at a terminal was considered to be a dream. But there are now discussions on the cost effectiveness of on-line search in a company library environment. The Chemline-Toxline services of the National Library of Medicine (NLM) are a major example of integration of information from multiple sources. Computer-readable input from BioSciences Information Services (BIOSIS), CAS, and several other sources is converted by NLM into one coherent multidisciplinary file for on-line search. Improved compatibility among data base suppliers would surely simplify the development of future systems in multidisciplinary areas.

USE OF AUTHORITY FILES

Figure 4 shows a two-part data base. One part, the Publication Data Base, contains information directly related to individual source documents, while the other, the Authority Data Base, contains lists of acceptable index terms, chemical names, synonyms, structural formulas, standardized journal title abbreviations, and so on. Information people call such files "authority" files because they represent what the processor allows himself to do in various aspects of producing his services. An integrated computer system such as CAS's makes possible the prompt updating of such authority files.

Users might profit from access to these files which they could use to support their searching activities. CAS has produced a number of search aids to support users of its biweekly computer-readable subject index service, CA Subject Index Alert. These are derivatives of some CAS authority files. It would not be possible to produce them without the help of both the authority file content and computer technology. To illustrate, one aid is a list of all main subject headings for the CA General Subject Index, including the taxonomic headings we use. Another aid links CAS Registry Numbers for more than 3,000 frequently indexed chemical substances to the many names commonly used for those substances. Still another aid is a Key-Letter-In-Context Index of CA Nomenclature Terms, which is a listing of words used in the highly systematic CA Index Names, fragmented, and listed at each letter. That list, combined with a word frequency list of the index words, is helpful in developing search strategies using chemical name fragments.

Two of the largest and most active of the CAS authority files are the chemical Nomenclature File and the chemical Structure File of the CAS Registry System. At the end of 1976, the Nomenclature File had 5,800,000 different names and the Structure File had 3,710,000 different structures in digitized form. In a typical recent year, the files grew by about 350,000 new structures and about 500,000 names. A number of excerpts of both files have been published: parts are in Chemline-Toxline at NLM, parts are in the Lockheed on-line offerings, and other proprietors of on-line services are making plans to incorporate portions. Several chemical organizations in the US and abroad are experimenting with the files.

DATA IDENTIFICATION AND DISSEMINATION

It was noted earlier that data centers had to grow in importance and that computer technology was essential to that growth. One of the major needs of information users is for numerical and factual information with some indication of the reliability of the data. The data center has the responsibility of providing evaluation, but the original information and accessing service producers can help by clearly calling attention to the existence of data in documents so that the data centers need not recomb the world's publications looking for the desired numbers.

CAS staff, with support from interested agencies in the community, have been experimenting with data tags. These are pointers to be added to the computer-readable files to mark documents in which analysts have found data to be reported. In 1976, CAS went through some 380,000 source documents. There is a tremendous amount of data in these documents, and CAS is working to determine how much and how to handle the tagging of it.

PRIMARY JOURNAL PUBLICATION

As noted above, the deep-seated changes in the way CAS does its business have created, among other things, an entirely new approach to composing publications. The American Chemical Society (ACS), of which CAS is the largest operating division, is also the publisher of some 20 scientific and technical periodicals. Over the last several years the ACS has been developing similarly advanced composition systems for these primary publications. A key to this system was an analysis of the procedures humans use to lay out journal pages. This analysis enabled staff of CAS's Research and Development Division to devise computer programs that accomplish page layout to virtually the same standards.[4] ACS is now gradually converting publication of its journals to this system, using techniques that permit each journal to retain its own personality with respect to typography, placement of various elements of an article, etc. The same composition technology is being examined for application in the Editorial Processing Center (EPC) concept that NSF has been espousing through support of studies and experiments.[5] This application of shared computer technology has the potential to become one of the most important developments for the near- and medium-range future of information dissemination. Like many important concepts, the basic idea is simple and persuasive, but it leads to many design and operational challenges. In essence, a publications production system more-or-less like that of CAS could be used remotely by publishers of small journals. The investment required to develop and operate such a system could not be justified for an individual publisher's use alone.

Ultimately, according to one version of the EPC concept advanced by Aspen Systems in an NSF-funded study,[6] authors, editors, and reviewers for each of the primary journals using the center would have on-line access to a central facility which would provide the whole gamut of services needed in publishing the journals, including page proofs, final composed pages, by-product computer-readable files, indexes, directories, membership and subscription records, labels, a variety of form letters, and financial accounting.

Achievement of such a configuration will require much hard work and some systems development but no technical breakthroughs. The most difficult problems to solve will be to establish effective interfaces between the people and the system. A successful EPC must preserve the primary journal editor's traditional independence of choice; the EPC must be a processor, not a publisher. At the same time, if he is to take maximum advantage of EPC support, the journal editor must objectively differentiate between editorial stature and window dressing. There is no doubt that the advanced configuration postulated by Aspen would permit journal editors considerable latitude in typography and format, as does the present ACS primary journal system operated by CAS.

INTERLINKING PRIMARY AND SECONDARY PROCESSING

There are great possibilities for improving the overall efficiency and effectiveness of information dissemination by increasing operational coordination between the primary and the secondary information communities. Computer-based information technology is an important key to progress in this area. CAS staff have stated elsewhere, "There should be one total publication system for scientific documents. The system must have a number of components, with great variety of character and freedom of intellectual action, and the responsibility for its operation and management must be decentralized, but the system must be conceived and made to function as a single entity."[7] Such a sweeping statement can be seen as realistic and practical, only if one understands the necessity of the present and potential impact of computers on primary-secondary interlinking.

This is a well-advanced example of cooperation between the primary-journal and the accessing-service activities of the ACS that helps to highlight this important contribution of computer technology to information dissemination. Another is the provision of CAS Registry Numbers for substances identified in primary journal manuscripts.

The CAS Registry System, which was previously mentioned, is an automated vocabulary control system for indexing chemical substances based on the details of their molecular structure. The CAS Registry is inconceivable without computer technology, and it is an essential component of CAS processing. Beginning ten years ago, CAS also began to use the Registry in behalf of the ACS *Journal of Organic Chemistry* to provide CAS Registry Numbers to the journal at its manuscript stage so that the journal could include the numbers in its issues. Registry Numbers are also provided in a similar way to the journal, *Inorganic Chemistry,* and the German journal, *Angewandte Chemie.* CAS Registry Numbers are used in CA abstracts and indexes, in a number of non-ACS handbooks, in publications of BIOSIS, in the TOXLINE and CHEMLINE online services of NLM, in various other government files, and so on, and the use of Registry Numbers is steadily spreading. Thus, including Registry Numbers in the primary literature establishes the identification of the substance from earliest disclosure and provides a link to other references to the substance.

SHARED INDEXING

In addition to the complex matter of chemical substance identification, CAS and 14 ACS primary journals coordinate their indexing of document subject content. CAS began over five years ago to prepare annual indexes for these journals by extracting from the CAS publications data base for each journal, the keyword index terms prepared for the abstracts of papers from that journal. CAS also provides this indexing support for two journals published by Verlag Chemie in Germany, one journal of The Chemical Society in London, and one commercially published US journal.

COORDINATED EFFORT

BIOSIS and CAS have been working together for some time in seeking improved cooperation and coordination. A part of that effort has had NSF financial support. Progress has been made and there is room for much more. In the area of computer technology alone, several accomplishments are worth noting. BIOSIS has modified its production stream and CAS its output system to enable CAS to photocompose the issues of Biological Abstracts and two other specialized abstract publications of BIOSIS. BIOSIS in Philadelphia prepares the output of its editorial processing in computer-readable form compatible with CAS photocomposition procedures and ships the tapes to Columbus. CAS returns fully composed pages for the BIOSIS printer.

CAS also supplies Registry Numbers to BIOSIS for their special service, *Health Effects of Environmental Pollutants* (HEEP). In 1975, CAS prepared for BIOSIS a Chemical Index Guide to HEEP by compiling all of the synonyms that pertained to chemical substances mentioned in HEEP. This index is valuable in relating trade names and chemical names, and because the index includes CAS Registry Numbers and CA index nomenclature, the users of HEEP are helped in utilizing the CAS data base if they need further information on chemical aspects of substances.

More recently, BIOSIS, CAS, and Engineering Index have been jointly studying questions of overlap and have been evolving a concept which sets forth a collective view of a "total" information system that the three services might be able to help achieve. Working within the three organizations could improve individual and joint efficiencies while providing more effective service to users.

THE FUTURE

Computer technologists have been criticized in the past for painting a too optimistic picture of the future. In many cases, such pictures were perhaps technologically sound, but failed to recognize economic and institutional barriers to progress. So the final section of this paper will describe a vision of what *can* be achieved without asserting that this vision is necessarily a *prediction*.

Our future system will be a network of networks, whose workings will be transparent to the user who, in the act of searching from a console, will be supported by software and hardware that automatically switches among networks as interim search results and strategy modifications demand.

We envision EPC's as major factors in the capture of original information, a set of activities now usually named in a more limiting way as primary publishing. A functioning network of such centers could expedite authors' finding the right outlet for new information that warrants disclosure and, possibly, could be reservoirs for the future electronic or microform equivalent of today's text and graphical presentation of information. (In fact, such centers might store, for search or retrieval, computer-readable copies of an author's original data, gathered by machine. There will be EPC's for accessing services, as well, putting at the disposal of this part of the information community the same economies of high technology and scale as are expected of the primary EPC. We foresee the two EPC networks interconnected to coordinate coverage; to share hardware, software, and information files; and to seek, where feasible, multiple use within the community of sharable resources.

There will be new and closer relations between libraries and accessing services where there is at present a lack of directed coordination among their related activities. The acquisitions programs of libraries and the selection policies of accessing services appear to be closely related only in the minds of users, rather than in the actions of libraries and accessing services. While the basic functions of these services are different, they could be brought much more closely together for the benefit of the user by the computer technology that each community is employing to an ever increasing extent.

The computer-based link between accessing services and computer-based information centers is already a strong one.

There is a need for improved compatibility in computer-readable information transfer formats among data-base publishers and in data-element form and content. It also seems inescapable that computer technology will be one of the major forces to bring information dissemination centers and libraries into functionally closer relations. And there will be new links developed between accessing services and data centers.

Thirty years ago, when digital computers were quite new, this concept would have verged on science fiction, albeit serious, almost believable science fiction. Ten to fifteen years ago, a Delphi study among the frontiersmen of information and computer science might have produced predictions like these. Yet today, at least three conservative, traditional organizations are convinced that they can make such a vision become reality.

REFERENCES

1. Farmer, Nick A., Carole A. Schermer and Roland L. Wigington, "The American Chemical Society Composition System," *CAS Report*, 5, 3, 1976.
2. Luhn, Hans Peter, "Keyword-in-Context Index for Technical Literature (KWIC) Index)," *American Documentation*, XI, 4, 1960, p. 288.
3. CAS Staff, "CAS to Study Registry of Chemical Compounds," *Chemical & Engineering News*, June 7, 1965, p. 23.
4. Bammel, S. E., "Automatic Full-Page Formatting of Technical Primary Journals," *Proceedings of the National Computer Conference*, Anaheim, California, 1975.
5. Rhodes, S. N. and H. L. Bamford, "Editorial Processing Centers: A Progress Report," *The American Sociologist*, 11, 3, 1976, pp. 153-9.
6. Editorial Processing Centers, *Feasibility and Promise*, Aspen Systems Corporation, Rockville, Maryland, 1975, 70 pp.
7. Tate, F. A. and R. E. O'Dette, Interlinking of Primary Publications and Secondary Information Services, XXIVth International Congress of Pure and Applied Chemistry, 7, 15, 1974.

Improving corporate information services in an automated word-processing network

by HENRY L. MAYFIELD
Shell Oil Company
Houston, Texas

ABSTRACT

The flow of corporate information is destined to change considerably with the arrival of word-processing technology and the expansion of computer communication networks. Computer programs will extract indexing information from documents prepared in word-processing clusters and pass the data to a central computer complex for filing. By querying that database, employees will gain access to documentation corporate-wide. This paper follows the steps in a document's life cycle. A methodology is examined by which corporate information service departments may plan for, and ultimately provide, the benefits of enhanced information access.

INTRODUCTION

Attention has recently focused on the reaction of corporate management and office support staff to revolutionary change brought with office automation. Many organizations, through distributed word-processing centers, document improved clerical productivity.[1] Several experiments in electronic mail cite speedier information flow as having a positive impact on high-level decision-making.[2] Largely ignored, until now, is the methodology by which the corporate information service department (CISD) will enable almost all corporate employees to derive immediate benefit from the new technology.

The potential beneficiary of improved information services is anyone who comes in contact with textual documents. The known benefits of word-processing are seen in the composition cycle of letters and reports. With the use of these services, corporations are better equipped to produce paper in what has been termed the "technical information explosion era." Present world-wide information growth rates of 12-13 percent are predicted to be four to seven times as large in the mid-1980's. Employees flooded by unstructured information will find they are ill-equipped for minor job crises unless new mechanisms are devised to aid them in the document reaction cycle. Standards and services, developed by CISDs, could act as one such positive mechanism.

Ultimately, employees will have access to computer reports indexing documents authored by themselves or their co-workers. (This paper does not explore information access to extra-corporate literature.)

Document retrieval from a locally available micrographics archive will then take only a few minutes. Supervisors, higher management, and other authorized personnel will dial into a central computer database and retrieve any (externally transmitted) documents by typing in keywords describing their content. Some documents will be transmitted via electronics. Others which formerly were reproduced in great quantity prior to shipping will arrive in an envelope on microform and be printed at the receiver's site. When appropriate, information specialists will provide personalized assistance to the information-seeker.

Before looking at how all of this may be achieved, a review of present systems—modified by word-processing/computer network technology—is in order.

INFORMATION FLOW IN THE NETWORK

The components of an automated word-processing network, including communications linking terminals and/or computers at the network nodes, have passed through several phases:

Phase 1—Communications link remote-job-entry or time-sharing terminals to a physically distant computer complex.

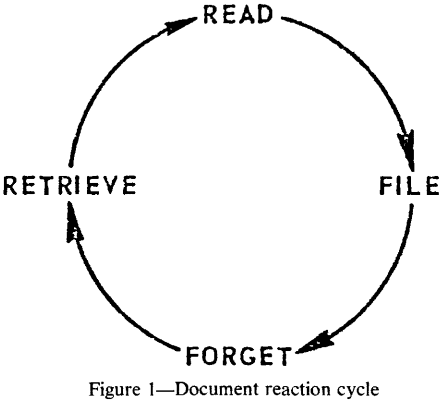


Figure 1—Document reaction cycle

Automatic text-editing systems permit operators to key-enter and edit documents using low-speed typewriter/CRT devices. Documents are filed on disk storage at the central complex.

Phase 2—Installation of "intelligent" terminals at the satellite nodes permit documents to be stored locally on cassettes or diskettes.

Phase 3—By replacing intelligent terminals with multi-terminal minicomputers, the enlarged network serves a greater number of operators. These low-cost shared logic systems placed within a local word-processing cluster replace the function of the distant central system of Phase 1.

Phase 3 networks can be utilized in several ways. Most user composition activity takes place at the network's satellite nodes. Documents are entered, edited, and printed within the cluster. When the network is used for distribution, a document may be transmitted electronically via communication links to other network nodes for printing at the addressee's cluster. When it is used as an electronic message center, the addressees are notified at their on-line terminals of the newly-arrived document. They may check the sender, date, and subject of the documents before printing or destroying selected ones. Replies may be composed and distributed electronically.

A method for document filing, storage, and retrieval to aid the "reaction cycle" spoken of earlier is being evaluated by researchers at the Massachusetts Institute of Technology.[3] Their DMS Message System takes document messages and transmits them to other points at a satellite node or to other computers attached to the ARPANET communications network. The ARPANET, begun in the 1960's, is an experimental network linking over 100 private and public computer/research installations from Hawaii to Europe. Users are able to lower their computer operating expense by sharing computer resources, e.g., central processing units, mass storage devices, plotters, and microform systems.

In preparing the documents for filing, messages are broken down into fields, such as subject, date, text, action-to, carbon-copy, etc. The assigned field-values are used to index and store/retrieve documents to/from a database. How DMS would economically interface with a variety of vendors' computer hardware/software in a corporate network is not clear at this time. Conceptually, features of DMS are certainly among those that could enhance corporate information services.

ENHANCING SERVICES IN THE NETWORK

There are two immediate goals that CISDs should seek in a Phase 3 network:

1. Specification and enforcement of corporate documentation standards. When satellite word processing clusters use common report-numbering, author/name, and date-entry formats (among other documentation standards), the problem of computer indexing becomes a manageable function.
2. Word-processing clusters able to communicate with each other and with a central computer complex. Local word-processing terminals unable to communicate with terminals elsewhere in the network fail to maximize their usefulness in an electronic message environment.

These goals, together with a corporate records management program, put CISDs in a position to meet the present demand for information at a lower cost, while increasing overall service levels.

In reality, fulfilling the two goals may parallel the actual services improvement, with each activity reinforcing the other. In analyzing how this may evolve, composition, distribution, and reaction cycles of a document will be examined in detail. Emphasis will be on a methodology which can be integrated into any Phase 3 network meeting the two objectives cited above, regardless of vendor software/hardware utilized.

Composition

"Information mapping"[4] is a method for defining and enforcing corporate documentation standards. Originally used in training materials on technical subjects, mapping structures textual writing by dividing the page with horizontal and vertical lines to form logical blocks. Only information of one functional type exists in a given block. The concept, while extendable to almost the whole of corporate documentation textual in nature, will be illustrated in the context of internal and external correspondence.

The typical business letter or memo may be divided into logical blocks, as in Figure 2. It should be noted that the text-block could relate to abstracts for other types of documents.

Within each of these blocks, a typist enters words

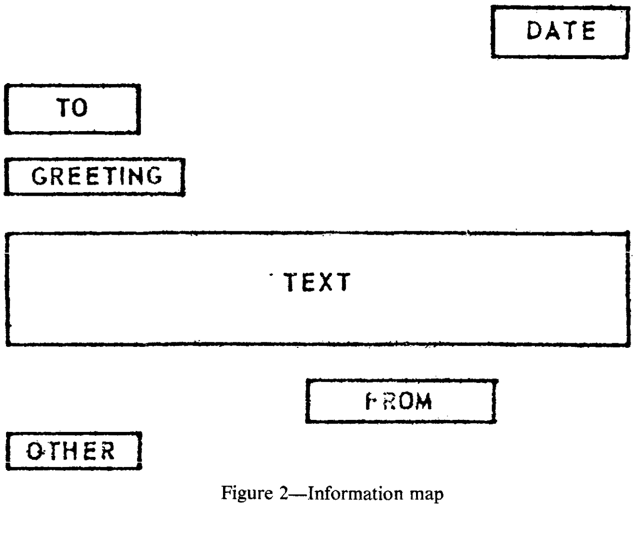


Figure 2—Information map

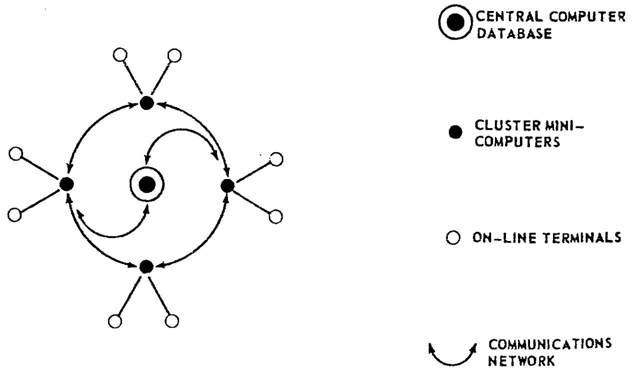


Figure 3—Phase 3 word-processing network

fulfilling a unique function—a type of information map. Chances are that not too far from the typist's keyboard is a secretarial handbook outlining the function and format for each block: specifications for vertical and horizontal spacing; addressee format and punctuation; in internal correspondence, when to use top-down or bottom-up departmental hierarchical TO, FROM identification; layout for dictator, typist, enclosures, copies, filing, references, attachments.

When correspondence is prepared on conventional typewriters, strict standards are often compromised. When using "intelligent terminals" programmed to validate operator entries, the skill in and practice of documentation standards are improved. At Dun & Bradstreet, Inc., an operator types a continuous string of characters prefixed by the logical block into which they are to be stored for printing.[5] When applied to information mapped correspondence, the typist may enter:

:DATE: December 1, 1976
:TO: A. R. Jones President, Software Inc.
2100 Broadway
New York, NY 10000

:GREETING: Dear Art:
:TEXT: Further to your letter of. . . . . .

.
.
.

and early resolution of the problem.
:FROM: Sincerely, T. R. Smith
:OTHER: DRJ/SWC

This type of data entry has been found superior to traditional CRT template orientation. Standards of 12,000 strokes per hour (equivalent to a typing speed of 40 wpm) have been established. Blocks of a document may be entered in any order and immediately validated for content, format (punctuation, indentation, spelling), and block length. The program which performs these activities may also establish page-positioning for display. Dun & Bradstreet has found that a new terminal operator can reach full productivity in one-quarter of the time that would be required in using a manual system.

Indexing/keywording

The most important aspect of standards established and enforced via information mapping comes to bear in computer indexing of documents.

With text-editor/data entry systems operating on a multi-terminal minicomputer, the indexing of material in the cluster is now feasible. Even when equipment manufacturers provide no generalized software specifically supporting data entry based upon information mapping, programs tailored to corporate specifications are installable. Indexing a document, in which physical position on the page defines the context in which keywords and phrases are used, is a straightforward approach to an otherwise difficult problem.

At Shell, a computer program has been written to process correspondence as mapped in Figure 2. Several activities are carried out by that program:

1. Non-text blocks are validated for content and format.
2. Non-text blocks are indexed (as multi-valued fields, when required).
3. As an alternative to keying all significant single words, keywords and phrases specified by the document's author are extracted from the text blocks.

The distinction at this point between "indexing" and "keywording" is important. In indexing correspondence, for example, an individual name ('JONES, T') appearing in the TO or FROM or COPIES blocks is an *index field value*. The context in which 'JONES, T' is used is clear-cut, both in how a computer would store the index field-value and how an information-seeker would query the computer database:

FIND TO IS 'JONES, T'

A document with multiple authors would be assigned multiple index field-values:

FIND AUTHORS ARE 'JONES, T' AND 'SMITH, A'

On the other hand, 'JONES T' appearing as a *keyword* in the documents' text-block is subject to varying degrees of usefulness in retrieval:

FIND 'JONES, T'

might retrieve documents in which the name was only mentioned incidently or miss documents where 'T JONES' appeared in the text-block. Clearly, terms which appear in document text do not always provide an adequate basis for retrieval, even when specified by the document's author. Automatic extraction of all non-stop keywords, like specified keywording, may at times produce too few or too many terms, causing a wide variability in the completeness and relevance of documents retrieved. Thus, other keywording is advised.

Extra keywords which enrich or clarify the subject content of documents may be assigned by a local information specialist from a controlled corporate thesaurus. Entered as a special information-map block, the keyword contents would not be routinely displayed in the distributed document.

Distribution

Once a document's composition cycle and indexing/keypunching based upon information mapping concepts are completed, the delivery may be carried out in several ways.

The document itself may be printed within the word-processing cluster and transmitted via regular mail. For speedier transmission to a physically distant addressee, the network may be used to transmit the document in a fraction of the time it took for initial entry (based upon an average 2400-characters-per-second transmission speed versus 40 WPM entry rate).

Indexing and keyword information may be held in the database of the local cluster for security considerations or for documents of narrow interest. Otherwise, indices, keywords, and the document itself are transmitted to the central computer for processing by the CISD.

Storage

Textual documents historically have arrived at CISDs in printed form. The effort in manually indexing, filing, and micro-filming documents of diversified formats is a task many CISDs are already hard-pressed to handle. With the projected increase in printed matter arriving at CISDs, staff-time now given to information seekers may be shifted to meet the increased indexing and filing burden. The completeness and relevance of information retrieved is bound to degrade unless the two objectives cited earlier are sought.

In an enhanced services environment, computers will perform many of the indexing, filing, and retrieval functions that earlier required human manual and intellectual effort. The information specialist will coordinate the receipt of electronic messages from all nodes of the corporate network that have been directed to the central computer complex. Electronic transmission of documents from one node to another need not pass through the central node. However, if the document sender or recipient wish to participate in the new services provided by CISD, the indices/keywords and, optionally, the document itself must be enqueued for processing by the CISD specialist.

Filing and retrieval services provided by CISD include:

1. Processing (computer) index/keyword messages. Validate format and control posting in corporate database. Maintenance and distribution of corporate thesaurus for keyword enrichment.
2. Collecting documents for archival tape storage and output to film.
3. Preparing and distributing micrographic files to designated word-processing clusters, with their accompanying computer generated subject-indices.
4. Maintaining employee interest profiles. Matching profiles to updated database. Distribution of relevant information via the network or regular mail.

Retrieval

With the document life cycle described above, a mechanism now exists to help anyone in the document reaction cycle.

The recipient or author of a document mailed electronically need only retain a printed copy until the arrival of the next set of micrographic files in the departmental word-processing cluster. Archival retrieval may always take place using the computer-generated subject index and film readers located in the cluster. If extra copies of a document are needed, a film-to-paper copier is available.

For documents not authored or mailed to the departmental cluster, authorized employees may interrogate the CISD maintained corporate database on the central computer. 'Queries are entered at a low-speed terminal, transmitted over the network to computer programs interacting with the database, and the results returned to the user.

A variety of traditional retrieval tools are available to aid the on-line information seeker. In formulating a query which will be complete and precise, a user may selectively display at his terminal:

1. An alphabetical thesaurus listing of index fields (TO, FROM, SUBJECT, TITLES, AUTHORS) and keywords with a count indicating the frequency with which the terms appear in the database.
2. Vocabulary analysis of terms, indicating possible alternate terms, e.g.,

—SEE ALSO'S—
Data Management—See File Management

—GENERIC RELATIONSHIPS—
Information Systems Manager—
BROADER TERM—Center General Manager
NARROWER—Information Design Manager
Information Support Manager

Retrieval using index/keyword techniques discussed earlier, together with aids for query formulation, has been measured and the performance judged competitive with pre-coordinate, strictly controlled keywording approaches.[6]

If further retrieval assistance is required, a user's query may be processed by a CISD specialist at the central complex and the results displayed on the user terminal within the cluster. Together, they are able to obtain sufficient information to learn which documents are relevant to the information seeker's interests and where the entire document may be retrieved. For locally authored/received documents, the film reader may once again be consulted. Another alternative is to interface a CRT to a micrographic retrieval device, which in turn is linked to a database index. When the computer responds to a user query, the resultant micro-image may be transmitted directly to the microfilm reader and the image displayed automatically. Otherwise, there are several alternatives for delivery of the document set:

1. Transmission over the network for high-speed printing within the cluster.
2. Normal mailing of batch print-out from the central computer.
3. Normal mailing of film for reproduction.

OTHER CONSIDERATIONS

As the services described above are put into place, the benefits which follow may be measured by more than just the availability and immediacy of information.

Word-processing is estimated to yield increases in document productivity of 100 to 500 per cent.[1] With the saving, an initial capital equipment expenditure of $15,000 to $20,000 per cluster operator station is expected to be paid out in two or three years.

The one-time expense of manual filing, mailing, and retrieval of a 750-character document is estimated at $.59.[7] To store the document prepared via word-processing for on-line retrieval would run approximately .004 cents per month. A retrieval using one of several popular storage and retrieval systems would run $.05 to $.25, depending on a variety of factors.

The expense of present information services versus enhanced services in a word-processing network environment may justify the implementation of the latter.

Other benefits are less obvious. In order for the word-processing network to function properly, there exists within it a directory of corporate employees. Each is identified by:

1. Employee number
2. Name/Title
3. Cluster assignment
4. Hierarchical department/function/role designation
5. Network security authorization
6. Telephone number

With the directory, documents may be related to authors/recipients with employee numbers alone. The formatted report, depending on its type, will substitute the appropriate employee text. Tags grouping different employee collections or hierarchical department designations may be built and stored to aid in the automatic preparation of routing information. Combinations of employee number, departments, and security authorizations will also be used to permit only qualified individuals to interrogate cross-sections of the CISD document database. The directory may serve other purposes as well.

As an interface with the corporate personnel database, the inventorying of employee skills, training, job history; the preparation and distribution of payroll are aided. The directory also acts as a convenient source for compiling organization charts and telephone directories.

Improved documentation standards and services, if adopted within the cluster, mean that personal filing practices may be impacted. The employee and a cluster information specialist will enter keyword/indexing information into the local database. Film and paper files are maintained by the technologist, leaving more time for the employee to pursue normal job responsibilities.

CONCLUSION

It is said that 5-10 years will elapse before the declining hardware costs will make widespread corporate word-processing clusters commonplace. In that time-frame, ". . . the cost of communications systems will drop so far that all kinds of applications which seem exotic today are going to become feasible and attractive."[8] Arthur D. Little, Inc., has reported that by the 1980's, decreases of 50 percent for central processing, 66 percent for communications, and 80 percent for terminals are likely. Costs for storing a page of information will decrease by a factor of 30 when paper is replaced by other media.

In the interim, lack of planning could result in divergent objectives and disparate systems implemented throughout the corporation. The potential for waste and duplication of effort is very real in parochial approaches to corporate word-processing.

Information service departments should move to:

1. Establish documentation standards for use in word-processing clusters to facilitate automatic indexing;
2. Cast a frame-work for decision-making in cluster implementation and operational procedures;
3. Coordinate planning in anticipation of providing full-fledged services in a Phase 3 word-processing network.

ACKNOWLEDGMENT

The comments and support of H. D. Baade, M. K. Johnson, and J. M. Wainright were invaluable in the drafting of this paper.

REFERENCES

1. Purchase, Alan and Carol F. Glover, "Office of the Future," *Business Intelligence Program,* Menlo Park, California: SRI, April 1976.
2. Martin, Shirley M., Edgar S. Von Gehren and Ronald P. Uhlig, *Practical Experience in Computer Based Message Systems.*
3. Vezza, A. and M. S. Broos, "An Electronic Message System: Where

Does it Fit?," Massachusetts Institute of Technology, Laboratory for Computer Science.

4. Horn, Robert E., "Information Mapping," *Datamation*, January 1975, pp. 85-88.
5. Gaudette, J. M., "Word Processing at Dun & Bradstreet," *Datamation*, November 1975, pp. 76-88.
6. Lancaster, F. W., *Vocabulary Control for Information Retrieval*, Washington, D.C., Information Resources Press, 1972, pp. 135-152.
7. Dartnell Corporation, "Inflation Soars 1975 Business Letter Cost to $3.79," Analysis and Staff Report, Chicago, Illinois, 1975.
8. Ferreira, Joseph and Jack M. Nilles, "Five-Year Planning for Data Communications," *Datamation*, October 1976, pp. 51-52.

A subject-content oriented retriever for processing information on-line (SCORPIO)*

by CHARLENE A. WOODY, MICHAEL P. FITZGERALD, FRANCIS J. SCOTT, and D. LEE POWER
Library of Congress
Washington, D. C.

ABSTRACT

Traditional obstacles to the development of on-line software for library/information center functions are overcome in the design and implementation of SCORPIO, an on-line information retrieval system developed at the Library of Congress, through the use of an "elastic" data base architecture, the phased development of reusable/disposable software, and the design of an open-ended, non-technical retrieval language. The data base architecture allows arbitrarily large files with no limit on the size of individual records. Software is developed in short time intervals to minimize investment and maintain customer interest; and the software itself is isolated from environmental factors most subject to change. The retrieval language is non-technical and system responses are people-oriented.

INTRODUCTION

Throughout the United States, automation supports the library/information center function with administrative tasks, circulation control, selective dissemination of information (SDI) and on-line information retrieval. As more information center functions become automated, customers demand an expanded scope of services and on-line access to more and more data bases. SCORPIO, an on-line information retrieval system, was developed by the Information Systems Office of the Library of Congress to address the special problems that the information centers in the Congressional Research Service (CRS) confronted in meeting quickly the demands of their customers—members of Congress and researchers in the CRS.

Traditionally, on-line access to library-type data has been constrained by the variety and size of both data bases and individual data records, by the huge investment required for software to maintain the data bases and to retrieve information on-line, and by the complexity and restrictiveness of retrieval languages. In the design and implementation of SCORPIO, such problems have been circumvented through: the use of an "elastic" data base architecture, the phased development of reusable/disposable software, and the design of an open-ended, non-technical retrieval language which can be used directly by customers, without an information specialist as intermediary.

Functional description

By using the SCORPIO retrieval language, a terminal user can perform any of the normal functions available from on-line information retrieval systems. After specifying the file to be searched, the user may browse the text dictionary or the index of available retrieval terms, select documents based on relevant terms, and combine selected documents with the Boolean operators "and," "or" and "not." Some files have hierarchical thesauri to aid in search term selection. Resulting sets of documents may be displayed, in part or in full, at any point in the search. The same commands are used for all files; and documents from any file known to SCORPIO may be displayed, though not selected, at any point. In effect, each file contains the data of the others.

Operating environment

The 37 working programs and 63 tables comprising SCORPIO, all re-entrant and written in ALC, require 99K bytes of storage. These programs running under control of a multi-tasking monitor (IBM's Customer Information Control System (CICS)), share resources with other on-line programs in a 1-megabyte region of the Library's IBM S/370 Model 158 computer.

THE "ELASTIC" DATA BASE

Each data base that is a candidate for on-line access has its peculiarities as to content, number of data records, record size and requirements for display. To allow data bases to grow, and to accommodate new ones, the following design goals were set for the data base architecture:

- no limit on the number of records in the data base

* A paper to be delivered at the National Computer Conference, Dallas, Texas, in June, 1977, on behalf of the Special Libraries Association and the American Society of Information Science.

- no minimum or maximum size for a single data record
- direct access to any data record or part of a data record (required for on-line display)
- on-line access concurrent with file maintenance
- addition/deletion of data fields without total file reorganization
- efficient use of auxiliary storage

Figures 1 through 4 illustrate the data base architecture developed to meet the design goals.

Physical space organization

A basic file consists of two physical files: a prime data space and an index file (Figure 1). The prime data space is partitioned into fixed-length areas, called "bins." Initially, all bins are elements of a "free space" list, each containing a link pointer to the next bin in the list. When a logical record is created or updated, bins are removed from the "free" list for use in data storage. To limit the number of auxiliary storage accesses required to process a logical record, the size of each bin is set at ⅓ of the projected average logical record length (which varies for each data base). Each bin contains an 8-bit "bin status" indicator and space for a "chain pointer," which allows for the logical continuation of data from one bin to another. The index file consists of fixed-length records, each containing a symbolic key and a pointer (bin number) to a "base" bin in the prime data space.

Data organization

Each logical record consists of data stored in one or more bins. The first bin, called the "base bin," contains (1) the fixed length fields which occur in every record and (2) the beginning of the "field directory." The "field directory" affords direct access to any "variable" field—a field of varying length which may or may not exist in each logical record. An entry in the field directory, consisting of a 16-bit field "tag" and a pointer (bin number) to the bin containing the data field, is present only for fields actually present in the record. If a field extends over more than one bin, the chain pointer of the bin points to a "continuation" bin, providing for fields of any length. The "field directory" is a data field that may be continued in the same fashion, affording an unlimited number of data fields. Data fields are updated "out-of-place"—new bins are acquired to hold the modified or new data, and the "field directory" entry is updated. Supplanted data bins are restored to the "free" list using standard list maintenance techniques.

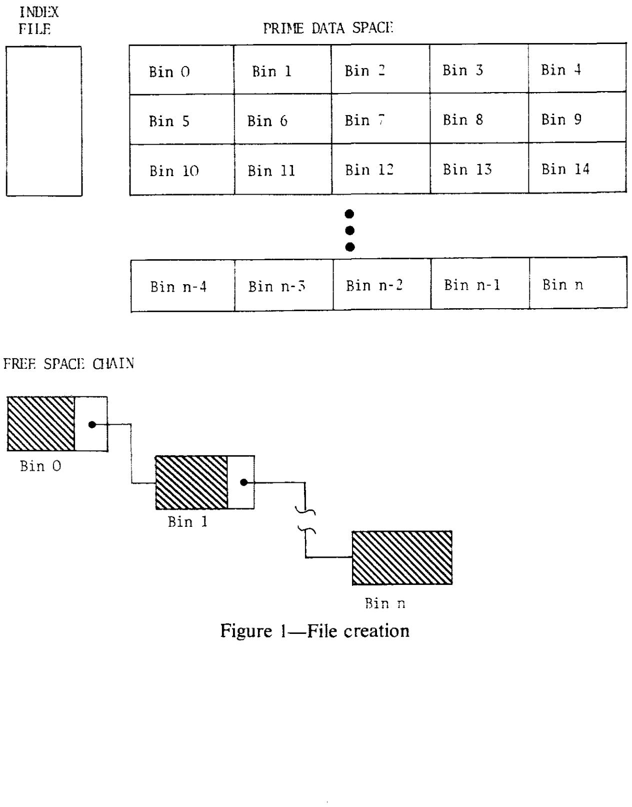


Figure 1—File creation

Each data field (Figure 3) is represented within a bin by a 16-bit binary number, or "tag," followed by a 16-bit length field. The data field may be subdivided further into subfields of fixed or varying length, each of which may be tagged (as to content) with an 8-bit code.

To achieve a degree of data independence each file is described by a File Descriptor Table, which contains a description of the physical characteristics of the file and, for each field, its storage format. In addition, each data field is associated with a symbolic name and other information used in the file maintenance and display programs.

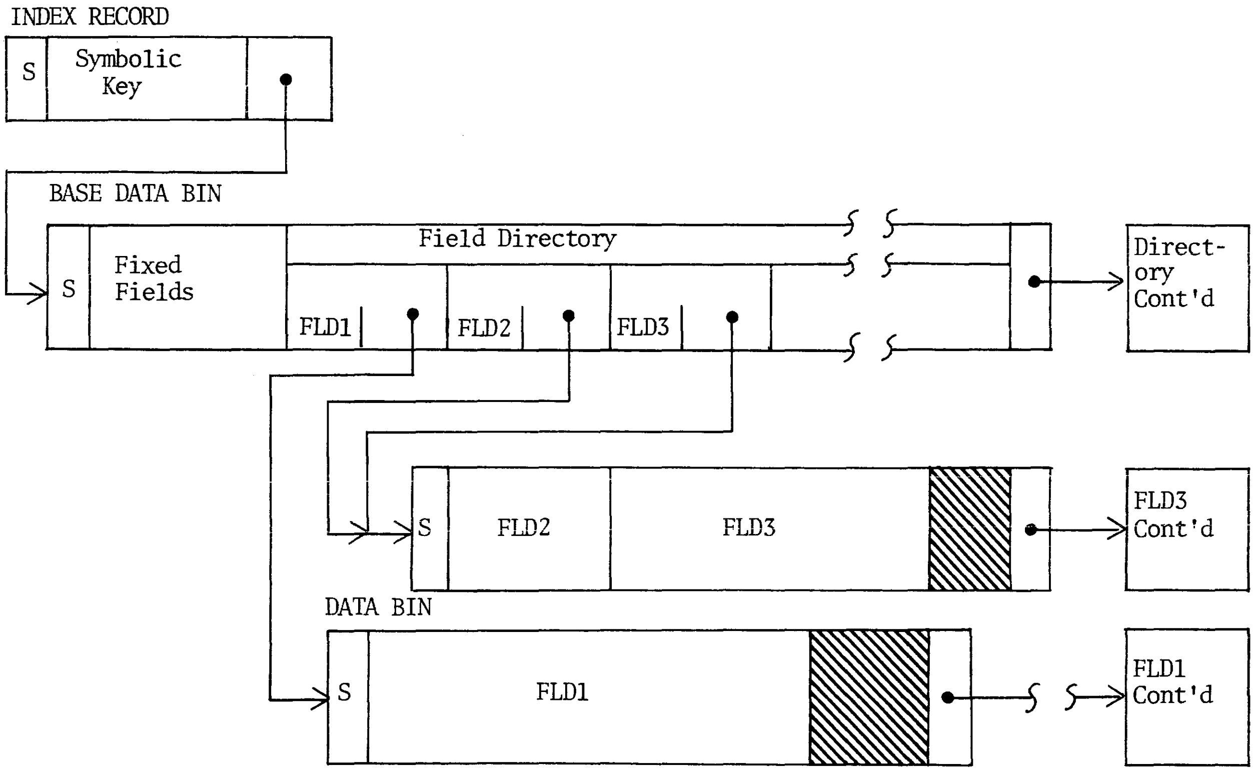
Each data file has an associated "inverted" file for subject access. The inverted file entries are updated on a regular schedule after the data file is updated (Figure 4). Data in the inverted file are stored in the same format as data in the prime data file so that both may be processed by the same data management software. Symbolic keys in the inverted file are subject terms; and data fields in the inverted file are lists of symbolic keys of data records in the prime file.

Due to the data base architecture, the same software (with well-defined, customized "special processing" exits) may be used to maintain and access any file available to SCORPIO—shortening development time and minimizing investment in new software.

REUSABLE/DISPOSABLE SOFTWARE

Software development is a major expenditure in any automation project, especially since personnel costs are rising as hardware costs are beginning to fall. A major component of SCORPIO is the software development philosophy, which requires a new product or facility every three to four months, and requires "dependency isolation" in programming.

The literature of automation is full of horror stories describing software projects laboring to produce a mouse. Millions of dollars have been expended for three or more years to develop a product which failed to meet the current needs of the sponsoring organization. In recognition of the fact that organizational requirements change over time, each new facility for SCORPIO is developed in three to



STATUS BIT VALUES

Bin Type	x.......	0= Base Bin 1= Data Bin
Updated	.x......	1= Updated recently
	..x.....	Not used
Inactive	...x....	1= Candidate for addition to "free" list
Deleted	....x...	1= Available for reuse; not on "free" list
Split	.....x..	1= Last field on bin has been split
Defective	......x.	1= Defective or incomplete
Continued	x	1= Last field on bin is continued on bin addressed by chain pointer

Figure 2—Logical record structure

four months—requiring a minimum investment by the organization, maintaining customer interest and providing for a customer "feedback loop." With a working product in hand, customers can visualize future automation requirements and evaluate knowledgeably other cost/benefit trade-offs. If newly-developed products fail to meet customer requirements, they may be rewritten or simply discarded with minimum loss.

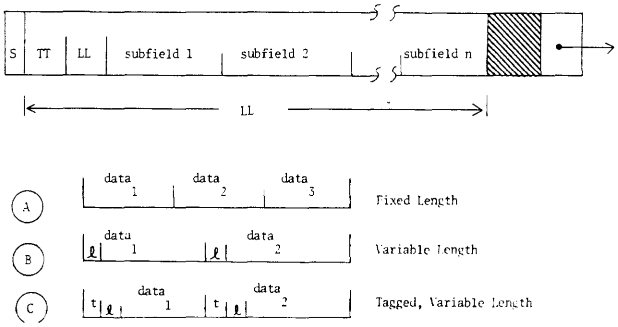
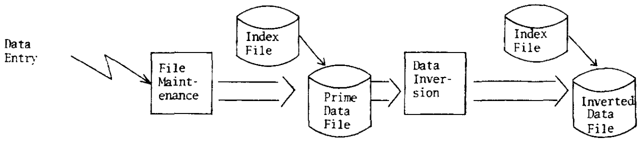


Figure 3—Data field structure

For the software development team, programming in a changing environment is simplified by "dependency isolation." In the development of SCORPIO, three major envi-



Index	Data	Index	Data
SNO2620	BILL sn02620	CANNON	SPON SNO2619
	DATE 11/5/75 ·	●	SNO2620
	SPON Mr. Cannon	●	
	INDX Trusts	●	INDX SNO2619
	Interest Rates	TRUSTS	SNO2620

Figure 4—File maintenance and inverted files

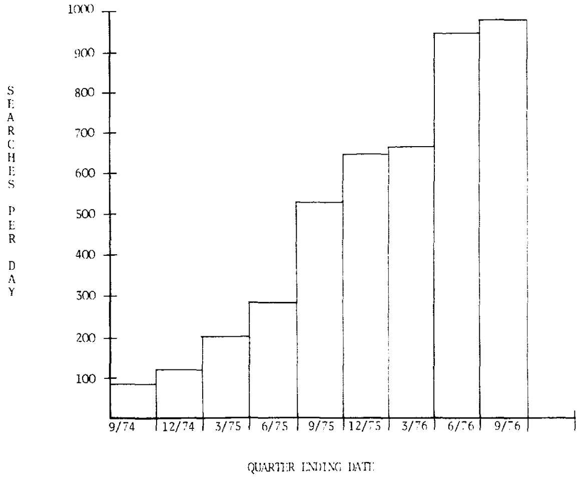


Figure 5—SCORPIO searches per day, September 1974 to September 1976

ronmental dependencies were identified:

- Terminal characteristics
- Data base characteristics
- Intra-program communication protocols

To avoid terminal dependencies, an "internal terminal" of limited functional capability was defined. Specifications for the "internal terminal" were:

- Upper case only (during input)
- Variable number of 79-character display lines
- Variable buffer size (minimum 960 characters)
- Limited functions:

—NL (skip to a new line)
—SOM (provide a start-of-message space)
—NP (new page (erase screen for CRT))

During input, a sub-program converts the device-dependent message to the internal format. In processing, each functional unit composes a response which is translated to an output form acceptable to the terminal which originated the request.

To circumvent data dependencies, a table-driven, data-directed display language was devised. With the data base architecture as a base, it is possible to define display functions based on the presence or absence of data elements in a data record. A programmer defining a display merely names the display option and defines a series of display commands ("skip to a new line," "move literal x to output," "move data field x to output") for interpretation by the Display Format program. Through subroutine calls, the Display Format program composes a response to a request for display.

To minimize dependencies in intra-program communication, all programs communicate at the external level, as though a terminal user had requested a service. All programs are written as "stand-alone," primitive functions, available to other programs as well as to the terminal user. Each functional component is isolated as much as possible from others. Individual programs, written in Assembler language, make heavy use of macroinstructions for common functions and local subroutine calls, including the Assembler language macroinstructions: IF, ELSE, DO, and CASE, developed by IBM.

The "dependency isolation" techniques used in SCORPIO have led to gratifying flexibility: new terminal types are accommodated with ease; new display formats are installed and discarded in a matter of hours; changes to a major feature of the system—the structure of keys in the inverted file—have been made with no consequences in system reliability; new commands are added without affecting the performance of existing ones; and the Mean Time Between Failure (MTBF) for the software in SCORPIO is six to eight months.

NON-TECHNICAL RETRIEVAL LANGUAGE

A major goal of the SCORPIO retrieval language is its use by "end users"—without specially-trained intermediaries. Contributing to ease of use are: the language conventions, the "accommodating" nature of system functions and the people-oriented nature of system responses.

As concerns the retrieval language, the only major restriction is that the first part of an input message must be a command code. The number of commands is deliberately limited; and all commands are free-form, imperative demands that something be done: "Select," "Find," "Browse." No change of "mode" is required for using search term selection, negotiation and display functions. Continuation of most commands requires a single keystroke.

System functions are "accommodating" in that the terminal user always receives a useful, specific response. For example, the response to an unsuccessful FIND command is an alphabetic display of the term index, providing the terminal user with information to be used in deciding how to proceed. In addition, the retrieval language is "tailored" for each data base, with different default processing options assumed for each function. Alternate command forms, including common mis-spellings, command abbreviations and fully-spelled command names are accepted in place of command codes.

System responses are "non-threatening" in recognition of two facts: most casual users approach a computer terminal with trepidation, and most consistent users dislike being scolded by a machine. No imputation of fault or error on the part of the user is made in any system message. The words "error" and "invalid" are not in the system vocabulary; rather, messages suggest that "SCORPIO was unable to interpret. . . ." In addition, all prompting messages are passive—"READY FOR NEW COMMAND" rather than "ENTER NEXT COMMAND," so that the user does not feel pressed to "keep up with" the computer. So far as possible, responses to commands appear in a form familiar to human beings, rather than in a rigidly-controlled form suitable for machines. For example, the prompting message in a multi-page response advises the user that the current page is "1 of 15" instead of "0001 of 0015." All of the preceding examples illustrate the elaborate precautions required of an on-line system which is to be used frequently by a "computer-naive" customer group.

CURRENT STATUS

Current usage of SCORPIO by the staffs of members of Congress, the Congressional Research Service and the Library of Congress indicate that an end-user-oriented information retrieval system can enhance the traditional services offered by information centers in a library environment. Daily use of SCORPIO began in February 1974; and Figure 5 shows the 1000 percent growth in SCORPIO usage from September 1974, to September 1976. Increased use appears to depend on two factors: an increase in the number of terminals and the availability of new files.

As of September 1976, the Library of Congress supported over 500 remote terminals, of all types, located in Congressional offices, the Congressional Research Service and all departments of the Library, of which approximately 250 are used primarily for accessing data bases under SCORPIO. Over 2,000 individuals have been trained in the use of SCORPIO, and records show that each terminal is used for four to six searches a day, with each search requiring eight to ten commands in a twenty-minute period. Terminals in the Library reading rooms are used by Library patrons with no training and only a few simple charts for guidance.

Seven data bases, requiring two billion characters of online storage, are available through the Library of Congress on-line system (processing 40,000 transactions a day, of which over 17,000 are SCORPIO commands).
The data bases are:

- *Legislative Information Files for the 93rd and 94th Congresses:* containing information on public general bills and resolutions. Retrieval terms are: bill number, public law number, sponsor, cosponsor, committee, index terms, actions and the short title of the bill.
- *Major Issues File:* a collection of concise, objective briefs on key issues of public policy. These briefs are written by CRS specialists in a set format that includes precise definition of the issue, a background and policy analysis, references to major current legislation, and additional references. As new issues emerge, they are added to the file. Retrieval terms are: issue number, title and index terms.
- *Bibliographic Citation File:* a collection of over 150,000 references to significant, current periodical articles, GPO and U.N. documents, selected CRS reports and interest group or lobby group publications. This collection is selected by the bibliographic staff of the CRS Library Services Division. Retrieval terms are: accession number, author and index terms.
- *Congressional Record Abstracts for the 94th Congress:* a collection of abstracts of the daily Congressional Record, prepared by Capitol Services, Inc., which guides a terminal user to relevant pages of the Congressional Record. Retrieval terms are: accession number, bill number and index terms.
- *National Referral Center Resources File:* a collection of more than 10,000 descriptions of "information resources", which are organizations qualified and willing to answer questions or otherwise provide information on virtually any topic in science and technology, including social sciences. Retrieval terms are: accession number, name of organization, geographic descriptors and index terms.
- *Library of Congress Computerized Catalog:* a file of over 600,000 references from the Library's MARC (Machine Readable Catalog) data base. This file contains references to English-language books catalogued since 1969, various foreign language books catalogued in the 1970s, and to the Science Reading Room collection of some 6,000 titles. Retrieval terms are: LC card number, LC classification number, title, author, and index terms.

Evaluations of end-user satisfaction with SCORPIO have been sporadic and unquantified. A study conducted by the Congressional Research Service for the Senate Subcommittee on Computer Services did find significant use of the Legislative Information files by the professional staff in Senate offices. Positive comments were made regarding the speed of response and ease of use of the system. Negative comments related to lack of training and system "downtime" due to terminal or central-site hardware failure. Reference cards, as opposed to voluminous reference manuals, were praised for brevity and clarity. Other measures of end-user satisfaction, such as the "comments log" maintained in the Science Reading Room for the walk-up, untrained customer who uses one of the three public terminals, are relatively unused, although the terminals are used over 100 times a day. More adequate user feedback mechanisms are planned for the future.

PLANS

Plans for the future development of SCORPIO include both new facilities and new files. Files planned for 1977 include:

- *Legislative Information File for the 95th Congress:* including expanded status information and the tracking of amendments, printed and unprinted, in a cooperative effort with the automation staffs of the House of Representatives and the Senate
- *Congressional Record Abstracts for the 95th Congress:* including retrieval by names of Senators and Congressmen
- Several *General Accounting Office Files:* including the Reports Required by Law data base.

New SCORPIO features include:

- Installation of SCORPIO in the computer centers of the Senate and House of Representatives
- Text retrieval (in addition to term retrieval)
- Various utility functions, such as on-line/off-line print, sorting of results sets, execution of stored queries, and on-line update of data records
- Investigation of machine-aided indexing, natural language retrieval and the addition of computational facilities.

Comparing equivalent network services through dynamic processing time prediction*

by SANDRA A. MAMRAK
The Ohio State University
Columbus, Ohio
and
National Bureau of Standards
Washington, DC

and

STEPHEN R. KIMBLETON
National Bureau of Standards
Washington, DC

ABSTRACT

Computer networks provide the potential for resource sharing. Realization of this potential requires knowledge of the available resources within the network. Moreover, if a given resource is available at more than one host, selection of the most appropriate host is required. This paper develops a dynamic means for host selection assuming that the evaluation metric is processing time. An experiment is described which provides an initial evaluation of the key component of the methodology on two separate systems. The paper concludes with a discussion of some overall insights into the applicability of the methodology and its implementation requirements.

INTRODUCTION

Computer networks provide a mechanism for sharing resources. At present, realization of this potential is limited by: the lack of a support facility for identifying available resources such as compilers, editors, transaction processors, etc., and their locations;[1] problems in educating users regarding access requirements of individual systems;[2] the problem of supporting any necessary data translation in an automated manner;[3] the requirement for translation of Job Control Languages;[4] and the need to select the "most suitable" of a collection of functionally identical services given that these other requirements have been met.

This paper describes a methodology for selecting the "best" host from among a collection of hosts providing a functionally identical service under the assumption that processing time is to be used as the selection criterion. This requires development of a means for estimating the processing time of an application job. We have adopted an empirical approach based on the observation that job processing time is the sum of the standalone processing time, the queuing time for the processor, and the queuing time for the I/O devices accessed by the job. Assuming, as we shall, that a synthetic module characterization of the job is known, it is straightforward to estimate standalone processing time. Thus it only remains to evaluate the queuing time. This is accomplished by initiating a Micro Benchmark which accesses each I/O device referenced by the application job and consumes a small amount of processing time in order to permit determination of average I/O and processor delay.

The general problem of selecting a processing site, hereafter referred to as the location selection problem, will be formulated in section two; section three describes the general, dynamic methodology which has been developed to meet the identified requirements. Section four then discusses the results of an experimental investigation of the suitability of the approach and section five concludes with some remarks regarding general implementation aspects.

NETWORK RESOURCE SELECTION

Developing an approach to network resource selection requires the specification of at least: (1) the entity responsible for performing the selection; (2) the support capabilities available for implementing the result of a selection decision; and (3) the metric used for making the selection decision.

The entity performing location selection decisions can be either the user or some network process. In either case one requires a capability for obtaining the information to be used in selecting the specific host computer containing the resource to be used.

At the present time, automated aids for network resource selection do not exist. This is not surprising in view of the fact that the basic components of a methodology are only now being developed. However, some theoretical analyses have been performed based upon queuing theory and integer programming techniques.[5-7]

Given that either the user or a network mechanism serves as the selection entity, the collection of possible decisions is still limited by the support capabilities available. A spectrum of such capabilities can be envisioned ranging from the rather unlikely, but very general case of decompilation of object code to generate source code capable of being executed on a different vendor's mainframe, to a relatively straightforward selection limited to a set of homogeneous systems having installed modules which are functionally identical. We note in passing that the capabilities being investigated under the names Network Access Machines[2] and Network Operating Systems[3] will significantly reduce the difficulties in providing ease of access to heterogeneous systems as well as those of moving jobs and data among machines.

Metrics for network resource selection can be based on a variety of factors including cost, ease of utilization or processing time. The issue of cost based selection has been investigated in other papers[8,9] with significant results. It has been shown that there are significant intersystem variations across product lines in the cost required to perform a specific job such as a compile or compile-load-go. Thus, the utility of cost based selection has been demonstrated. Moreover, work has been done in the area of maintaining a data bank of information on job types and the costs of their execution on different systems.

Selection based on ease of utilization is of natural interest. However, support of such an approach will require a model of the user since current research demonstrates that user perceptions regarding the adequacy of a command language are dependent upon the level of sophistication of the user.[10] As a result, there is no single "best" command language but rather only the most appropriate language for a given user. Indeed, the collection of constructs deemed appropriate to the accomplishment of a specific job will vary based upon user sophistication.

The two preceding metrics are relatively static since the decision made will be valid for relatively lengthy periods of time. There is also a need for a more dynamic metric allowing selection among systems with time varying workloads. This requirement has led to our interest in processing time as the metric for selection. Specifically, we assume that a user has access to a collection of functionally identical capabilities located on different hosts. The metric to be used to select the host-resource pair is processing time, and our concern is the development of a practical methodology for processing time estimation. That is, we will develop a means for predicting the processing time for functionally identical resources located on a collection of hosts and we assume that the user can determine the "best" host given this information. If the decision is solely based on processing time minimization, it is evident that the entire selection process can be automated. However, selection may also involve subjective factors such as reliability which force the user to make the decision. Note that we are not explicitly concerned with the identification of the set of hosts containing these functionally identical capabilities nor with the precise manner in which processing time estimates are evaluated to determine the host-resource pair to be used.

PROCESSING TIME PREDICTION

The collection of (non interactive) jobs processed by a computer system can be categorized depending on whether or not resource requirements are known. If resource requirements are unknown, it is unlikely that meaningful a priori statements concerning processing time (elapsed time from initiation to termination of job processing) can be made.

If job resource requirements are known it is reasonable to seek to predict the processing time of the job as a function of the state of the system at the time of the request for its processing. To accomplish this, two basic approaches can be considered: prediction based upon static or dynamic information.

Static prediction of processing time attempts to develop correlations between processing time and job resource requirements such as amount of core, total CPU time, number of disk accesses, etc. Such an approach has the attractive advantage of being simple to implement. Unfortunately, its independence of the state of the system at the time of arrival of the job renders the results relatively imprecise as previous investigations have shown[11] and as the intuition of the reader would suggest.

Dynamic prediction of processing time provides a potentially more accurate estimator; the disadvantage is the requirement for sampling either each time a location selection decision is made or, alternatively, on a continuing sample driven basis.

The basic approach used in this paper for estimation of the processing time (PT) of a job requires: (1) assuming that the synthetic module characterization of job resource requirements is known; (2) expression of the processing time of an application job, PT(AJ), in terms of this synthetic module characterization plus terms representing processor and I/O device delays reflecting the existence of other jobs in the system; and (3) estimation of these device delays for the application job through their measurement based on utilization of a Micro Benchmark which is a small test job for determining the average delay for the processor and each I/O device accessed by the application job whose processing time is to be estimated. Thus, execution of the Micro Benchmark results in the generation of a table of processor and I/O device delays. This information, in turn, combined with the synthetic module characterization of application job resource requirements permits estimation of job processing time. We now develop this basic idea in more detail.

Synthetic modules[12–14] provide an abstract characterization of a job in terms of its resource requirements on an individual system. Such characterizations consist of at least: an average CPU burst representing the amount of processor time used between I/O operations, a probability distribution characterizing the selection of an I/O device following the completion of a processor burst, and expressions for the amount of information to be read/written from a given device given that an I/O operation is directed to it (this permits determination of the I/O service time for the device).

Given the synthetic module characterization, one knows the total CPU time required to process the job and the number of I/O operations which will be made to each device. Thus, the processing time of the job can be expressed in the form:

$$* \quad PT = TCPU + \sum(NIO(i)*SIO(i)) + WCPU + \sum(NIO(i)*QIO(i)) + ST$$

where the sum is over all the I/O devices i in the system and:

PT	denotes the processing time of the job
TCPU	denotes the total CPU time of the job
NIO(i)	denotes the number of requests to I/O device i
SIO(i)	denotes the standalone time to service a request to device i
QIO(i)	denotes queuing time per request to device i
WCPU	denotes the total processor wait time
ST	denotes the system scheduling time for the job.

Note that the first and second terms in equation (*) are known from the synthetic module characterization of the job, the third and fourth are to be determined via utilization of the Micro Benchmark, and the fifth will be determined in a side experiment.

To estimate the third and fourth terms, observe that it is conceptually straightforward to construct a Micro Benchmark which accesses each I/O device in the system once, performs a small, known amount of CPU processing and terminates execution. In fact, since one is performing something conceptually analogous to sampling from a random phenomenon, suitable estimation of the average delay requires more than one access to each device. Thus, a Micro Benchmark consists of a series of NIO(i) requests to each device i in the system alternated with the utilization of known amounts of processor time.

Given that the Micro Benchmark has been executed, the anticipated average delay in accessing each I/O device can be tabulated, as can the average delay per unit time in utilizing the processor. (This argument tacitly assumes the scheduling of an individual job is independent of its resource utilization. If this is not the case, such scheduling effects must be estimated either through expansion of the size of the Micro Benchmark to place it in the appropriate scheduling class or, alternatively, through post processing of the delay estimators yielded by the Micro Benchmark.)

The fifth term reflects time used in scheduling the individual job which is, therefore, chargeable to the job but is of an overhead nature. The estimation of this term is discussed in the example.

To summarize, one first forms the synthetic module representation of the job, then runs the MB to obtain the delay estimators, and then uses (*) to estimate PT. We now describe the results of an experiment aimed at providing some insight into the accuracy of the results which can be obtained.

AN EXPERIMENT

To gain insight into the quality of the prediction afforded by our processing time prediction technique, test runs were made on two systems for a job accessing a single disk device. One of these systems is a computer used primarily for research, a DECSystem-10 running the TOPS-10 Monitor. The other is a production oriented UNIVAC 1108 running EXEC 8.*

THE MICRO BENCHMARK

The Micro Benchmark was written in Fortran in the interests of portability and is essentially a DO Loop whose interior consists of two major components. The first component is another DO Loop whose parameter can be adjusted to consume a specified amount of processor time. Thereafter, a routine is called which performs a READ from a file located on a disk followed by a WRITE. Utilization of the combination of a READ and WRITE to estimate average delay was adopted to "average out" any differences due to the existence of different scheduling policies for READs and WRITEs.

A key issue requiring resolution before execution of the Micro Benchmark is the determination of the index of the DO Loop. As commented earlier, it is unlikely that an index of one would yield a sample value for processor delay or I/O device delay sufficiently close to the population average. In view of this one can seek to use either a pragmatic, cut-and-try approach or to provide a theoretical justification for index selection. We have adopted the latter approach.

Theoretical justification of index selection is naturally approached via statistics. The underlying population is stochastic in view of the well-known time variation in computer system loading. However, if we assume that the rate of time variation in the basic population parameters is small in comparison with the time required to execute the Micro Benchmark, it is reasonable to use standard statistical techniques. Thus, we seek to determine that index N such that for a sample of size N, the difference between the population mean and the sample mean will be likely to be "small enough."

* The identification of certain commercial equipment and products in this report is done to adequately describe the experiments conducted. In no sense does the identification imply recommendation or endorsement by the National Bureau of Standards.

Note that the determination of N is specific to each individual system; indeed, in view of its dependence upon the performance of the Micro Benchmark, it may be time varying. However, one could determine an upper bound for N and use that as the common value. The indices used for N in our experiments are presented in the top two sections of Table I. Derivation of the value of N is discussed in the appendix.

Although the preceding discussion of the technique for selecting N is reasonable, some experimental justification seems desirable. As a result, test runs were made with jobs whose iteration parameter in the DO loop was a multiple of those in the Micro Benchmark. The agreement proved to be very satisfactory. Thus, if the DO loop index was 20 times that of the Micro Benchmark, the required I/O processing time was very close to 20 times that needed for the Micro Benchmark (actual values were 20.06 for the DECSystem-10 and 19.98 times on the UNIVAC 1108). This observation also lends credence to our assumption that system delay parameters change slowly in comparison with the Micro Benchmark processing time.

EXPERIMENTAL RESULTS

Two assumptions were made to simplify the initial evaluation of the adequacy of Micro Benchmarks. These are: (1) job resource requirements are multiplicative factors of those of the Micro Benchmark, and (2) CPU and I/O delay WCPU(MB) and QIO(MB) are combined into a single wait term W(MB)=WCPU(MB)+QIO(MB). PT(MB) can now be expressed as:

$$PT(MB)=TCPU(MB)+NIO{*}SIO(MB)+ST+W(MB)$$

where ST denotes the system scheduling time. As a result, the experimental data presented in Table I now permit us to write the following equation for the DECSystem-10:

$$PT(MB)=.186+5.248+.08PT(MB)+W(MB)$$

where, through a side investigation, the system scheduling time was found to be proportional to the total processing time of the job and is represented by the term .08PT(MB). (This proportionality to total processing time rather than standalone processing time seems generally consistent with knowledge concerning the properties of operating systems scheduling.) Given this information, if PT(AJ) denotes the processing time of the application job, whose processor requirements are n1 times those of the Micro Benchmark and whose disk requirements are n2 times those of the Micro Benchmark, we can now write:

$$PT(AJ)=n1{*}TCPU(MB)+n2{*}NIO{*}SIO(MB)$$
$$+.08PT(AJ)+W(MB){*}PT(AJ)/PT(MB)$$

This equation assumes that the ratio of the wait time to the total elapsed time for the application job will be the same as the ratio for the Micro Benchmark (for the special case which we are considering of a system consisting of one processor and one I/O device—the more general case would require modification of these arguments on a per device basis). Substituting the values provided in Table I and solving for PT(AJ) now yields:

$$PT(AJ)=(.186n1+5.248n2)/(.92-W(MB)/PT(MB))$$

Empirical investigation of the accuracy of this equation has shown that the predicted processing time was within 20 percent of the observed processing time approximately 75 percent of the time, regardless of the system load.

Prediction proved more difficult for the UNIVAC 1108 reflecting, perhaps, the more dynamic workload of this system. As a result, it proved necessary to enlarge the values of the indices of the DO loop to the "revised" values listed in Table I. Using the information displayed in Table I, the basic processing time prediction becomes:

$$PT(AJ)=(1.125n1+5.805n2)/(1-W(MB)/PT(MB))$$

and was found to predict within 20 percent of the observed processing time 75 percent of the time. Figures 1 and 2 graphically display the prediction error for both of the systems under study.

DISCUSSION AND CONCLUSIONS

The objective of this paper was the development of a methodology for selecting among hosts providing functionally identical services using processing time as the basis of selection. This requires development of a means for estimating processing time which, in turn, requires consideration of possible alternatives.

TABLE I—Stable Micro Benchmark Components

	MB Component	DO Loop Index	Mean Processing Time (ms.)
DEC System-10	CPU IO	6000 50	185.7 5247.9
UNIVAC 1108 (initial)	CPU IO	30000 10	224.5 326.9
UNIVAC 1108 (revised)	CPU IO	150000 250	1124.81 5805.75

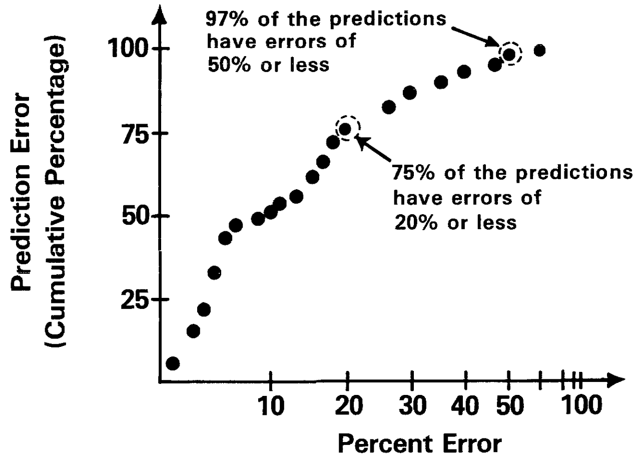


Figure 1—Error in response time prediction: DEC System-10

Intuitively, it seems reasonable that a dynamic approach to processing time prediction based on utilization of a Micro Benchmark should prove more satisfactory than a static approach based on a limited number of system performance measures. Acceptance of this statement, however, requires demonstration of the feasibility of the Micro Benchmark approach. This requires formulation of the general approach coupled with experimental evaluation.

Although these advantages are attractive, they must be weighed against the disadvantages reflecting: (1) costs of executing the Micro Benchmark, (2) difficulties in determining the synthetic module characterization of job processing requirements, and (3) questions concerning the feasibility of the approach in a complex environment with jobs accessing a large number of system resources.

The first problem is essentially a sampling problem and seems amenable to further investigation via sampling methodologies. The second is effectively an organizational issue since most of the required information is available to the operating system and, via some of the more sophisticated system accounting packages can be made available to the user.[15] The third problem requires further investigation. Although it is our opinion that the Micro Benchmark approach is generally feasible, verification requires continued testing and evaluation in operational environments. In addition, extension of this approach to include estimators for scheduling time, i.e., the elapsed time from arrival of the job until initiation of service, would also be of interest.

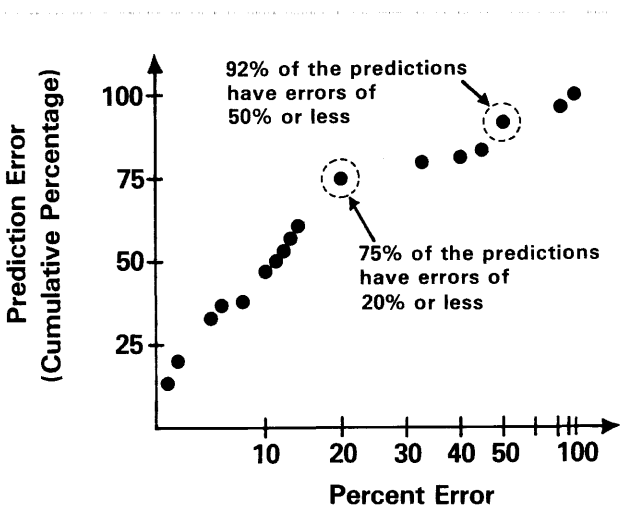


Figure 2—Error in response time prediction: UNIVAC 1108

In summary, it would appear that development of a methodology for processing time based selection among host computers is feasible, that the overhead cost in utilizing the methodology is acceptable, and that continuing investigation is in order.

REFERENCES

1. Benoit, J. W., E. Graf-Webster, "Evolution of Network User Services—The Network Resource Manager," *Proc. 1974 Symposium, Computer Networks: Trends and Applications,* IEEE Inc., May 1974, pp. 21-24.
2. Rosenthal, R., "Network Access Techniques—A Review," *AFIPS 1976 Conference Proceedings,* National Computer Conference, Vol. 45, June 1976, pp. 495-500.
3. Kimbleton, S. R. and R. L. Mandell, "A Perspective on Network Operating Systems," *AFIPS 1976 Conference Proceedings,* National Computer Conference, Vol. 45, June 1976, pp. 551-559.
4. Uzgallis, R. C., "Four Language Experiments in Computer Language Design," University of California, Los Angeles, California, July, 1975.
5. Balachandran, V., J. W. McCredie, and O. I. Mikhail, "Models of the Job Allocation Problem in Computer Networks," *COMPCON 73—Seventh Annual IEEE Computer Society International Conference,* IEEE Inc., 1973, pp. 211-214.
6. Barr, W. J., "Cost Effective Analysis of Network Computers," Dept. of Computer Science, University of Illinois at Urbana-Champaign, Report No. UI-DCS R-72-538, August 1972.
7. Bowdon, E. K., "Network Computer Analysis," Dept. of Computer Science, University of Illinois at Urbana-Champaign, Report No. IU-DCS R-72-505, January 1972.
8. 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.
9. Mamrak, S. A., "A Network Resource Sharing Module to Augment User Cost-Benefit Analysis," *Proceedings 1976 Symposium, Computer Networks: Trends and Applications,* IEEE Inc., November 1976.
10. Heafner, J. F., "Protocol Analysis of Man-Computer Languages: Design and Preliminary Findings," ISI/RR-75-34, Information Sciences Inst., Univ. of Southern California, Marina del Rey, June 1975.
11. Mamrak, S. A., "A Predictive Response Time Monitor for Computer Networks," *Proceedings Third International Conference on Computer Communications,* Toronto, Canada, August 1976, pp. 626-630.
12. Buchholz, W., "A Synthetic Job for Measuring System Performance," *IBM Systems Journal,* Vol. 8, No. 4, 1969, pp. 309-318.
13. Sreenivasan, K. and A. J. Kleinman, "On the Construction of a Representative Synthetic Workload," *Communications of the ACM,* Vol. 27, No. 3, 1974, pp. 127-132.
14. Kimbleton, Stephen R., "A Heuristic Approach to Computer Systems Performance Improvement, I—A Fast Performance Prediction Tool," *Proceedings 1975 National Computer Conference,* Vol. 44, June 1975, pp. 839-846.
15. Durbin, G., et al, "Analysis and Validation of the System Management Facilities of IBM Operating System MVT," Tesseract Corporation Report, 1976.

APPENDIX

Determination of the appropriate value of N given that the distribution of the underlying population is normal with known variance requires utilization of the Student's t distribution. Thus, let s be the population standard deviation (determined through a side experiment), and assume that we wish a confidence level of 100(1−a)% that the true population mean m will lie in the interval (EX−d, EX+d), where EX denotes the mean of the random variable X. Then the appropriate value of N is given by:

$$N=\frac{(ts)^2}{d^2}$$

where t corresponds to the value of the Student's t distribution for (1−a/2), N−1. Since the value of t viewed as a function of N is effectively constant for N>=30, this equation permits straightforward determination of N for the values (>=50) used in the Micro Benchmark.

A structured data base computer conferencing system

by GEORGE W. ARNOLD* and STEPHEN H. UNGER
Columbia University
New York, New York

ABSTRACT

A system called CBIE (Computer Based Information Exchange) is described, whose function is to facilitate communications among groups of people who cannot conveniently meet face-to-face. Each individual, at his own convenience, examines the existing conference records, via a terminal connected to the computer by a telephone line, and adds his own comments. A key feature of CBIE is that the conference record is structured by the users in the form of a network of elements systematically related to one another. The system is designed so that, utilizing a simple set of commands, users can peruse the stored information in an orderly way in accordance with their own interests. Applications include meetings of faculty committees, governmental committees, people engaged in joint research or development projects, and seminars or courses. An initial version of CBIE has been implemented under the RSTS time sharing system running on a PDP 11/50. A general outline of the implementation is presented.

INTRODUCTION

A major occupation of modern man is attending meetings. Few vocations are exempt from large doses of this activity; some—such as the academic and governmental professions—are notorious for overindulgence. If one thinks of meetings as including all occasions on which two or more people exchange significant amounts of information on one or more well defined topics often with the object of making decisions, then we are indeed discussing an important matter.

Certain basic problems arise when frequent meetings are necessary in a fast paced, mobile 20th Century environment. The necessity for such meetings places a severe drain on the time of those who are frequently involved, and this, in turn, causes scheduling problems. That is, it is often difficult to find a time that is mutually convenient for each member of a set of busy people. The necessity for interactions among widely dispersed individuals often makes it difficult to find a suitable place for a meeting. The need for people to travel to the meeting place constitutes an additional burden on their time—as well as entailing financial costs.

In a broad sense the necessity for many meetings, particularly those suffering from the above mentioned difficulties, is a consequence of our complex, technologically based society. Fortunately technology can also be part of the solution.

The most obvious and widespread form of technologically assisted meeting is of course the two-way telephone conversation. Many serious two-person meetings are held via telephone, and at a modest price the problem of distance is solved. Much less widely used is the multi-person telephone conference. Using standard equipment, telephone conferences involving up to 30 locations can be set up.[1] At present, with commercial telephone systems, these must be arranged in advance through a special telephone conference operator, who calls each party involved. This is a high cost operation in comparison with directly dialed two-person calls. However, conferences involving up to four parties can be dialed up directly by people served by the most modern telephone central offices,[2] and military versions of the same equipment are used for direct dialing of even larger conferences.[3]

Contrary to earlier beliefs, two-way telephone conversations are quite satisfactory for most purposes, despite the lack of non-verbal cues that we use to facilitate communications in face-to-face situations.[4] Furthermore, experiments with conference calls involving nine or more people have shown that they are also quite satisfactory to the participants; such problems as identification of speakers and orderly access to the floor are not as serious as had been anticipated.[5]

Video conferences would be still more satisfactory in general, and where pictorial displays are required, such facilities are particularly important. But video conferences are quite expensive and awkward to arrange. This will undoubtedly remain the case for at least a decade. Less expensive techniques for conveying pictorial data in restricted forms are, however, becoming available, for example, the remote blackboard and the scribblephone.[6]

* This author is a member of the technical staff of The Bell Telephone Laboratories and his work on this project was partly supported by that organization's graduate tuition re-imbursement program.

It thus appears that the telephone plant now in existence and new facilities rapidly being brought online can be effectively used for many meetings, both two-person and multi-person. Where the main problem is the spatial dispersion of the participants, this can be a very satisfactory solution.

A very different approach has been taken to deal with the problem of temporal dispersion as well as certain other problems not yet discussed. This is to use a computer to buffer information fed in by conference participants from (usually) remote terminals. Basically the computer is used as a bulletin board on which participants can write messages and read those posted by others. A key point is that each conferee can, at any time, dial up the computer to review the status of the conference and to add his own input as desired. Thus it is not necessary that all participants be available at any particular time. This non-real time feature is a principal advantage of computer conferencing. Anyone with a terminal that can be linked to a phone line can participate, and since an ordinary voice channel is quite adequate for such transmissions, the cost for remotely located conferees is not excessive.

The idea of computer conferencing seems to have been first implemented by Hall[7] and Turoff[8-10]. They were particularly interested in implementing Delphi systems (a technique for arriving at quantitative estimates of difficult to assess factors by interweaving the estimates of a number of individuals), but they also implemented general purpose systems. Facilities are incorporated for directing messages to specified subsets of the participants and for the casting, tabulating and displaying of votes. A number of other people have also explored the general concept.[6,11-13]

The subject of this paper is an enhancement of the computer conferencing idea, whereby the items entered are not simply recorded as a linear sequence, but can be structured in a meaningful manner by the conferees. The system, called CBIE (Computer **B**uffered **I**nformation **E**xchange), which has been implemented under the RSTS time sharing system of the PDP-11, will be described from the user's point of view in the next section and from the implementation point of view in the third section of this paper. It has been tested via an ongoing conference, on the development of the system, which has significantly aided cooperation between the authors, who are able to meet face-to-face only once a week.

A USER'S VIEW OF CBIE

Basic concepts

‚A conventional meeting whether face-to-face or via telephone takes place in *real time*, that is, every event occurs and is simultaneously observed by all participating members. The computer conferences mentioned are not taking place in real time, since different participants may "tune in" at different times to review past events. However both classes of conferences may be considered as linear, in the sense that each *input* to the conference is added to the end of a linear list of items. This is obviously true for conventional meetings. It is also true for the computer conferencing systems referred to earlier, in the sense that each new item is entered, and subsequently displayed, in sequence, immediately after the previous new item.

When a conference is long and ranges over many topics, it is very useful to have the discussion, and the record at any time, structured according to topics and subtopics. Each participant (we refer henceforth to computer conferencing systems) can then review, in a systematic manner, those portions that are of immediate interest to him and append comments, questions etc. to the precise portions of the record that they pertain to. Cross references should also be possible.

Instead of being a linear list of items, the structure of the conference becomes a directed linear graph, in which each item is a node with arcs directed to other items (successors) that are related in the sense of being subtopics, elaborations, refutations, etc. (Actually, since the successors to each node are ordered, there is somewhat more structure than is generally true for linear graphs.) It should be convenient to peruse this graph, or network, moving back and forth among related items, and appending or deleting items as appropriate. Jumps to arbitrary items on some personal list should be possible. Since the type of network under discussion does *not* completely order the items sequentially, it should also be possible for a participant to ask the system to specify which items have been entered or altered recently. Other desirable features include arrangements for voting, bulletins broadcast to all participants and the ability to direct private messages (side remarks) to specified subsets of participants.

As will be discussed later, such a system, properly implemented, could be useful in a wide variety of situations including joint research projects, seminars, courses at many levels, faculty meetings, software development, or even as a means for an individual to accumulate an organized data base on some set of topics. It should be emphasized at the outset that no information is generated or structured by the computer. Both the content and structuring of the data is specified by the users. The role of the computer is to facilitate the inputting, structuring, storage and retrieval of data by a multiplicity of users.

An example

The operation of CBIE will now be illustrated thru an example. The implemented version, currently under revision, has most of the features to be discussed.

Assume now that you are a member of the Southwoods Board of Education, and that the school year is about a month old. Several times a week, perhaps as often as every evening, you connect your home terminal (either CRT or hard copy type will do) to your telephone data set and dial up the municipal computer center, logging on to the ongoing School Board Meeting. You identify yourself to the system by entering your personal password and are then presented with the initial "frame" of the conference. Let us pick up a typical session at this point (Figure 1).

```
   SOUTHWOODS SCHOOL BOARD MEETING (S, T) #2
1. RECRUITING OF A NEW SUPERINTENDENT (S, T)
2. BUDGET ITEMS (S, T)
3. TENURE DECISIONS (S, T)
4. CURRICULUM QUESTIONS (S)
5. DISCIPLINARY MATTERS (S)
6. MISCELLANEOUS ITEMS (S)
```

Figure 1—Initial frame

The first line of the frame is the *title*. After the descriptive phrase, the parenthesized symbols indicate whether there is text and/or a successor list associated with the item. In this case, there are both, and the fact that the S appears first indicates that a successor list is being displayed. The "#2" indicates that the reference number of the current item is 2. (At any time, the command #2 will result in the display of this frame.)

The six numbered successors are the titles of topics under discussion. Each of these has a successor list and some also have text associated with them—as indicated by the parenthesized symbols.

Suppose you would like to review the discussion of budget matters, and assume that you've already seen the text directly associated with this item (this might give the total budget figure and a reference to a detailed document on the budget). In order to see the list of subtopics under budget, you give the command S2, which results in the display of the frame shown in Figure 2. Seeing this, you might then wish to examine issue number 5, pertaining to the superintendent's salary. Hence you follow up with the command T5 (each command is terminated by a carriage return) to obtain the text associated with that item. After reading this material (Figure 3), you might then wish to follow the ensuing arguments and thus you call for the successor list by striking the S key (see Figure 4). Note that each frame can, on command, be accompanied by a line (not shown in our examples) called a heading, that gives such information as the author (if he wishes), the date entered and the date of the most recent change.

Reading the assertion that appears as the title of successor 1 in this frame, assume that you wish to pursue this thread further and hence call for the associated successor list by entering S1, which produces the frame shown as Figure 5. Note that the first successor in this frame is an item (with reference number 17 not shown) that was developed in the course of a discussion of another topic, namely successor 1 of Figure 1 (the development is not illustrated here). Someone engaged in the current argument added this cross reference by the use of a command I#17.

Suppose now that, after calling for and reading the text associated with item 3 of Figure 5 (not shown here), you decide to enter a rebuttal to the argument. Then, with that frame in view, you would input the command S, calling for the successors of the current item. The response would be to the effect that there are no successors. You could then enter

```
I "A pay cut will keep almost any good person away."
```

The effect could be to insert a successor with the title as specified within the quotes. In order to add accompanying text, you would then issue the command T1 (requesting the text of the newly created first successor). The resulting response would be a message indicating that there is no text. Following this, you enter the command I, which results in a message instructing you to enter the desired text, terminating your entry with an empty line (i.e., two consecutive carriage returns after the final character).

At this point, you might wish to examine some of the other portions of the record. As you progress thru the network of items, the reference numbers of the items examined are pushed onto a stack. The command B (for back-up) pops the top number off the stack, and displays the corresponding item. The effect of n applications of B (for any positive integer n) is obtained by the command Bn. Thus, continuing our example, the command B2 would restore the frame of Figure 5. One might then examine successor 1 or 2 etc.

At some point in the debate over the superintendent's salary, the item shown in Figure 4 may have added to it a third successor as below:

```
3. VOTE ON THE QUESTION (Q).
```

The Q indicates a question to be voted on, and the "ballot" is then displayed (Figure 6) in response to the command Q3. In order to vote no, for example, one would enter VOTE N. Each participant is permitted only one vote, and such matters as to the closing time (if any), whether the vote is secret or roll call and whether intermediate tallies are to be made available are specified, in response to queries by the system, at the time the balloting is opened.

More CBIE commands

Other CBIE commands, not illustrated in the example can be used to

(1) Delete or change items or lines of text or members of successor lists.

```
   BUDGET ITEMS (S, T) #21
1. APPROVAL OF ROUTINE APPROPRIATIONS
2. REPLACEMENT OF OIL BURNER IN HIGH SCHOOL (S, T)
3. ENLARGEMENT OF MIDDLE SCHOOL LIBRARY (S, T)
4. RENEWAL OF CONTRACT WITH TRASH REMOVAL COMPANY (T)
5. SALARY INCREASE FOR SCHOOL SUPERINTENDENT (S, T)
```

Figure 2—Result of S2 command when Figure 1 is on display

```
   SALARY INCREASE FOR SCHOOL SUPERINTENDENT (T, S) #32
1. IT IS PROPOSED THAT, IN ORDER TO KEEP IN LINE WITH
2. SALARIES PAID ELSEWHERE IN THIS AREA BY COMPARABLY
3. SIZED COMMUNITIES, WE INCREASE THE SUPERINTENDENT'S
4. SALARY FROM $30,000 TO SOMEWHERE BETWEEN 35K AND 45K.
5. (SMITHVILLE PAYS 41K, LINDEN 38K AND NEW RIVER 43K.)
```

Figure 3—Example of text (line numbers are to facilitate editing)

(2) Display a predecessor list for the current item.

(3) Display a list of items that have been added or changed since a specified date.

(4) Transmit a message to a specified subset of conference participants. As they next log in to the conference they will be notified that they have a message and can then call for its display. If currently logged in, they will be notified at once. Messages received can be saved if desired.

(5) Post a bulletin which will be displayed to all conferees as soon as they log in or at once if they are already logged in.

(6) Compile a stack of item reference numbers, each of which may be accompanied by a comment. This data can be appropriately displayed and used to call for displays of the items involved.

(7) Call for a list of all commands, with short descriptions, or for a detailed explanation of a specified command.

(8) Specify who may read or modify any item or set of items.

(9) Specify whether the items one is generating should be signed or anonymous.

(10) Change the number of lines in a frame. Where a text or successor list exceeds this size, it will be displayed in appropriately sized pages, each of which, except for the last, terminates with "-more-." A carriage return then commands display of the next page.

(11) Control whether heads should or should not be displayed.

(12) Stop further printing of a display either irrevocably or until a resume command is given.

Still other features have been implemented or are under consideration, but enough has been said to indicate what the system is like. A principal goal is to keep the command structure as simple as possible so as to facilitate use of the system by technologically unsophisticated people, with a minimum amount of instruction.

Other aspects and problems

The problem of security is important. Access to certain conferences, or even to selected portions of conferences must often be restricted to certain sets of individuals. Voting privileges must also be controlled. Where very confidential matters are being discussed (e.g., personnel questions in the Board of Education example) it is important to ensure a high degree of protection against attempts to breach security. The problem is by no means unique to conferencing systems and known techniques employing passwords and the encrypting of stored information are applicable.

Some special problems arise if it is proposed to use computer conferencing for meetings of public bodies such as legislative committees, zoning boards, or boards of education. Care must be taken to avoid blocking the general public out of the process. Several remedies are possible. Since one of the principal advantages of a CBIE conferencing system is that a complete, well structured record of the proceedings is stored at all times, provision should be made to have appropriate printouts made available to the public on a regular basis. Complete records could be made available in public libraries and issued to the press, while summaries (perhaps all titles of items) could be printed in the newspapers. In order to allow direct public participation, terminals open to public use could be situated in post offices, libraries, schools, etc. Public participants might be allowed to read all items not of a confidential nature and to insert questions and comments—that might be appropriately labelled or located to indicate their origins. Finally, in cases where public meetings are *now* being held, there is no reason not to continue that practice—perhaps on a somewhat reduced scale—for the primary purpose of letting the public have its say and to inject into the discussion some of the non-verbal elements missing in a remote system. In the somewhat longer run, it may be that home terminals, installed mainly for other purposes, may become so widespread as to allow private citizens to participate even more directly and conveniently. For this and other reasons it is essential to maintain simplicity in the command structure and compatibility with simple terminal equipment.

SYSTEM IMPLEMENTATION

CBIE has been implemented under the DEC RSTS/E timesharing system and runs on the Columbia University

```
   SALARY INCREASE FOR SCHOOL SUPERINTENDENT (S, T) #32
1. SALARY INCREASE ESSENTIAL TO ATTRACT TOP CANDIDATE(S)
2. HOW MUCH ARE WE TALKING ABOUT? (S)
```

Figure 4—Associated successor list

```
    SALARY INCREASE ESSENTIAL TO ATTRACT TOP CANDIDATE(S) #91
 1. HANSEN'S CURRENT POSITION IN THE PINE WOODS SYSTEM (T)
 2. WHAT WOULD IT TAKE TO GET WILSON? (T)
 3. I DON'T BELIEVE MORE MONEY WILL GET US A BETTER MAN (T)
```

Figure 5—Elaboration of argument in Figure 4.

Computer Center's PDP 11/50 minicomputer. A host system for a conferencing system such as CBIE should exhibit several important characteristics: orientation toward interactive applications, support of file sharing among multiple concurrent users with a granular record locking scheme, and support of a programming language with convenient string manipulation facilities. These requirements are satisfied by RSTS and the version of extended-basic which it supports.

The CBIE processes run as an ordinary user program under the timesharing system. The program is divided into four functional categories: an upper level user interface, a lower level interface which controls user interactions within conferences, command execution modules, and a data management module. The upper level interface is the module invoked on entry to the system. Its function is to request and authenticate the user's identity and determine what he wishes to do. He may wish to perform any of several "housekeeping" functions, such as creation of a conference, seeing a list of conferences in progress, changing conference membership, etc. These functions are also handled by the upper-level interface. In most cases, however, the user wishes to enter a conference and control is passed to the conference interaction controller.

This module accepts and interprets user commands, which fall into three broad categories: item retrieval and display, item modification, and miscellaneous functions. Item retrieval commands include requests to see an item's text, a successor list, or various other information associated with specified items. Item modification commands include requests to edit an item's text, add a link to another item (i.e., insert an element in a successor list), create and possibly link to a new item, change a title, etc. Examples of miscellaneous functions include sending private messages to other users, asking for tutorial information on system features, asking which users are logged on the conference, etc.

Item retrieval operations are handled by a retrieval and display module. Retrieval of information associated with an item is accomplished through communication with a conference data base manager, which extracts the requested information from the conference data base. The item retrieval module may also request portions of items not explicitly requested by the user in anticipation of a forthcoming request. For example, if the user requests a display of the first frame of a given item, the retrieval module asks the data base manager for all the item's text and retains the information in a buffer for subsequent display.

The data-base manager extracts data associated with specific items and incorporates requested changes to them in the data base. Because multiple users must be able to interact with the conference simultaneously, the data-base manager must ensure exclusive access to affected portions of the data-base during up dates. This can be accomplished if the host timesharing system has a record locking mechanism. Due to the logically linked structure of conferences, modifications requested in one item may require changes in related items. For example, a request to remove an item from a successor list also requires a modification to the successor item's predecessor list. Update commands which operate on subnetworks rather than individual items also require exclusive access to many items simultaneously. The possibility of deadlock therefore arises if two or more users simultaneously attempt updates on the conference. The data-base manager must either prevent deadlocks from occurring, or detect and break them.

In a system which allows processes to request exclusive access to resources dynamically during execution, there are essentially two methods of avoiding deadlock. The first is through a hierarchical ordering of resources which constrains the order of requests. In CBIE, determining which items to reserve often requires traversing a subnetwork of the conference during which the required reservations become known. Imposing an ordering on the sequence of reservations would require first traversing the subnetwork to construct an ordering reservation list, followed by a sequence of requests in the allowable order to lock the appropriate records. The second method of avoiding deadlock is through preemption of resources when a deadlock situation occurs. An algorithm is required to detect deadlocks, and an algorithm such as Chamberlin's[14] is required to manage preemptions. The preemption approach in con-

```
VOTE ON THE QUESTION (Q) #173
SHALL THE SALARY OF THE SUPERINTENDENT OF SCHOOLS
BE RAISED TO $40,000 PER YEAR
Y:  YES
N:  NO
A:  ABSTAIN
VOTING WILL CLOSE AT 12:00 11/3/76.
```

Figure 6—Example of a ballot item

junction with a very simple scheme for detecting potential deadlock situations, has been used in the initial version of CBIE.

There are a number of ways in which the data base management module can physically structure the conference data base. The following structure is used in CBIE. Each conference item consists of certain information which is of fixed size. Most of the information is variable in size, however. Space is therefore allocated to items as needed in noncontiguous, linked records. To avoid the need for occasional compaction of storage, which would introduce undesirable delays in the interactive system, fixed size records are used. The penalty is less efficient utilization of space, but this penalty tends to diminish as users add information to items and they grow in size.

One or more linked lists comprise an item, depending on what information is associated with it. These lists are pointed to by a directory record, which also contains all fixed-size information associated with the item. Some of the variable length information (e.g., text or successor list) is also contained in the directory record. Brief items, therefore, may consist of only a single directory record. To reduce the number of I/O operations needed to display a successor list, item titles are stored redundantly, not only in the corresponding item's directory record, but also in the successor list of any predecessor item.

Several types of files are associated with CBIE. There is a global directory of conferences in progress. Each conference has a file of directory records and a file containing the remainder of the linked structure. Also associated with the conference is a file containing member names and other global information. Each conference member owns an individual system-created file which contains a scratch-pad memory and other information needed to provide continuity between the user's sessions in the conference.

CONCLUSIONS

In addition to making meetings involving people who cannot conveniently find common places and times to meet face-to-face more convenient, computer conferencing has certain other unique advantages. Problems often arise in conventional meetings as a result of mismatches among the styles of participants. Those whose minds and tongues operate swiftly must often sit, drumming their fingers impatiently, while others are expressing themselves at a far slower pace. Some enjoy participating in the exploration of tangential issues, or listening to or relating humorous anecdotes that they feel add depth to the discussion—while this may exasperate the more businesslike, "get to the point"-types. A CBIE system makes it possible for each conferee to operate according to his own style and at his own pace. Some may quickly scan the item titles, select the texts they wish to read in detail, make their terse remarks and exit. Others might carefully work thru all that has been said, ponder over each item, enjoy the digressions, perhaps even address some side remarks (in the form of messages) to some particular colleague and carefully compose their own additions. Both extremes can find satisfaction with the same proceedings.

Obviously the elapsed time of a computer conference will be relatively long (some may never actually end, as new topics are occasionally added and old business completed). A compensating aspect is that, unlike a conventional meeting, where topics are dealt with serially, in a CBIE conference, all agenda items whose consideration is not dependent on the outcomes of discussions of as yet unresolved items may be discussed in parallel. Note also that the power of the chairman is greatly reduced. This is because there is no need for rulings on who has the floor. A more subtle point is that, when frequent meetings are inconvenient, chairmen are often assigned interim authority to make certain decisions. The power to call a meeting may indeed fall in this category. Such delegations are less necessary where computer conferencing is used.

People who, because of physical handicaps, find it difficult to get around easily, are obvious beneficiaries of remote conferencing techniques. Those who are hard of hearing or who have speech impediments derive special advantages from computer conferencing.

Reference was made earlier to the ease of producing well organized transcripts and summaries of CBIE conferences. Methods for accomplishing this in a flexible manner are now under development.

There are two modes of perusing a CBIE conference. One is to enter at the head node and then systematically follow selected paths of successors, and the other is to call for the display of specific items identified by their reference numbers. At present, these reference numbers can be on some list compiled by the user from previous excursions thru the network, or they may have been retrieved from the system by a command requesting those items that have been changed or added since some specified date. Techniques for retrieval on the basis of logical functions of author, date and key words in the title are under consideration.

At least one computer conferencing system,[15] which is related to the PLATO system,[11] has some graphics capability, and certainly it would also be desirable to allow speech input. Both of these features will probably be available in future systems.

In addition to the kinds of meetings discussed thus far, there may be great value in using a CBIE type system for educational purposes at all levels ranging from elementary school subjects thru advanced seminars. Unlike conventional teaching machine programs, which rigidly dictate the actions of the student and accept only stereotyped responses from him, a CBIE system can be used to set up structures that allow genuine dialogue between instructor(s) and student, and which allow the student considerable leeway in selecting the order in which he wishes to learn the material and to some extent, what material he wishes to explore in depth. This reflects the basic concept put forward by Theodor Nelson[16,17] whose ideas were influential in motivating the CBIE concept.

ACKNOWLEDGMENTS

The authors wish to thank Professor Christian Gram of Danmarks Tekniske HØjskole for stimulating discussions during the initial stages of this development. Dr. Howard Eskin of the Columbia University Computer Center has been most helpful with advice and cooperation concerning the implementation.

REFERENCES

1. Kuebler and Reid, "A Party-Line for 30," *Bell Laboratories Record,* January 1969, pp. 18-21.
2. "No. 1 Electronic Switching System," *Bell System Technical Journal,* September 1964, complete issue.
3. Gorgas, J. W., "AUTOVON—Switching Network for Global Defense," *Bell Laboratories Record,* April, 1968, pp. 106-111.
4. Reid, A., "Comparison Between Telephone and Face-to-Face Conversation," London Symposium on Human Factors in Tele-Communications, September 1970.
5. Remp, R., "The Efficacy of Electronic Group Meetings," *Policy Sciences,* May, 1974, pp. 101-115.
6. Bedford, M., "Trends in Teleconferencing and Computer Augmented Management Systems," *Proc. IEEE National Conf.,* 1975, pp. 32.20-32.22.
7. Hall, T. W., "Implementation of an Interactive Conference System," *AFIPS Conference Proceedings,* Vol. 38, 1971, pp. 217-229.
8. Turoff, M., "Delphi Conferencing: Computer Based Conferencing with Anonymity," *Technological Forecasting and Social Change,* March, 1974, pp. 159-204.
9. Turoff, M., "Delphi and its potential impact on information systems", *AFIPS Conference Proceedings,* 1971, pp. 317-326.
10. Turoff, M., "Party line and discussion-computerized conference systems", *Proc. Int. Conf. on Computer Communications,* 1972, pp. 161-177.
11. Turoff, et al, "Computer Based Conferencing: A Progress Report," Summary of panel discussion, *Proc. ACM Annual Conf.,* San Diego, November, 1974, pp. 741-742.
12. Englebart, Watson and Norton, "The Augmented Knowledge Workshop," *AFIPS Conference Proceedings,* Vol. 42, 1973, pp. 9-21.
13. Prager, D. A., "A Proposal for a Computer Based Interactive Scientific Community," *CACM,* February 1972, pp. 71-75.
14. Chamberlin, D. D., et al, "A Deadlock-Free Scheme for Resource Locking in a Data Base Environment", *Proc. IFIP Congress,* 1974.
15. Carter, G., Presentation at workshop on Computer Conferencing, National Computer Conference, June 1976.
16. Nelson, T., "No More Teacher's Dirty Looks," *Computer Decisions,* September 1970, pp. 16-23.
17. Nelson, T., "A Conceptual Framework for Man-Machine Everything," *AFIPS Conference Proceedings,* Vol. 42, 1973, pp. M21-26.
18. Smith and Sherwood, "Educational Uses of the PLATO Computer System," *Science,* 192, 23, April 1976, pp. 344-352.
19. Unger, S. H., "Technology to facilitate citizen participation in government", Center for Policy Research Report, Feb. 1972. Talk on same material entitled "Technology for getting people involved", *1973 IEEE Conf. and Systems, Man Cybernetics,* Nov. 5-7, Boston, Mass.

An analytic model for parallel computation

by ROGER M. FIRESTONE

New York University
New York, New York

and

Sperry Univac
Roseville, Minnesota

ABSTRACT

A multiprocessor (MP) system is defined to be two or more independent processing units accessing a common memory for instructions and data. The common memory may be divided into independently accessible banks. A parallel program is a program operating on an MP system which makes use of more than one processing unit to achieve its computational goals.

An analytic model is constructed which exposes certain properties of parallel programs, based on certain idealized assumptions. These assumptions are that:

1. the program algorithm may be divided into independent parallel-executable sections in any way desired;
2. the program is compute-bound, so that input-output considerations do not affect its behavior;
3. there exist linear cost functions for elapsed time and for computer time used;
4. initiation and termination of each independent activity of a parallel program incurs an identical amount of overhead;
5. competition for access to storage causes degradation in performance, manifested as reduced computational speed; and
6. references to storage are independently and uniformly distributed.

Under these assumptions, the model gives an optimal numbeı of processing units to use for the program in order to achieve minimum cost. This optimum is dependent on the overhead, the degradation, and the ratio of the cost of elapsed time to the cost of computer time. In the limit of very large compute time, a bound for the cost ratio is established, below which the use of parallel methods is not economically feasible. This lower bound depends only on the behavior of the degradation factor. Further analysis shows that the degradation factor is linear in the number of processing units employed, inversely linear in the number of independent storage banks, and quadratic in the relative storage accessing rate.

A Monte Carlo model was constructed by J. L. Rosenfeld which gave similar results for those situations where the assumptions of the two models coincided.

INTRODUCTION

A multiprocessor (MP) computing system consists of two or more computational units able to access a common storage unit. Independent computational units ("processors") may thereby share in instruction stream or data area. Examples of such systems are the Sperry Univac® 1100 Series, the Burroughs D825, and the IBM® System 360/67. The number of processors available is usually less than ten.

When a particular computational task is constructed in such a way as to take advantage of instruction and/or data sharing through the use of more than one processor, the task is said to be multiprocessed or programmed in parallel. Since most computations are not constructed in such a way as to be processable in parallel, some program redesign is required to run in parallel, if indeed the underlying algorithm permits such a redesign.

Assuming that an algorithm can be programmed in parallel, the question arises as to whether there is any advantage to be gained relative to a conventional sequential implementation. Secondly, if there is some advantage, it may exist only under certain conditions, and it is then desirable to inquire what these conditions are. They will normally depend on the nature of the problem and the structure of the target system hardware and software. One way of answering this question is to build a simulation model of a typical program, and, by varying the model's parameters, attempt to deduce behavior characteristics which may generalize to other problems. A number of researchers have chosen this method; some of this work is briefly reviewed in the second section. An alternative is the construction of an analytic model of the behavior of an abstract program, making simplifying assumptions where necessary, and then applying the derived relationships to specific cases, hoping that the simplifying assumptions do not destroy the validity of the model. Such an analytic model will be constructed in a later section.

SIMULATION MODELS FOR PARALLEL COMPUTATION

A number of researchers have built simulation models of parallel programs, such as Draughon, *et al.*[1] Lehman,[2] and Rosenfeld.[3] Of these, we shall consider the Rosenfeld model in more detail, as the computer system modeled there more closely resembles the commercially available systems mentioned above.

Rosenfeld modeled the operation of a program to solve the distribution of currents in a resistive network by a procedure of Chazan and Miranker[4] called chaotic relaxation, a close relative of the classical Gauss-Seidel iteration, altered to run in parallel. The simulation was performed on a 7094 using a program called SIMP. The alterable parameters were: an internal program characteristic, the number of processors, and the number of storage modules (independently accessible subunits of storage, later referred to here as "banks").

By varying each of these parameters in a series of experiments, Rosenfeld obtained a number of results, presented as graphs in his article. These may be summarized briefly as follows:

1. Run time decreases as the number of processors used increases, but not as much decrease is observed as would be predicted by a simple inverse proportion law. (Rosenfeld, Figure 6)
2. Total processing time is an increasing function of the number of processors used. (Rosenfeld, Figures 6 and 7)
3. Increasing the number of processors used beyond a certain point produces negligible additional decrease in run time and may in fact increase the run time. (Rosenfeld, Figure 4)
4. Storage interference increases in a roughly linear way with the number of processors used. (Rosenfeld, Figure 8)
5. A measure of quality may be defined which is maximized for one particular choice of the number of processors used. (Rosenfeld, Figure 12)

These conclusions are extremely valuable to planning for parallel computation, as the model is sufficiently general to assume that they apply to other programs. The drawback is that, except for the specific case studied, these conclusions are purely qualitative. Moreover, it is not clear what other characteristics of the program, besides the independent variables already mentioned, will affect the results, and in what way. Thus, if there is an optimum number of processors to use for a problem, Rosenfeld's model gives one no way to compute it, except by his series of experiments.

A model is therefore desired which will indicate the important parameters and allow direct computation for the program behavior. In the following section, such a model will be developed. The concept of optimality will be based explicitly on cost functions, to provide a more concrete representation than Rosenfeld's quality factor.

THE ANALYTIC MODEL

Assumptions underlying the model

We shall now examine a model for the operation of a parallel program. This model will be of an analytic (i.e., statistical) nature rather than of an experimental or simulated construction. Hence, certain restrictive assumptions will be made in order to allow the computations to proceed.

The first assumption to be made is that the program can be divided into independent sections in arbitrarily many ways or at least in N ways, where N will be some sufficiently large number. This assumption is a very strong one, as most known algorithms have a strongly sequential nature. Chaotic relaxation is one of the few algorithms known that satisfies this assumption. It is clear that if a program can be run in N parallel sections, any smaller number will also suffice. Further, we shall assume that the program under consideration is compute-bound or compute-limited; in other words, all time available to the program for execution of instructions will be used by the program to perform useful work. Finally, we shall assume that an indefinitely large number of processing units are available to the program.

Determination of optimal number of processors

Introduce the following notation:

T=elapsed time (may also be used as denotative subscript)
C=compute time (may also be used as subscript)
t=time required for execution of an instruction sequence
s, p=subscripts used to indicate sequential and parallel, respectively
I=time required to create and terminate an independent process
G=cost factor (cost per unit time, with subscript indicating kind of time)
E=cost (time multiplied by cost factor, also subscripted)
d=degradation due to storage interference (compute time will increase by a factor of 1+d)

We assume that there are two cost factors G_T and G_C, corresponding to costs incurred for elapsed time and for compute time, measured in the same units, of course. The objective is to reduce the actual cost of running the program by introducing parallel operation, which is to say we want $E_p<E_s$. For each of these, we have

$$E_s=G_TT_s+G_CC_s$$

$$E_p=G_TT_p+G_CC_p$$

Now,

$$T_s=I+\sum^{n} t_i \qquad (1)$$

and

$$C_s=T_s \tag{2}$$

where we use t_i to indicate the compute time (=elapsed time) of an independent portion of the program, and n is some integer, $n \leq N$. On the other hand,

$$T_p=I+(1+d)\max\{t_i\} \tag{3}$$

and

$$C_p=nI+(1+d)\sum^{n} t_i \tag{4}$$

since T_p will be the length of time it takes the longest subset of the program to execute (where t_i must be multiplied by $1+d$ to account for storage competition among the various activities), and C_p must be augmented by the storage interference value and also by the time necessary to initiate and terminate the extra processes. Since C_p is strictly greater than C_s, reduction of E_p must come from a decrease of T_p. Therefore, we shall seek to minimize T_p.

We may assume, without loss of generality, that $t_1 \geq t_2 \geq \cdots \geq t_n \geq 0$. Let $T_e=\Sigma t_i$. It is then clear that T_p is minimized if $t_1=t_2=\cdots=t_n=T_e/n$. We then have, for all n,

$$E_p=G_T[I+(1+d)T_e/n]+G_C[nI+(1+d)T_e] \tag{5}$$

We shall now assume that $d=an+b$, where $|b|<1$, $a>0$. This assumption will be supported later. Let us temporarily allow n to be a continuous real variable, $n>0$, instead of a positive integer. Then $E_p(n)$ is convex for $n>0$ and thus has a minimum at n_0, where

$$n_0^2=(G_T/G_C)(1+b)T_e/(I+aT_e) \tag{6}$$

and the minimum for E_p with integral n is either $[n_0]$ or $[n_0]+1$, where $[x]$ denotes the greatest integer not exceeding x. What is significant about this result is that the values of G_T and G_C are no longer important; rather, the quantity that appears is the ratio of the two cost factors, as one might suspect should be the case. Moreover, we now have an expression for the optimum number of processors to use on a problem. It is interesting to observe that n_0 has a limit as $T_e \to \infty$, which indicates that for given cost factors, there is a maximum number of processors that can be used effectively. This limiting value is developed in the following section.

In order to make parallel processing useful on a given task, we must have $E_p<E_s$. We now assume that $n>1$ to exclude the sequential case. Then we have

$$\begin{aligned}E_s-E_p&=G_T(T_e-(1+an+b)(T_e/n))\\&\quad+G_C[(1-n)I-(an+b)T_e]\\&=G_T[(n-1-an-b)/n]T_e\\&\quad+G_C[(1-n)I-(an+b)T_e]\end{aligned} \tag{7}$$

which we shall require to be positive. We next assume that

$$n \geq 1+an+b$$

which, as noted, will be discussed further later. Then $E_s-E_p>0$ becomes

$$G_T>G_C n[(an+b)T_e+(n-1)I]/(n-1-an-b)T_e \tag{8}$$

which must be true for parallel execution to be less expensive than sequential. The n on the right hand side may be taken as n_0, which is the integer value of n minimizing E_p.

It may already have become apparent that b should be equal to $-a$ for the degradation value to be zero when $n=1$. Using this fact, we shall examine the results of the previous calculations as $T_e \to \infty$.

$$\lim n_0^2=(G_T/G_C)(1+b)/a=(G_T/G_C)(1-a)/a \tag{9}$$

or

$$G_T/G_C>n(an-a)/(n-1-an-a)=an/(1-a) \text{ as } T_e \to \infty$$

Letting $n=n_0$ in the second relation, we have, using (9)

$$\begin{aligned}G_T/G_C&>(G_T/G_C)^{1/2}(1-a/a)^{1/2}(a/(1-a))\\&=(G_T/G_C)^{1/2}(a/(1-a))^{1/2}\end{aligned} \tag{10}$$

or

$$(G_T/G_C)>a/(1-a) \tag{11}$$

Since the right-hand side of inequality (8) is decreasing in T_e, if this relation (11) is not satisfied, then (8) cannot be satisfied for any finite value of T_e. Thus we have a necessary condition for parallel processing to result in a lower cost than sequential computation. For a sufficient condition, the more complex relationships (8) and (6) must be used.

Analysis of storage interference

It is now necessary, as promised, to examine the calculation of d. Let us begin by defining two storage areas as independent if accesses (reads or writes) can be made simultaneously to both. The alternative is dependent storage, where, if two accesses are made simultaneously, one must wait until the other has completed its data transfer. Independent storage areas will be referred to as *banks*. Two storage areas within a single bank are, of course, dependent. (Industry practice for binary computers is to use certain bits of a memory address, perhaps several high-order and a few low-order, to select the appropriate bank, while the remainder of the bits specify the word within the bank. Thus, the number of banks is a power of 2; also the desirable feature obtained is that words whose address differs greatly are in distinct banks, and consecutive words are also in different banks.)

Suppose j accesses to a single bank are attempted. The first is made immediately; the second waits one unit of time for the first to complete before it is granted; the third waits two time units for the first and second; and so on. (In general, the accesses are granted in an arbitrary order, so the terms "first" and so on are not meant to imply any ordering process performed on the access requests before any one is granted.) The total delay in this situation is $j(j-1)/2$.

Let us assume that a storage reference is made on every storage cycle by each of the processors operating on the program. This assumption will not, in general, be warranted, so it will be removed later on. Let k be the number of banks; n will continue to be the number of processors operating. The degradation value d will then be the average total delay divided by n, which is the average delay per processor. Execution thus takes longer by a factor of (1+d).

Omitting some tedious calculations in which the average total delay for each storage reference assignment is summed over all possible combinations of storage assignments and then averaged again, the value of d as a function of n and k is given by

$$d(n; k)=(n-1)/2k$$

Notice that d has the desired form $an-a$, where $a=1/2k$.

Now we introduce q, the probability that a storage reference is made by a processor on a particular cycle and drop the previous assumption that $q=1$. The degradation factor may then be recomputed as a function of n, k, and q by a similar lengthy set of calculations, giving the final result

$$d(n; k; q)=q^2(n-1)/2k$$

As before, this has the form $d=an-a$, and the analysis is complete.

Justification of assumptions

We must now examine this model to determine the extent of its correspondence with the real world. More precisely, a number of assumptions have been made which should briefly be examined for validity. One of these assumptions dealt with the number of processors available to the program. Although we assumed an indefinitely large number were available, the actual maximum number required is $[n_0]$ or $[n_0]+1$. If fewer than this number are available, maximum possible cost decrease would be obtained by using the maximum number of processors available, so no severe problem arises. In any case, values for n_0 are not likely to become too large. The expression for n_0 in (6) is increasing in T_e with the limiting value given in (9). Since, for a reasonable value of $a=q^2/2k$ given by $q=.5$, $k=8$, we have

$$n_0{}^2=63(G_T/G_C)$$

if compute time is only five to ten times as expensive as elapsed time, n_0 will fall somewhere between 2 and 4. This is well within the number of processors provided on currently manufactured MP hardware. Nevertheless, there are situations where elapsed time may be assigned costs equal to or even greater than those for compute time, such as real time operations. An example might be weather prediction, when it is clearly desirable to produce a 24 hour forecast in less than 24 hours. In such cases, presently available hardware cannot reach n_0, and programs cannot be run at minimal cost. Generally, the costs involved for elapsed time in these latter cases are of a different nature, where the results of the program become worthless after a certain moment. The cost function is thus no longer linear (nor necessarily continuous), and one of the other assumptions is thus violated.

The other assumptions made deal more with the type of the program to be run. We have already seen that the results obtained so far are meaningless without the assumption of a linear cost function. This assumption is reasonable in light of the general use of linear cost functions by computer center managements, as well as by businesses in general. As far as the restrictions to compute-limited programs is concerned, such programs do exist and are frequently those with largest T_e as well. (For a discussion of the types of FORTRAN programs being written, see Knuth[5].) Whether or not a program may be divided into arbitrarily many parallel segments, it will still be possible to obtain a minimum feasible cost by selecting a permissible number of processors as mentioned above. The integer programming problem is easy to solve, as we have noted in the discussion of the behavior of E_p as a function of n. An example of this situation might be a loop of M iterations, each iteration being independent of the others. The possibility of separating the problem into parallel tasks may depend on the factorization of M if strict equality is to be maintained among all individual parallel tasks; however, if the strict equality is relaxed, it is clear that n processors can be assigned in such a way that each processor executes $[M/n]$ or $[M/n]+1$ iterations, which should be close enough.

The assumptions dealing with storage accessing and interference are more difficult to justify, particularly those dealing with uniform random distribution of accesses in time and among banks. One possibility is that enough factors such as differing instruction execution times, background activity, and so on would produce the necessary randomness, while any deviation from a uniform random distribution would be more orderly and should therefore produce less degradation. It is of course true that without this assumption the above analysis would be impossible. Note, however, that it is not necessary to require that the access pattern for any individual processor be random (this is usually not the case, of course, as computer addressing designers are aware), but only that the overall pattern is randomized such that each access assignment is equally probable.

A final criticism of this model is that the storage interference analysis should proceed by Markov chain methods, considering the system as a finite population queueing system (with population=n, servers=k). It is hard to answer this criticism directly without completing the analysis and comparing the results. The Markovian analysis may be found in the work of Baskett and Smith,[6] but the results derived there are not in a form which permits easy comparison with the present approach. In general, it may be said that the Markovian analysis will provide greater detail of the statistical behavior of the system (such as standard deviation, fluctuation magnitude, and so on). In the system described here, however, the real concern is a long-period time average, since that is the significant observable parameter. It remains the author's contention that the simpler analysis presented here is adequate to calculate this long-period average under the assumptions made for this model, and the correspondence to the results of Rosenfeld's model are significant evidence to support this contention.

COMPARISON OF MODELS

Before assessing the similarities between the simulation model of Rosenfeld and the analytic model presented here, it is proper to mention the two main differences. First is the introduction of cost functions in the analytic model. The effect of these in the simulation model is hidden, but the implicit assumption made in that paper is that the cost of processing time and the cost of elapsed time are identical. The other difference is that the program used by Rosenfeld contained a critical section or "lockable block" of instructions which could only be executed by one processor at a time. If e denotes the fraction of the code which is in the critical section, then when $ne>1$, at least one processor will always be waiting to enter the critical section. The effects of this are manifest as non-linear portions of Rosenfeld's graphs for large n.

Reviewing the five results of Rosenfeld's work described before, it is evident that each of them also applies to the analytic model. In particular, the overhead value accounts for the variation from the inverse proportion rule for run time as a function of number of processors, while storage degradation explains the increase in run time. Further, the storage degradation behaves linearly in the analytic model as it did for Rosenfeld's model, with similar slopes if q is approximately 1. Finally, the value of n which maximizes the quality factor used by Rosenfeld may be determined from the analytic model to be

$$(1-a)T_e/(I+aT_e).$$

Of course, this will be the same as n_0 (the minimum cost value) for only one choice of the cost ratio, but the behavior of the quality factor as a function of n is similar for both models.

CONCLUSIONS

The disadvantage of an analytic model is mostly the simplifying assumptions made in order to allow the calculations to proceed, which may then produce an invalid model through lack of conformance to reality. By examining the assumptions carefully and by comparison with a simulation model, it is hoped that the validity of the analytic model can be established. A valid analytic model may then have significant consequences for future hardware and software design.

In particular, the model developed here indicates, firstly, conditions under which parallel computation can produce benefits in terms of reduced cost. This depends on the hardware structure (number of banks and number of processors available), the operating system (overhead), the program characteristics (compute time required and rate of storage access), and the environment (cost ratio). Further the exact amount of improvement that can be made by varying each of these can be calculated. Hardware and software designers may take advantage of this knowledge by designing their systems more efficiently. Since storage degradation is the limiting factor for the cost ratio when T_e is large, as shown by (11), greatest benefits can be obtained by making $a=q^2/2k$ as small as possible. Since a is quadratic in q but only (inversely) linear in k, greater benefits can be obtained by reducing q than by increasing k by a like factor. This means that by providing storage local to each processor, such as buffer memories or scratchpad registers, thereby reducing q by eliminating the need for temporary storage locations or reducing the need to access shared main memory, performance can be improved as much or more than by the alternative of increasing the number of independent storage banks.

The model also has consequences for compiler designers who wish to produce parallel code automatically from serially written source language. All other things being equal, the analytic model shows cost to be a decreasing function of T_e. This means that cost is minimized by converting to parallel that portion of the program which takes longest to execute. Thus, while optimization techniques are most productive when applied to the innermost loop, parallelization is best applied to the outermost loop, which will take the longest to execute.

In short, the analytic model presented here not only costs less to construct and use than a simulation, it also provides wider applicability, more precise results, greater insight, and more specific guidelines for future development.

ACKNOWLEDGMENTS

Part of this research was performed while the author was a National Science Foundation Graduate Fellow at New York University-Courant Institute of Mathematical Sciences. Computer time was provided by Sperry Univac Division of the Sperry Rand Corporation. I thank Professors Malcolm C. Harrison and Jacob T. Schwartz for their advice and assistance.

REFERENCES

1. Draughon, E., R. Grishman, J. Schwartz, and A. Stein, *Programming Considerations for Parallel Computers*, Courant Institute of Mathematical Sciences Report IMM 362. New York University, New York, New York.
2. Lehman, M., "A Survey of Problems and Preliminary Results Concerning Parallel Processing and Parallel Processors," *Proceedings IEEE*, Volume 54, Number 12, December 1966, pp. 1889-1901.
3. Rosenfeld, J. L., "A Case Study in Programming for Parallel Processors," *Communications of the ACM*, Volume 12, Number 12, December 1969, pp. 645-655.
4. Chazan, D. and W. Miranker, *Chaotic Relaxation*, IBM Research Report RC-1976, Thomas J. Watson Research Center, Yorktown Heights, New York, January, 1968.
5. Knuth, D. E., *An Empirical Study of FORTRAN Programs*, Stanford University Computer Sciences Department Report CS-186, Stanford, California.
6. Baskett, F. and A. J. Smith, "Interference in Multiprocessor Computer

Systems With Interleaved Memory," *Communications of the ACM*, Volume 19, Number 6, June 1976, pp. 327-334.
Note:
This paper contains a comprehensive bibliography of literature dealing with investigations of storage interference.

APPENDIX: COMPUTATIONAL DETAILS

It is the purpose of this appendix to expound the details of the computation of the degradation factor d, omitted from the main body of the paper.

As noted previously, the average delay per bank was $j(j-1)/2$. In this model, there are k banks with n processors operating. The degradation factor is computed as the average total delay divided by n, which is the average delay per processor. Execution will thus take longer by a factor of $1+d$.

How many different ways are there of making n accesses to k storage banks? The answer is k^n, for there are k slots for each of the n accesses, and each access may be assigned individually, independent of all the other choices. This introduces a factor k_{-n} to compute the average delay over all possible access assignments.

An access assignment may be described by the k-tuple of non-negative integers $(n_1, n_2, \ldots, n_k)$ where $n_1+n_2+\cdots+n_k=n$. This means that n_1 processors are attempting to access bank 1, n_2 access bank 2, and so on. The number of access assignments corresponding to a single k-tuple is given by the multinomial coefficient which we denote

$$(n|n_1, \ldots, n_k)=n!/\prod_{i=1}^{k} n_i!$$

where

$$\sum n_i=n$$

This is so because there are n processors to be assigned with ordering unimportant. The delay resulting from an access assignment to a k-tuple $(n_1, \ldots, n_k)$ is given by

$$\sum n_i(n_i-1)/2$$

which we obtain by summing the delay for each individual bank.

Combining these results, summing over all possible k-tuples, and dividing by the factors mentioned, we obtain

$$d(n; k)=(k^{-n}/n) \sum [(n|n_1, \ldots, n_k) \sum n_i(n_i-1)/2]$$

There are a number of things we can do to simplify this formidable expression. First we note that the sum of delays behaves rather nicely by being symmetric in the variables n_i. All values for any one n_i are taken by all the others in a different order. Hence, we can replace this sum by k times one of its terms, obtaining

$$d(n; k)=(k^{-n}/2n) \sum (n|n_1, \ldots, n_k)n_1(n_1-1)$$

Next, we introduce the following generating function

$$g_0(x_1, \ldots, x_k)=(x_1+\cdots+x_k)^n$$
$$= \sum (n|n_1, \ldots, n_k)x_1^{n_1} \ldots x_k^{n_k}$$

Differentiating both sides of this twice with respect to x_1, holding all other variables fixed, gives

$$n(n-1)(x_1+\ldots+x_k)^n$$
$$= \sum n_1(n_1-1)(n/n_1, \ldots, n_k)x_1^{n_1-2} \ldots x_k^{n_k}$$

Now, let $x_1=x_2=\ldots=x_k= 1$ on both sides, giving

$$n(n-1)k^{n-2} = \sum (n/n_1, \ldots, n_k)\, n_1(n_1-1)$$

Substituting this into the previous expression gives the result

$$d(n; k) = (k^{-n}/2n)kn(n-1)k^{n-2} = (n-1)/2k$$

This is the result specified in the text.

The above procedure can be followed for the computation of $d(n; k; q)$, but each access assignment must be multiplied by the probability of its occurrence, which is given by the usual binomial coefficient. The first reduction gives

$$d(n; k; q) = (1/2\ nk) \sum {}_nC_m\, q^m(1-q)^{n-m}\, m(m-1)$$

This is summed using the generating function $f(x,y) = (x+y)^n$, which is differentiated twice in x, multiplied by x^2, and then evaluated with $x=q$, $y=1-q$. This produces the final result

$$d(n; k; q) = q^2(n-1)/2k$$

as discussed in the text.

Dominance relations in computing systems

by DANIEL G. HAYS
University of Alabama in Huntsville
Huntsville, Alabama

ABSTRACT

This paper discusses various kinds of dominance relations that may hold among parts of computing systems, and between computers and people. That computers enter into "social relations" seems clear. Four models prevalent in thinking about computers are discussed: the Master-Slave, the Egalitarian Workgroup, the Division of Labor, and the Clamoring Children models. To help explicate the claim that "dominance" in a computing system may vary both in type and over time, steps towards a formal treatment of interacting devices are suggested, involving possible sequences of influenced behavior. Several varieties of dominance are then outlined, such as competition for resources, or attentional dominance for interrupts. Finally the matter of computer intentions is noted.

INTRODUCTION

This paper inquires into kinds of social relations which may hold among parts of computing systems, including some cases where people are involved. While no assumption is made that computing machines have feelings, minds, a sense of group spirit, jealousy, love, or other attributes of social animals such as ourselves, they do interact. That is, the output of one device serves as input to another device, and makes some difference in the behavior of the second device; device two may return a message; and so on. Indeed, computers function more and more in social contexts: among themselves, with people, and with effector devices such as tail flaps and milling machines. The growth of computer networks, where a number of fully-functioning modules and communication devices make requests of one another and do some useful work, has been pronounced over the past several years.[1] As networks increase in size and as advances in programming allow more flexible handling of requests, the kind of 'computer sociology' envisaged in this paper should become important as an area of study.

The class of relations to be examined is *dominance* of one part of a system over another. One or another kind of dominance is widely found in groups in the animal kingdom,[2-4] and may be essential to understanding such arrangements in general. In the discussion to follow, dominance will be treated more as "causal effectiveness" than in terms of the trappings of human power such as the wearing of crowns or having a title on the door. (Only people, to date, wear crowns; both people and computers may have titles, carpets, etc.) Exactly what dominance can mean in the context of the moment-to-moment behavior of a system is a major question to be addressed. One proposition that will be developed is that the answer to the question, "Who (or what) is in charge here?" is not an all-or-nothing matter, and may change from time to time, for both computing devices and people.

Thus although in some cases dominance, once gained, may perpetuate itself, turning into absolute and enduring power, this will not always be the case; and fears that computers may come to totally dominate people may be as unconsidered as the comfortable belief that people are presently the absolute masters of computers, with the latter holding no effective demands on the former.

The discussion to follow will be informal, though some steps towards a formal treatment of dominance relations among devices will be sketched in the third section. The second section will examine several common views of computer relations as social phenomena. The fourth section will distinguish several kinds of dominance, mostly applying to interacting devices. Finally, several senses in which machines might dominate people, some quite mundane, will be noted.

STANDARD VIEWS OF COMPUTER INTERACTION

Though probably most laymen and virtually all computer professionals would deny that computers are sentient,[5] it is easy to think of them as analogous to humans. Correspondences come to mind so readily that one suspects there may be something to them. Certainly the apparent similarities should not be dismissed summarily.

To impute intentions or feelings to a computer would seem to require more of a leap of faith about things unknown than to attribute social relations to them. Questions of computer intelligence, will, etc., have attracted more attention,[6] but people also speak of computers in social terms, with much clearer basis in what can be observed.

At least four underlying models, or typical pictures of computer interaction seem to be prevalent. They are informal, almost metaphorical, but they seem to capture essential similarities between some human work groups and arrangements among devices. Only one, the Master-Slave model, is widely held and really explicit. The other are more implicit to discussions among workers in the field.

The master-slave model

The image of computers as conceived by the public frequently involves a Master-Slave relationship. This model probably also represents a substantial part of basic professional regard of appropriate machine role (at least when the professional computer person is not engaged in trying to get something done with a computer, in which case more of a give-and-take situation probably holds).

Usually it is assumed that people are the Masters, but they may be Slaves to the computer in fearful fantasies.

This model may also apply to cases where only devices are interacting. One computer, or central processor, or supervisory program, is seen as being comprehensively in charge, demanding instant obedience, serving as the sole source of permissions, and in some instances making decisions about device status, inclusion in the current configuration, and so on.

The control may be lenient, when the metaphorical Despot is Benevolent, or may be structured for close and frequent supervision. (A summary of research on consequences of frequency and detail of supervision in human groups is contained in Reference 7, which may be suggestive for supervisory schedules in computers.) In either case, one device or creature is viewed as being in charge, and the main duty of others in the system is obedience.

In popular lore, the Master-Slave imagery is cast in extreme terms. Computers are regarded as Super-Slaves, wholly controlled by technical persons, capable of producing magical and powerful results. But conversely, the insecure edge of the fantasy is revealed by fear that the computer will gain total control, perhaps deviously, over its former masters. One may only speculate about deep psychological reasons for the apparent wide appeal of this attribution. It is a plausible but untested possibility that persons high on the trait of Authoritarianism[8] could be especially prone to such fantasies. Or the fantasies may be related to beliefs about how much a person's behavior is under his own control as contrasted with control from external factors.[9] Perhaps also the fantasies of computer control represent either fears or a deep realization of the extent of the control of individuals by governmental, corporate, and other social constraints. Or a strong desire to control computers as contrasted with using them to get something done could represent a failure in other relationships.

Whatever irrational components this model might be associated with in some individuals, there may be practical difficulties in its exclusive use. If a person who is designing an interacting system thinks only in terms of a rigidly hierarchical model, his or her design choices may be limited. Thus what people think of computers may not be just an interesting sidelight in the folklore of technology but may have practical consequences.

Though some hierarchical arrangement for attention-getting and compliance with requests may be desirable in computing systems, the amount of centralization of function may present a number of design choices if the system is at all elaborate, multiply redundant, or dispersed. Other choices may have to do with the closeness of supervision, or the choice of having a central processing unit dictate a specific program in contrast with its communicating some requirements to a subordinate unit and leaving that unit to produce acceptable results by means of its choice.

Certainly hierarchy and strict control may be advantageous in simple systems and in ones where there is very great risk of zonking out; but other models are based on more cooperative premises.

Egalitarian task-group model

In this pattern, a number of processors share the work to be done, and there is no linear dominance ordering except possibly a transitory one—for example, based on a first-come first-served rule. Another basis is fitting the tasks to be done to the units or configurations that can best handle the work (see Division of Labor Model, below). In the extreme case of this model, any processor may request activities of any other processor.

The underlying human analogy is that of a work-group of peers. A political formulation is "From each according to his abilities; to each according to his needs."

The conceptual key here is *equality of units*. Equality can be subsumed in a hierarchical system if there are a number of units on one level which have no more than transitory precedence for involvement in tasks. An ordinary multiprocessor system works in this way, with a task evaluator and scheduler sitting on top as a Honcho Unit. When a processor becomes "dedicated" across possible programs and moments of time to a certain source or to tasks with some arbitrary attribute, then the situation becomes less than egalitarian.

As in the Master-Slave model, the Egalitarian Workgroup image may involve devices only, or may include persons. Since people and computing machines differ so drastically in what they are good at doing, how quickly they do it, and their ready means of communication, this aspect is perhaps more relevant to the Division of Labor model.

In popular lore a view of the computer as partner has emerged over the past few years. Especially in children's programs on television, robots may work together with humans. Although they are depicted as having special abilities, these ambulatory computers are generally friendly. Computer professionals must find the capabilities and human-like intuition and foibles of these devices astounding indeed, but their depiction seems to indicate some easing of the tensions that people feel toward computing devices.

On the professional side, a person can certainly regard a computing device as a partner in problem solution. It is a question for empirical research whether attitudes of equality, contempt, awe, or indifference have some effect or no effect on quality of machine use.

Division of labor model

Though the notion of task specialization of a unit was invoked in discussing equal status arrangements, it can apply as well in a Master-Slave model. Perhaps the clearest case of Division of Labor thinking applies to designs where the machines are structured for certain classes of activity, for example by special sensor or effector hardware, by resident program sequences, or by general characteristics such as word size or memory depth that are optimal for special purposes.

Some version of this model is as old as electronic computing, with the divisions of input, output, and central computation. The proliferation of program sequencing and test-and-branch logic over many parts of a computer system, some quite specialized, is taken for granted today; and has served to blur the distinction between machine specialization and specialization due to a particular program.

Still, in thinking of devices, there does seem to be a difference between a machine that is specially constrained for one kind of processing and a more general purpose machine that is arbitrarily assigned that processing. At the one extreme is a mechanical hand, for example, scuttling across a radioactive area, with specially shaped "fingers" but little logic of its own. At the other extreme is the Jack-of-All-Trades Processor. Somewhere in between would be the chunk of logic and memory at a remote batch entry site, or controlling details of disc storage and retrieval.

There appears to be two underlying motivations for task specialization. One is efficiency and cost; the other is convenience. Efficiency and cost apply both to device construction and to the choice of alternate devices—where there is a choice—to process more programs appropriately. In cases where efficiency is problematical (as it may in fact often be) the motivation of convenience suffices. When devices are assigned on purely arbitrary grounds, only convenience is involved.

Clamoring children model

In this model, a number of physical units or potentially active programs seek the attention of a central control source and clamor for the occupation of systems locuses. The central source emulates the behavior of some kind of Parent, or Child-Care Unit. This is the familiar model of various multiprocessing systems.

The clamoring of the "children" components may be insistent. The Parent configuration may adopt various strategies of care for the stated needs. (Though a fair-sized technical literature exists on this subject, the underlying analogy does not seem to have been enunciated.) Generally the behavior of the Parent system is post-Dr. Spock, in that attentiveness and a certain amount of permissiveness seem to be important criteria. Generally it is assumed that the Parent will devote maximum time and resources to the behavior of the children components unless some pathological condition exists. Still, insensitivities to the actual needs of the children, and a lack of understanding may be found. It is as if the Parent were saying, "I don't know what it is you're doing, or exactly what you'll need, but you can have 10 milliseconds to do it in." After some period of lenience, the Parent may unceremoniously toss the Child out of the system.

A number of dimensions may be involved in giving an adequate description of the relations in this model. Attention-getting in the sense of interrupts, occupation of processor time, channel use, resource assignment are just a few. What is interesting is that in this very common kind of system the relations are far from the Master-Slave paradigm that seems so readily to describe computer-based interaction in the lore of both public and professional. Certainly the Parent aspects of the system are "in control" and have decisional evaluation and choice. Yet the Clamor Units are heavily involved in the determination of the actual sequences of events which will be realized.

DOMINANCE: BASIC CONSIDERATIONS

In the four informal models discussed above, dominance of one or another kind seemed very much at issue, though was fairly implicit. In this section some suggestions will be sketched for a more precise treatment of interactive computing machine behavior, including varieties of dominance.

The arena in which dominance activities of one or another kind take place may be variously viewed as a world of devices, logical or functional units, or processes or procedures. Ordinarily one would think of either device or program as having some *boundary* based on coherent criteria, whether (a) physical and communicational in that events within the unit are vastly more available than they are outside the unit, (b) causal, where separate events are in their minute connections determined vastly more from within than from without, or (c) logical, where there is some symbolic pattern belonging to the subsystem that distinguishes it from others.

A unit or process exhibits *activity* or *behavior*. It is assumed that it is possible to identify discrete behavioral units. (A more general assumption would not require discreteness, or would associate a discrete representation of symbols with some underlying continuity.) The activities or behaviors may be described in terms of rules of composition which may provide activities composed of more basic activities, and specify event structure. These rules characterize the well-formedness of events in the system; but they are not the same as the underlying processes which may be thought of as capable of producing the behaviors in a strong sense.

Activities are *realized* in the actual processing of the system. In considering what might happen in the system, we are looking at it in a temporal framework and projecting possible sequences of behavior for various parts of the system. The logician would say that we are quantifying over moments of time, and possible states of each processor device, perhaps also over programs.

A device A is said to exhibit *socially effective action* r on device or process B if ensuing behavior s of B is different than it would have been, considering possible sequences of behavior, if A had not exhibited r. In practice, this is not generally a problem to identify. It ordinarily involves a clear channel, and it is usually direct social action with no intermediary process.

The *scope* of an action r of A on the behavior of B is not so readily defined, even in these informal terms, but it is an important notion. Considering just A and B, the scope set S' of r of A on B is the set of sequences of B's behaviors that are distinctly associated with A's having performed r. The key notions involved are (a) what the sequences are, (b) the length of the sequences, and possibly (c) some internal measure for the sequences in the scope set.

The *actual scope* is that sequence of affected behaviors of B that is realized, if any. "Scope", thus defined, is open to complications if more than one process may affect B's behavior or if remote conditions in other sequences of A or B might affect the relevant behavior. Even if one assumes that there are 'program-limited' bounds to the calculation of sequences, certainly a reasonable assumption for most systems, the possibility of effective multiple interacting processes presents problems in analysis, perhaps not insoluble ones. At the worst, formal analysis along the lines suggested here might uncover deep problems with the assessment of multiple causality. However it may be possible to make definite certain classes of interactive influence, perhaps to systematically limit the depth of causal influence among devices, and come up with some useful results. For present purposes, the notions are important that some behaviors have more or less impact on the behavior of other processes; that the behaviors may be directly effective or indirectly effective through 'ricochetting' effects on the behavior of intermediate units; and that the space of possible activity sequences is the field in which analysis takes place.

Dominance by any definition should involve socially effective behavior, but it is at once more restrictive and more complex. Either repeated social action, or interaction (two-way) could be involved.

Dominance effects of behavior a may be said to be *behavior-limited* to the effects on the other of that behavior. If dominance is viewed as stemming from a behavior d of A, then A is *behavior-dominant* or B-dominant. If dominance is limited to effects of a program or program segment, then it is *(program)* P-dominant for that segment.

Dominance could be of several types, such as those identified on content and intuitive bases in a later section. Suppose the types are α_1 through α_k; then dominance would be further qualified as

$$\text{B-dominant, type } \alpha_i\text{, or (B, } \alpha_i\text{)-dominant; or} \qquad (1)$$

$$\text{P-dominant, type } \alpha_i\text{, or (P, } \alpha_i\text{)-dominant.} \qquad (2)$$

For example, one device might dominate another for short tasks but not for long tasks, or be able to request disc-access computations but not arithmetic sequences in general.

If dominance extends beyond a behavior-scope or a program, there is required some measure of the dominance over possible programs, or over actual realized programs. With the provision that it is clear whether possibility or actuality is being referred to, suitable measure functions could be constructed,

$$\mu(A\ \alpha_i\text{-dominates } B), \qquad (3)$$

$$\mu(A\ \alpha_i\text{-dominates } B\text{; on condition } q). \qquad (4)$$

so that for instance μ would be a relative frequency of domination in relevant occasions.

Just generally it is not clear that dominance would be a transitive relation. It is conceivable that, if dominance of type x depended on the consequent events of B from something A did, dominance of the same type would involve establishing that the same could happen from B to C. Still, those behavior/effect sequences might not hold from A to C, because of special social arrangement or hardware characteristics.

If A α_i-dominates B over all situations, then A is said to *totally* α_i*-dominate* B. Domination over all types would amount to total domination. Total dominance would describe a partial order of devices.

Though it has been assumed that dominance is *based on behavior* of the dominating device or process and the effects on the dominated party, some kinds of dominance may be based instead on *definition* and symbolic acceptance. In practice, definitional acceptance may lie behind the behavioral impact of A. For example, if a unit is labelled in an appropriate entry in a descriptor table as a VSOP (for very special operations computer), and that labelling is used to facilitate requests for input-output service, then label is translated quickly into behavioral preference. Thus, a special kind of dominance would be the power of a device to insure the acceptance of a dominance definition whenever it was transmitted to another device. Note that the actual behavior need not actually occur, and there need be no other justification than the labelling itself.

DOMINANCE TYPES

Several kinds of dominance which may be conceived as holding among parts of a computer system—or person-computer system—will now be described informally.

1. *Attentional dominance* refers to the gaining of the attention of a unit by a signal sent to that unit, perhaps indirectly. Ordinarily this is regarded as a short-term activity, a prelude to further processing. The first unit says, in effect, "Halloo there," and the receiving unit acknowledges with what might be called a Hark!-act. The common case is an interrupt. Attentional dominance is transitory unless there is some priority system for acknowledging signals involved. The Halloo-ing unit might be said to

dominate the Hark unit upon successful completion of the behavior cycle; or it might be said to dominate other competing units (but see below for competition dominance).

2. *Channel dominance* is similar to attentional dominance, and may be involved in it. It has to do with superior means of accessing a communication channel.

3. *Decision-making dominance* means that one unit has the programming necessary to evaluate information leading to the choice of one of a number of possible act-sequences; and the other unit does not. As in the discussion of scope limitation above, decision-making may be perfectly general (total decision-making dominance), it may be limited to specific decisions or specific classes of decisions. Since most computational units have test-and-branch provisions, decisional dominance would be almost always distributed in detail. However, since the function is so important, provision for making one or another kind of decision is by no means trivial, and it is probably the case that at least systems decisions are highly centralized in present-day configurations.

As with human decision makers in organizations, facets of the decision making process in computing devices may be examined more finely for their distribution over units. The *amount and kind of input to decisions* over an interacting system could be identified and made quantitative. An especially important facet of human and machine decision-making is the *sources of the alternatives* to be evaluated. Other kinds of input are *informational input* and *evaluational participation,* including voting arrangements.

4. *Resource competition dominance* involves another very widespread function in computer-based systems, and should usually depend on some kind of decisional evaluation. Dominance is construed as over other competing devices or processes. Both amount of resource use, and timeliness of use with respect to the requisites of the using device apply.

5. *Request/compliance dominance* overlaps resource competition dominance, in that most competition involves explicit requests, and the outcome is mediated by a scarce compliance. It is more general, however. Request dominance is conventionally defined from requesting to potentially complying unit. As in resource dominance, decisional evaluations (the alternatives being at least "grant this request" and "don't grant this request") are generally involved. The evaluations may incorporate a diversity of forms, and draw on a variety of information. Or, little information may be evaluated, for example when treatment is based on arbitrary grounds such as unit identity alone. Rejection or acceptance may be limited to some behavior class for a given unit, also. This evaluation would involve only the identification of class of behavior and unit identification.

If a decision making unit which evaluates requests is especially restrictive, it may be useful to speak of *permission dominance,* or *balk power.* Permission dominance is more general than balk power. Both qualify as behavior-based dominance ploys since they are likely to have notable effects on the behavior of the requesting unit.

A series of decisions may be involved in a request. For example one unit may approve the worthiness of a request for storage usage, but the unit monitoring the storage device may refuse the request temporarily because of current space usage. Decisions may be not only sequential but also distributed; and it is always possible for one part of the system to have information that is inconsistent with another part of a system.

6. *Outcome override and other illegitimate dominance.* Decisions may be evaluated and still overridden by a suitably programmed unit. Other violations of procedures "assumed" by someone or some unit to be the case are found in criminal violations and pranks. For the computer, unless programmed for "moral" recognition in some sense, this amounts merely to having consequences irrelevant to a computational procedure.

7. *Nominal and symbolic dominance.* Depending on the label, this kind of dominance was discussed in the last section. It is used by both humans and supervisory systems.

8. *Reprogramming dominance.* This kind of dominance can be vast, if effective programs are available or can be constructed. It amounts to a kind of second-order dominance. Presently it is difficult for a machine to reprogram another machine, and somewhat easier for a human to do so.

9. *Sanctioning dominance.* If the parameters of a process can be changed by input from outside, as in a learning program, then the process that supplies corrective feedback has sanctioning dominance to the extent that its input changes the target process.

10. *Ritual dominance/deference.* This is an emotional matter, of use in computer-human interactions, or good simulations of those. The computer greets the human with, "Good morning, Boss," and otherwise expresses willingness to comply, flatters the human, and so on. The human feels good. More humbling interaction may occur later.

CONCLUDING COMMENTS

The above listing by no means exhausts the possibilities for interaction which may hold among computing devices, or computers and people. Not mentioned, for example, were sequences such as negotiations, barter, and more complicated structures such as coalitions. These arrangements, and others, may be subjects for study as computers become more widely interconnected.

Other considerations not mentioned above are especially relevant to humans, and to the problem of possible computer dominance—at least, the problem of having healthy relations with these complex devices.

One matter is the issue of intent. It is not necessary for a device to "intend" anything for it to have an effect. A stalled car presumably has no possible intentions, but it can be effectively coercive in some situations. Similarly, the 'blind' behavior of computers can in fact consume great amounts of time of their human associates. Often this human time is not counted in evaluations of system efficiency. Since computers are symbolic devices, that work in time, it is possible for a person to be effectively conditioned to interact with them in certain ways (for example, frequently, or with a high frustration level, or only on challenging problems). This can be quite accidental.

If intent is not necessary for influence and for use of human resources, neither is it beside the point. Suppose a machine did "intend" to influence a person in some way, for instance to enslave the person, or to make him or her happy frequently, or to send out for refreshments. If a blind machine can influence behavior, then one with goals should have a better chance of success. It would be necessary for the machine to have an "understanding" of the person, or at least of tactics which were likely to work with members of the species, in addition to having goals of influencing the person. There is some controversy whether computing devices can have intentions and understandings of this sort.[5] Certainly they can have goals in very specific senses, and can exhibit adaptive behavior in some contexts. How well a machine, suitably programmed, might do in an influence attempt is another question. Simply the ability to have some intentions, for good or evil (as the human views it), does not mean that a machine would be successful. People, after all, frequently intend to influence each other, but do so only with varying success.

REFERENCES

1. Kimbleton, S. R., and G. M. Schneider, "Computer Communications Networks: Approaches, Objectives, and Performance Considerations," *Computing Surveys,* 7, 3, September 1975, pp. 129-173.
2. Blau, P. M. and W. R. Scott, *Formal Organizations: A Comparative Approach,* Chandler, San Francisco, 1962.
3. Bernstein, I. E., "Primate Status Hierarchies," in L. A. Rosenblum (Ed.). *Primate Behavior: Developments in Field and Laboratory Research,* Vol. I. Academic Press, New York, 1970.
4. Jacobs, T. O., *Leadership and Exchange in Formal Organizations,* HumRRO, Alexandria, Va., 1971.
5. Matson, Wallace, *Sentience,* U. Calif. Press, Berkeley, 1975.
6. Raphael, Bertram, *The Thinking Computer,* Freeman, San Francisco, 1976.
7. Likert, Rensis, *New Patterns of Management,* McGraw-Hill, New York, 1961.
8. Christie, Richard, and Maria Jahoda, (Eds.), *Studies in the Scope and Method of "The Authoritarian Personality",* Free Press, New York, 1954.
9. Rotter, Julian B., "Generalized Expectancies for Internal Versus External Control of Reinforcement," *Psychological Monographs,* Whole No. 609, *80,* 1966.

Structured training—A common-sense approach to developing ADP skills for improved job performance

by ALEXANDER P. GRANT and JACK L. STONE

Computer Education International, Inc.
Washington, D.C.

ABSTRACT

"Structured training" is a collection of instructional techniques, devices, materials, and methodologies, selected and tested by the authors, which has been successfully used to develop ADP technical skills and improve performance on the job. Structured training, in many situations, should result in more effective training than current on-the-job training programs, and standardized training courses provided by outside suppliers. Structured training has major applications inside medium and large scale production-oriented computer centers.

A major premise of structured training is that course design should be specifically oriented to performance improvements on the immediate jobs to meet training cost/effectiveness benefits. Structured training courses are therefore tailored to the needs of a specific installation. Best results are obtained when computer center objectives, course content, and student objectives are mutually consistent.

Major characteristics of structured learning are these: instructional modules are designed for an order of presentation that moves from the general to the specific. Reinforce~~ment of learning is achieved with extensive use of quizzes~~, directed class discussions, and coordinated on-the-job training or job-related workshop problems.

A case study of a structured training course implemented by the authors for a multi-mainframe U. S. military installation is presented to illustrate structured training.

It is concluded that, although structured training demands a substantial development and implementation effort, the benefits derived seem to be significant and cost-justifiable.

INTRODUCTION

The structured training course approach to ADP technical training discussed in this paper has been developed by the authors over the past several years during the process of designing and implementing a variety of ADP training courses for a wide range of government and commercial organizations. Installations for which structured training courses have been developed and presented have been medium and large scale production-oriented computing centers whose primary workloads consist of "commercial" type applications.

BACKGROUND

In data processing, as in any field subjected to the stresses of rapid change and expansion, the general level of performance of personnel tends to deteriorate over time. Although, in many installations, concurrent improvements in management, supervision, technical support and training have served to arrest or even reverse this tendency, in others, such measures have not been entirely successful. In particular, training efforts have often been either absent or disappointing in their outcomes.

Thus, in far too many modern computer centers, the status of development of technically trained personnel is well below that required to apply current computer technology in an effective and economic manner. Several major factors have contributed to the development of this situation, including:

(1) Substantial increases in workload volume, coupled with increasing pressures from users for improved responsiveness to their needs;
(2) Increasingly diverse and complex workloads;
(3) Rapid changes and increasing capabilities in hardware and software;
(4) Higher levels of responsibility imposed on computer operations and problem programming personnel;
(5) Continuing loss of trained personnel through promotion, transfers and terminations.

As to any specific installation, low levels of technical personnel effectiveness may be evidenced by one or more indicators:

(1) Excessive or increasing production re-runs;
(2) Excessive or increasing testing activity for equivalent program development workloads;

(3) Recurrent difficulty in meeting production commitments with an acceptable level of data accuracy;
(4) Excessive or increasing overtime to meet normal development or production requirements;
(5) Difficulty in responding to new or changing user requirements.

Accompanying these indicators may be more subjective, but nonetheless real, observations, including:

(1) Decreasing management and user confidence in the ability of the computer center to meet development and production commitments;
(2) Low employee morale;
(3) Excessive or increasing rate of voluntary separations, especially of the more efficient workers;
(4) Excessive reliance on a progressively smaller group of individuals in order to "get things done".

Faced with such problems, computer center managers have often turned to personnel training as one way of improving their operations. The outcomes of such training efforts have varied widely, but it seems reasonably safe to say that few managers are totally satisfied with the quality and utility of their training programs. The structured training course approach to training ADP technical personnel attempts to ameliorate the more common shortfalls of many current training programs for computer center personnel.

Of the several basic approaches currently in use for training in the center, none are objectionable in and of themselves, and any or all of them may find a useful place in a well designed and implemented training program. Among the more common techniques in current use, which are discussed below, are:

(1) On-the-job training (OJT);
(2) Standardized training courses supplied by equipment manufacturers or independent suppliers, including self-study courses with or without audiovisual aids;
(3) Formal on-site training courses.

On-the-job training is an invaluable adjunct to any technical training program and may be the primary method of choice in some restricted situations. However, in many installations, OJT is a mere facade used to mask the lack of any real commitment to personnel training. In other installations, OJT fails to attain its objectives because:

(1) It is administered on an *ad hoc* basis without any meaningful planning, control or evaluation;
(2) The level of training that can be achieved is limited by the capabilities of the personnel who can be made available to serve as instructors and the extent to which they can be made available;
(3) It is often restricted to the most rudimentary skills and most routine functions of the job;
(4) It is essentially a one-on-one instructional method and, therefore, relatively expensive.

Standardized training courses available from commercial sources include both instructor-presented courses and self-study packages. Instructor-presented courses are usually conducted at the vendor's site. Because most of those courses are designed to offer a general curriculum to a wide variety of installations, they are often not specific enough to meet the unique needs of a particular installation. Even when these courses are highly specific, the diversity of interest among participants from different organizations tends to reduce the effectiveness of the program.

So-called "self-study" and "programmed instruction" packages are offered either on an actual self-study basis or an assisted basis, i.e., a "monitor" or "course supervisor" is to be provided by the center to assist the student. The experience of the authors with these materials is that they have limited effectiveness when used on a self-study basis because many students have neither the self-motivation to struggle through the technical details nor the study skills needed to acquire knowledge in this manner; however, these materials, especially the audiovisual component, may have considerable value when used as support to an instructor in a classroom situation by providing diversity of presentation and reinforcement of learning.

Formal on-site training courses may be provided by either in-house staff or vendor sources. Such courses can be very good, but often fall short of attaining realistic training goals, because:

(1) Training is used only as a response to "crisis" situations with little or no planning;
(2) Training is often supervised by technical or administrative personnel who have little training, interest or experience in education or training;
(3) programs are often oriented toward academic objectives rather than specifically to job related training needs.

Since the authors are actively engaged in providing training services to computer centers on a commercial basis, these problems have had considerable importance and immediacy. To address these problems, over time a large number of different devices, techniques, materials and methodologies were tried by the authors and were either found wanting and discarded, or were proven and incorporated into a set of training strategies that, for convenience, are called collectively "structured training."

STRUCTURED TRAINING CONCEPTS

The structured training course approach is built upon the premise that cost/effective training in the production-oriented computer center requires structuring of courses that specifically provide training to maximize performance on the job. The conceptual framework underlying this approach encompasses a set of ideas that are neither new nor unique, but that in current training practice are as often honored in the breach as in the observance. The more important of these ideas are discussed in the following paragraphs.

A major consideration is that each structured training course be tailored to the needs of a specific installation.

The training objectives of the installation form the basis for course design as well as for evaluation of the course outcomes. Each instructional unit must contribute to the satisfaction of one or more of the computer center's training objectives.

Closely connected with this notion is the idea that course content must be consistent with the students' job-related objectives. Adherence to this idea can only be attained if students either have or are about to have assignments for which the training course is appropriate. Best results are attained when the computer center objectives, the course content, and the students' objectives are mutually consistent.

Course content must not only be consistent with student and computer center or objectives but it must also be structured so as to facilitate learning. The underlying concept here is that new knowledge is most easily acquired and retained when it can be readily integrated with existing knowledge. For this reason, the instructional modules of a structured training course are designed for an order of presentation that moves from the general to the specific.

Students are first presented with a conceptual overview of the course that serves both to forecast the material to be presented during the course and to illustrate the connections between the course content and the students' previous knowledge and experience. Similarly, each major module of the course is introduced by an instructional unit that presents an overview of the module and relates it back to the course overview. Thus, important ideas are constantly reviewed, reinforced and related to the students' existing knowledge.

Further reinforcement of learning is achieved with three techniques. The first of these is based on the ancient maxim that, "You teach what you test." The development of tests and quizzes that emphasize important concepts and details is basic to the success of this approach. In a structured training course, the primary objective of testing is not the traditional one of evaluating student progress but is instead to:

(1) Provide immediate reinforcement of learning;
(2) Improve student self-motivation by providing immediately realizable goals;
(3) Provide each student with an aid to self-evaluation.

Quizzes are used on a very frequent basis to achieve these objectives and the fact that such use also provides an excellent evaluation of student progress is a welcome but relatively unimportant result.

The second and related technique is the "directed" class discussion in which the instructor encourages the students to ask questions and to answer them. In this technique, the instructor serves as interlocutor: eliciting a question from one student and passing it to another to be answered. To the extent possible, the instructor refrains from directly providing answers.

A third method of reinforcement may take one of two forms dependent on circumstance: related on-the-job training or related workshop problems. If on-the-job training is used, great care must be taken to assure that such training is effectively coordinated with formal classroom sessions. Unless OJT is properly scheduled and consistent with formal instruction its effect is largely vitiated.

In situations where OJT is not practicable, workshops or laboratory problems may be used. Laboratory or workshop exercises should address problems that are based on specific operating situations in the computer center for which the structured training course was designed.

Design, development and implementation of a structured training course proceeds in accordance with a commonsense plan. Each step in the plan is necessary, but the extent to which each step is pursued is dependent on the needs of the specific installation for which the course is being developed. If the installation has a training program plan, the structured training course is developed within the context of that plan; otherwise, it is developed as a standalone course.

A necessary first step in development involves collection and analysis of information about the training needs of the installation. Such information may be obtained from: interviews; personnel records; management reports and planning documents; problem area analyses; and similar formal or informal sources. The output of this step is a program description document that is submitted to the installation for review and acceptance.

The program description document is, in essence, a preliminary recommendation for a training course. It contains an exposition of objectives, scope, student prerequisites, proposed student materials, instructional methodologies, and a summary statement of course content. Normally, this document also includes a detailed and time-phased topic outline. Upon acceptance of the program description, student materials are selected or developed and a detailed instructor's guide is prepared. The instructors' guide is designed to assist the instructor and contains:

(1) A summary description of the course content, methodology and objectives;
(2) A course schedule;
(3) A listing of student materials;
(4) A lesson plan for each instruction unit.

Each lesson plan contains:

(1) An estimate of the amount of time required for the unit;
(2) A statement of the objectives of the unit;
(3) A topic outline of the content of the unit;
(4) A description of listing of the student materials and study aids required;
(5) An exposition of student and instructor activities for the unit;
(6) Suggested instructional aids, i.e., audiovisual materials, chalk board diagrams, handouts and the like;
(7) Suggested tests or quizzes and discussion guides to be used with the unit;
(8) Where appropriate, case study materials, problem statements and the like for out-of-class study or workshop sessions.

The first step in implementing the training course is the course announcement. Such an announcement may be either in the form of a written bulletin or may be delivered orally at a meeting. In either case, it can serve several purposes, including:

(1) Informing all potentially interested persons about the content of the training course;
(2) Publicizing the organizational and personnel objectives of the course;
(3) Providing potential students with the opportunity to ask questions and to interact with management in relation to their individual needs and the relevance of the proposed training to career progress.

Persons selected to participate in the training course should not only meet course prerequisites but should also have a specific job-related need for the training offered. Unless a participant is going to apply knowledge and skills gained shortly after the conclusion of the course, both the individual and the organizational benefits of training will be largely dissipated. Additionally, students who lack personal job-related training objectives can not realistically be expected to have the level of motivation required to accept the workload and discipline necessary to a successful training experience.

The structured training course approach assumes and requires that the students be active participants in the training experience and not mere empty vessels into which knowledge is to be poured. Classroom operation is disciplined, course content is highly structured, and the level of student participation expected is high. Each class session is expected to result in a measurable increase in each student's knowledge or skills.

A typical class session begins with a short review of the material covered in the previous session. This is immediately followed by a short quiz designed to provide reinforcement of the important features of the previous instructional unit. The quizzes are then corrected in an interactive exercise in which the students provide the quiz answers with minimum intervention by the instructor. This exercise also affords the students an opportunity to seek clarification of the subject matter or to discuss related experiences.

This dialogue is followed by a presentation by the instructor of the next instructional unit. Short duration audio or video tapes may be used as adjuncts to the instructor's presentation. Depending on the type and length of presentation, the students may next address exercises or enter the workshop mode to solve large problems. Finally, the instructor introduces and discusses the outside reading assignment that is to be completed before the next class session.

At the conclusion of the training course, each student is evaluated by the instructor in terms of the progress he has made during the course. The instructor's evaluation is based on: quizzes and occasional formal tests; student participation and performance in discussions and workshops; attendance and attitude.

Student evaluation of the course is provided by questionnaire that may be submitted anonymously. Each student is requested to evaluate:

(1) The extent to which the course satisfied his individual training objectives;
(2) The facilities provided;
(3) The course content;
(4) The instructor's presentations and workshops;
(5) The student materials.

Student evaluations are tabulated and analyzed to provide data for the improvement of future courses.

CASE STUDY

A recent application of the structured training course approach illustrates many of its features. This application was designed and implemented to train computer operations personnel in a multi-mainframe U. S. military installation.

Over the years, this installation accumulated a large number of second generation mainframes, now at a single physical location, which were manufactured by a number of different vendors and are currently being phased out. The workload presently on this equipment is being transferred to an IBM 360/65 and an IBM 370/165.

The state of both employee training and morale when the course was conceived was considerably below an acceptable level. None of the operators had received any formal training in recent years with the exception of a three day briefing in 1974 on third generation equipment. Because of the continuing press of current workloads, those operators assigned to second generation equipment were not even being considered for training on the 360/370 computers. These conditions quite naturally led to lowered work efficiency and poor employee attitudes.

In this situation, the management of the computer center consulted Computer Education International, Inc., concerning the feasibility of producing a training course to meet both subject matter training and motivational objectives. As a result of these discussions the authors undertook the design and implementation of a structured training course for the operations personnel of the installation.

After assessing the training needs of the organization, a set of course objectives was documented and approved by management. The course objectives were to:

(1) Provide intensive formal training to advance all computer operators into the third generation system 370/OS environment;
(2) Introduce or reinforce skills in the purposive use of technical documentation;
(3) Improve the capability of operations personnel to communicate among themselves and with other computer professionals;
(4) Improve employee morale.

Investigation revealed that the target student population in general lacked an adequate understanding not only of OS operations but also of basic data processing systems development and implementation methodology. Although most operations personnel could respond adequately to routine situations, they lacked the knowledge to handle the unusual or problem situations and did not have the study skills necessary for self-improvement and self-training in the new third generation equipment.

From this study, it followed that the course content had to be based on the assumption that the student population had no prior training in the field of data processing including hardware, software and applications. Based on this assumption, the content included six instructional units: (1) Data processing system concepts; (2) 360/370 computing system equipment; (3) 360/370 Operating system concepts and facilities; (4) OS/HASP console operations; (5) OS job control language; and (6) OS service programs. The course was conducted four times during the last six months of 1975. Each offering consisted of thirty two-hour sessions, ten hours for each unit.

The course content was structured as a set of instructional modules and units organized in a "top-down" fashion. Characteristic of this approach to course design is that both the course and its major modules move progressively from the general to the particular. The first module of the course is, therefore, a comprehensive overview of the course that relates the course content to the students' background and forecasts the major subject areas to be covered. This order of presentation provides the student with a conceptual framework that, while it is constantly expanding, allows him to easily relate new knowledge to his past experience.

In the present case, the first ten hours constituted a major instructional module which served the purpose of providing an overall conceptual framework. The basic concepts of each of the included instructional units were covered and connected together in the presentation. Each of the five subsequent instructional modules began with a recapitulation and review of the concepts presented earlier, followed by detailed development of the current subject area.

Actual course presentation followed the structured training course model rather closely. Consistent use was made of quizzes and directed group discussions. Reading assignments were frequent, and extensive use was made of videotapes and other audiovisual aids.

Students were provided with a substantial amount of student materials, including: study texts, illustration booklets, and technical manuals. Adequate instruction and practice in the use of reference manuals for problem solving was afforded to all participants. Selected reference manuals were presented to each student for permanent use after the conclusion of the training course.

The objectives of the course were met reasonably well as determined by the installation management and by the students themselves. Because of the rather difficult personnel environment that existed prior to the class, the course was initially greeted with some skepticism; however, at the conclusion of the program, most students were pleased with the progress they had made and management was satisfied.

Computer center management indicated that, in general, employee attitudes were more positive, with operations personnel indicating an increased interest both in their work and in follow-on self development. Computer operators felt that they were able to communicate more effectively with programmers and production control personnel.

Although the course was generally well received and the course objectives were reasonably well satisfied, the authors feel that there were a number of areas where improvements might have been made; these include the following, which are discussed below: (1) Student selection; (2) Class scheduling; (3) Course design; and (4) Testing.

Subsequent to initial course design, management broadened student selection criteria. This resulted in a range of student experience, capabilities, and individual job-related objectives far wider than was desired. Work assignments of the actual student participants ranged from the purely clerical functions in production control to the highly technical functions of computer operators working at the 370/165 console.

Experience of the participants ranged from none to nearly sixteen years of operator experience. Among computer operators, participants had work experience ranging from second generation only to two years with the 370/165. Thus, for many participants, major segments of the course were not adequately relevant to either their immediate job needs or their individual job-related objectives.

Instructional efficiency would have been improved had the student population been more homogeneous. For example, production control personnel should have been assigned to one class, second generation operators to another, and so on; however, this was not possible in the production situation. This, in fact, may be a problem that cannot be totally solved in a training environment involving technical people with production commitments.

A related problem was the scheduling relationship between class sessions and the participants' work shifts. Although most students were able to handle the heavy workload imposed by the daily two hour class session plus outside reading in addition to their regular eight hour shifts, some students who attended at the conclusion of their shifts were simply too tired to benefit fully from training. Students perform better if they attend class prior to their daily shift.

All things considered, the course design was reasonably successful. However, two major changes are suggested for future courses. The first change is to make the instructional modules smaller in scope and to increase the number of levels of detail between the most general and the most detailed. This change will assist the instructor to more readily adjust the speed and content of his presentation to the capabilities and individual objectives of his students.

The other change relates to the establishment of course objectives. In the design of the course discussed in the case study presented in this paper, the authors attempted to assure that the design would meet management goals by delivering a briefing and preliminary outline to management for their review and concurrence. However, many students had different views as to their technical training needs. In the future, it is suggested that a preliminary topic outline and questionnaire be distributed to all potential class participants to elicit from them their views of their training needs before the course content is finally adopted.

In the initial presentations of the course, some departure was made from the structured training course approach to the use of testing. The frequent testing was thought too difficult for these students; therefore, testing was infrequent and used principally for instructor evaluation of student progress. After evaluating the outcome of these earlier sessions, the decision was made to return to the original testing concept. The use of testing was increased and redirected to reinforce learning, promote student self-evaluation and improve retention of learning. This change resulted in a noticeable improvement in student motivation and levels of performance.

CONCLUSION

Experience with this and other technical training programs has convinced the authors that structured training courses are an effective, common-sense approach to providing ADP skills training in commercial-type computer centers. While it demands a substantial development and implementation effort, the benefits derived seem to be significant and cost-justifiable.

The role of a formal training program in attracting and developing computer professionals

by LAWRENCE F. LUNETTA, JR.

Honeywell Information Systems
Phoenix, Arizona

ABSTRACT

Because of the wide range of career choices that face computer-oriented graduates at the bachelor's level, a formal training program is often an effective mechanism to attract and maintain a high-powered technical staff. Often the fundamental choice for the new graduate is between full-time graduate school or full-time work. An explanation is given of how a training program can combine work experience and graduate school, and the impact that these development opportunities have on the organization's recruiting, development and planning efforts. Honeywell Information Systems' Advanced Engineering Program is used to illustrate a successful training program, and its constituent parts are highlighted: rotating work assignments, graduate education, and practical problems and seminars. Also, the importance of permanent placement for program graduates, and the net result of program training, are discussed.

INTRODUCTION

The individual graduating with a bachelor's-level degree is today faced with a number of difficult, yet important career decisions. On the most basic level, the choice is between pursuing a graduate degree on a full time basis or entering the workforce on a full time basis. More specifically, the individual who has a computer emphasis in his or her undergraduate program of study has still further choices besides the two mentioned above. If graduate school is chosen, then which program is appropriate? Electrical Engineering, Computer Science and Business are all possible avenues of graduate education. The full-time work environment is just as varied, whether it be hardware or software, applications programming, systems analysis, or systems programming. The choices are as diverse as they are plentiful.

Because of this myriad of choices facing the computer-oriented graduate, recruiting and maintaining a high-powered staff of computer professionals is a challenging task for any organization. This paper will focus on some effective techniques for attracting and satisfying talented computer professionals, with a particular emphasis on the role of formal training within the organization. The use of training, and in particular, a formal training program, has a direct impact not only on the quality of individuals attracted by a recruiting effort, but also on the career these individuals will have as permanent members of the organization.

As an example of the role a training program can play in an organization utilizing computer professionals, Honeywell Information Systems' Advanced Engineering Program (AEP) will be presented. The experiences and lessons learned in implementing and administering the AEP will illustrate the effectiveness and importance of a formal training program in the computer environment.

BACKGROUND AND ENVIRONMENT

Honeywell Information Systems is the "computer arm" of Honeywell, Inc. Its products represent complete computer systems starting with a line of mini-computers (Level 6) all the way up to some of the largest computers in the world (Level 66). Honeywell appreciably enhanced its computer business (particularly at the top end) through a merger with General Electric's computer interests in 1971. One of the locations acquired by Honeywell at the time of the merger was the Phoenix Computer Operations in Phoenix, Arizona.

Honeywell has built the Phoenix Computer Operations into one of the largest integrated computer design and manufacturing centers in the world. In this one location, the entire design, development, and manufacturing of Honeywell large computer systems (Level 66) is accomplished. Within the design and development effort is included all the hardware and software for Level 66.

One of the legacies that GE passed on to Honeywell was several training programs in the engineering, finance, and manufacturing areas. One of these training programs, the Advanced Engineering Program, was started in 1968 in Phoenix to attract and develop top computer talent for the Phoenix Computer Operations. Along with the finance training program and manufacturing training program, Honeywell has continued the Advanced Engineering Program, and over the years it has evolved into a vibrant, dynamic, effective source of computer professionals for Honeywell.

Today, the Advanced Engineering Program has three facets aimed at developing well-rounded, complete computer professionals: rotating work assignments, graduate education, and internal training/problem solving.

DESCRIPTION OF THE ADVANCED ENGINEERING PROGRAM

The Advanced Engineering Program is a three year program and as such affords an individual a substantial opportunity to see many different aspects of the computer business through the use of rotating work assignments. The work assignments are designed to last between six and nine months, thus allowing the program member to see between four and six different areas. The AEP takes advantage of the fact that Honeywell's Phoenix Computer Operations provide almost the entire set of computer technology in one location. The assignments are carefully designed by the Engineering Unit Managers to provide challenging work which contributes to overall product development, and are selected on that basis by the program member.

While a great deal of learning is derived through work experiences, continuing graduate education is a key contributor to an AEP member's development. To facilitate obtaining a technical Masters degree, the program member is given time off from each work week to attend classes at nearby Arizona State University in Tempe, Arizona. The Masters degree programs which program members pursue are either Electrical Engineering, Industrial Engineering, or Math-Computer Science. The technical Masters degree generally takes two years to complete after which the AEP member has the option to continue towards a PhD or pursue an MBA.

The third part of the program—the internal training—allows for a dynamic tailoring of the three year program to reflect state-of-the-art technology. This aspect of the program involves lectures and seminars conducted by experts on the Engineering staff and may cover topics such as compiler or operating system design, error detection and correction techniques, the impact of microprocessor technology, etc. The criteria for topic selection are applicability to Honeywell's design effort, and susceptibility to rapid change through technology improvements. The seminars are usually followed by a series of practical problems to provide "hands on" experience for the program member and to further illustrate the use of a particular concept or idea.

RECRUITING FOR THE ADVANCED ENGINEERING PROGRAM

If the Advanced Engineering Program sounds particularly rigorous and time consuming, that's because it is. It takes a very special individual to be successful on the AEP, one motivated enough to spend 50–60 hours a week in work-related activity. There is no "typical" profile for an AEP member, but there are certain characteristics (or accomplishments) each member must have. Recruiting for the AEP is done nationwide through an extensive mailing campaign as well as through a corporate recruiter. As a result, out of a group of 10–15 new program members (hiring is done once a year in June), each one will have graduated in the top five percent of his or her class with a bachelors degree in Electrical Engineering or Computer Science. Perhaps ten different schools will be represented, and the group's cumulative grade point average will be around 3.7/4.0.

In addition to a very high GPA, program members tend to be very active in social, professional, and honorary societies as undergraduates. Since program members are expected to work in highly integrated design groups dealing with a very complex technology, this "people orientation" is very important. Advanced Engineering Program graduates are targeted to be technical and managerial leaders of Honeywell's computer business. Thus, the individuals who join the program tend to be very motivated persons and aspire to a high degree of achievement. It is not unusual for program members to publish technical papers and participate in patent developments very soon after joining the program.

ADMINISTRATION OF THE ADVANCED ENGINEERING PROGRAM

Another interesting aspect of the AEP is the manner in which it is administered. As Honeywell has designed it, the program resides in the employee relations (Personnel) function for the purpose of budgeting and headcount allotment. The net result of this is that the program members are budgeted in employee relations, but work on a day-to-day basis in engineering. This allows tremendous flexibility on the part of both engineering managers and the program members in setting up and selecting assignments, free of any budget constraints.

There is a full time administrator for the Advanced Engineering Program who works directly for the Manager, Technical Individual Development in Employee Relations. The Manager of Technical Individual Development is responsible for all development activities for the technical community in the Phoenix Computer Operations. This includes in-plant classes, seminars, etc., in addition to the AEP. The job of the administrator is to insure the smooth operation of all facets of the AEP, and, in fact, the person who fills the job is on the third year of the AEP. The administrator and manager jobs are one year positions filled by a rotation sequence which starts in the administrator's position and ends with a one year appointment as Manager of Technical Individual Development. This keeps control of the program very close to the program members themselves, with a large amount of support and guidance from upper management. The administrator's job is filled by the top performer of all those on the second year of the program and, as such, has a tremendous measure of respect and influence.

The program members enjoy the fact that the individuals who administer the program have experienced their problems and, in fact, have often shared some of the projects on which they have worked. It is much easier to get problems, complaints, or requests out in the open since the program members easily relate to recent graduates of the program. Peer management problems are avoided because the Manager of Technical Individual Development is always a program graduate and upper management lends support and recommendations whenever needed.

THE IMPACT OF A TRAINING PROGRAM ON RECRUITING

The activity which feels the immediate impact from a training program is the recruiting effort. As outlined earlier, the computer-oriented graduate faces a myriad of career choices upon degree completion. A training program, such as the Advanced Engineering Program, which combines practical work experience with graduate education is very attractive to highly motivated graduates. It allows a "dual-path" approach to establishing a career (education and work experience), relieving the graduate of an often difficult choice.

The highly successful individual who is attracted to a program like the AEP is attracted by the "special" opportunity a training program offers. A common sentiment expressed by program members is that the four or five years of hard work that their undergraduate record required has been "rewarded" by such a wide-ranging opportunity.

A company like Honeywell establishes a very progressive image through its training programs. A formal training program is ample evidence that there is commitment to individual development, and a measure of career planning for the new college graduate. Again, an important question which often arises during a job interview is "what type of development opportunities are available for me, both in education and work experience?" The Advanced Engineering program is explicitly designed to answer that question and to allay the fear that the young engineer will "get stuck in a job that I don't like."

A successful training program sells itself. A key aspect of AEP recruiting is the contact that the prospective program member has with engineering managers and current program members throughout the interview day. Neither the managers nor program members are "coached" on what to say, but the obvious positive feeling about the program comes through again and again in a spontaneous fashion. It is impossible to overestimate the value to the recruiting effort of this phenomenon of the position "selling itself."

Now that the AEP has been established for several years, the job of finding potential program members on college campuses has been made easier. Because of the wide variety of schools represented by current and past program members, many Electrical Engineering and Computer Science departments are aware of the AEP and encourage their best students to contact Honeywell. By maintaining correspondence with their undergraduate schools, program members are often able to recommend friends that are a year to two behind them in school—a very reliable and fruitful source of candidates.

With the major emphasis in recruiting now shifting to minority and female placement, the Advanced Engineering Program provides a tremendous advantage in attracting high caliber individuals. Given that the training program is a springboard for future leadership opportunities, this is a natural mechanism for affirmative action.

Additionally, since there is a full-time program administrator, any special problems or concerns that might arise are handled quickly, carefully, and on a personal basis. Since the AEP is very youth-oriented, the assimilation and adjustment process for each program member, and in particular, for females and minorities, is a very comfortable one. It is often these intangibles that are overlooked in the recruiting and placement process, but a training program such as the AEP is very successful in attracting high-caliber engineers because of its stress on personal adjustment and development.

DEVELOPMENT OPPORTUNITIES ON THE ADVANCED ENGINEERING PROGRAM

Although the recruiting effort is appreciably strengthened, the ultimate purpose of a training program is to develop technical and business leaders for the future. The Advanced Engineering Program is designed to provide Honeywell with solid computer professionals, and each facet contributes to a program member's development.

The rotating work assignments allow program members great latitude in acquiring practical skills and work experience. By moving to a different technology area every six to nine months, essentially software-oriented individuals can hone their skills through experience in compiler and operating system design, but still obtain an exposure to one or two hardware areas. The converse is obviously true for hardware-oriented individuals.

In addition to learning the multi-faceted computer technology, the program member is also introduced to a wide variety of work environments through the rotating work assignments. Over the course of three years, an AEP member will have worked for five or six different managers, thus experiencing different managerial styles and project leadership. This experience is invaluable when it comes time to choose a permanent assignment, as the program member learns to recognize the work environment in which he or she is most comfortable.

Graduate education at Arizona State University provides a good measure of the theoretical foundation needed in the development effort. Every program member is required to obtain a technical Masters degree in either EE, IE, or MATH/CS before graduating from the program, and the commitment to this education effort is underscored by the fact that program members are given time off from the work week (up to eight hours per week) to attend classes with all tuition, fees, and books paid for by Honeywell.

The formal educational opportunities provide an additional benefit for those seeking to "cross" specialities or change career emphasis. It is not unusual for an individual who joins the AEP with a predominantly hardware background to discover an interest in software on a work assignment and then decide to pursue that interest in graduate school. Even more gratifying is the Computer Science major who joins the program with little or no hardware background only to discover through classes and a carefully selected work assignment that there are career opportunities in that area also.

The bridge between the theoretical emphasis of graduate school and the practical application of the work assignment is provided by the seminars and practical problems. This technique is used to expose the group to a new technology very rapidly. This also allows the training program to emphasize concepts important to Honeywell as a business, while staying very close to the leading edge of technology.

Because of the wide variety of opportunities that the Advanced Engineering Program provides, each program member can create a development program specially suited to him or her. It is this special feeling of control that each program member has over his or her own career which "turns on" AEP members.

BENEFITS TO THE COMPANY

Honeywell derives many benefits, some obvious, some not so obvious, from the Advanced Engineering Program. As mentioned earlier, from a recruiting standpoint, the program cannot be beat. It is an outstanding vehicle through which a large number of high potential graduates are introduced into the company. This periodic influx of new talent not only bodes well for the future, but also reduces the prospects of the engineering staff falling prey to technical obsolescence.

Because the program members continually require challenge, their work assignments usually involve work on critical new products. The payoff here is that the activity contributes directly to a product so that in addition to the program member benefiting via the experience, the company benefits from useful work being done.

The fact that program members rotate throughout the organization provides for rapid technology migration. One of the best communication techniques is to have "living witnesses" for a particular concept carry the word from assignment to assignment. Since there is such close integration between the hardware and software of today's computer systems, it is most advantageous to have individuals who can carry new product ideas from one area to another. It is not unusual to have an AEP member design hardware on one assignment and then write software for that hardware on a later assignment—a close-to-optimum situation.

THE CHALLENGE OF A TRAINING PROGRAM

Graduates of a training program—particularly the Advanced Engineering Program—present a challenge to the organization. This challenge is to effectively utilize the skills and the training developed by each program graduate.

An AEP graduate represents a well-rounded computer professional. He or she has had three years of intensive training in both hardware and software areas through the use of formal education and practical work experience. An AEP graduate has learned to lead groups, as well as to cooperate within a group. In short, since the program attracts ambitious, aggressive individuals, there is a potential problem in placing them in suitable permanent assignments.

Expectancy theory postulates that the level of activity towards a certain goal depends to a large extent on an individual's perceived reward upon reaching that goal.[1] This concept is quite evident when dealing with the Advanced Engineering Program. There is continual scrutiny of the opportunities of program graduates by current members of the program. It would be impossible to routinely ask for 50–60 hours of work per week from these individuals if the rewards were not readily apparent. Indeed, the prospects of retaining program members and program graduates in light of their superior credentials would be dim without a concerted effort to carefully place and monitor each program graduate.

Honeywell enjoys close to an 85 percent retention rate for program graduates. (There are no contracts or binding agreements.) Successfully placing each program graduate is probably the most difficult aspect of the entire training process to accomplish and Honeywell's success involves careful human resource planning, along with extensive individual counselling. Each program member chooses a permanent assignment, not forced into a convenient opening.

Thus, the real cost of training, in terms of time, effort and planning is in permanent placement of the program graduates. Unless an organization is willing to commit to providing opportunities of responsibility, leadership and impact for its program graduates, the program could fail. As computer professionals, graduates of the Advanced Engineering Program approach the highest level of Maslow's Need Hierarchy—that of self-actualization.[2] Given the appropriate opportunities to grow into leadership roles, these individuals are a tremendous resource for the organization.

SUMMARY AND CONCLUSIONS

A formal training program such as the Advanced Engineering Program presents a "no lose" situation for both the corporation and the individual program member. From Honeywell's standpoint, for the dollar investment represented mostly by tuition, fees and book costs for graduate school (assuming the positions must be filled anyway), the return in increased recruiting effectiveness, work contribution, organization communication and future leadership, is tremendous.

The program member enjoys, of course, the benefits of free graduate education and flexible job mobility. The accelerated development that the AEP member enjoys is a natural outgrowth of experiencing several different learning environments. Through the "timesharing" of activities throughout the work week, a program member does not have to sacrifice practical work experience for graduate school, or vice versa. In fact, the general consensus is that each of those experiences is complemented in a synergistic fashion because they do occur simultaneously. In addition, there is a tremendous advantage in a career with Honeywell by being associated with a winner—the Advanced Engineering Program.

This concept of being associated with a winner is what makes a training program so effective. Recruiting is made easier because both program members and managers alike will sell the opportunity. However, success like this does not happen overnight. The support of the entire organization stems from the "small" successes achieved on a day-to-day basis over several years. Since 1968, each facet of the Advanced Engineering Program has provided tangible proof of its success. There have been over 50 graduate degrees awarded to AEP members, 50 patents, and over a score of papers published.

While these statistics alone do not provide the raison d'etre for the AEP, they do reinforce the feeling that it is "working." Combined with consistently high performance appraisals for work assignments, the patents, paper and degrees make it very easy to fund the program year in and year out.

Based on the experience with the Advanced Engineering Program, it is safe to say that a formal training program can play a key role in resource planning in an EDP-related organization. Computer technology is very complex and computer design and usage requires a good understanding of the relationship between the constituent parts of a system. Honeywell uses the Advanced Engineering Program to insure a steady supply of technical and business leaders who understand the total computer environment to support a growing, dynamic business.

A growth-oriented organization will find that a formal training program provides a return on human investment worthy of the most lucrative financial investment, particularly in the computer environment.

REFERENCES

1. Hersey and Blanchard, *Management of Organization Behavior,* Prentice-Hall, Inc., Englewood Cliffs, N.J., 1972, pp. 9–21.
2. Torgersen and Weinstock, *Management—An Integrated Approach*, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1972, pp. 197–205.

Personal computing—An overview for computer professionals

bu JIM WARREN

Dr. Dobb's Journal of Computer Calisthenics & Orthodontia
Menlo Park, California

ABSTRACT

The brief history, current status, and foreseeable developments in the areas of home and hobby computing are outlined. Major points in the two and half year history are presented. Characteristics of current hardware, software, and systems configurations are discussed. Mention is made of a variety of activities for which these systems are currently being used. Emphasis is placed on those features and uses that are unique to personal and hobby computing. Differences are noted between personal computing and professional computing. Developments known to be in progress are outlined. Several interesting and feasible projects that the author believes no one is yet pursuing are mentioned.

INTRODUCTION

Within the past several years, general purpose digital computers have become a consumer product. A number of developments have occurred that are unique to this new area of personal and hobby computing. More are imminent. Some of these developments are beginning to impact the professional computing community. This paper surveys the brief history of home computers, the current situation, and the foreseeable developments. The reader should be aware that, since this paper was written over six months prior to NCC 77, portions of it are either obsolete history or six-month prognostications. And, six months is a fifth of the lifetime of the home computing movement.

There are two distinct subjects within the realm of home computing: hobby computing, in which the users view the hardware and software systems as the primary items of interest, and personal computing, in which the participants are primarily interested in the applications of personal computers. Up through 1976, users of home computers were invariably hobbyists. The hardware that was available in a consumer's price range was too complex[1] for the casual user who was primarily interested in serious personal applications. Additionally, the available systems software was, for the most part, insufficient to support serious applications work. This situation was about to change at the end of 1976. By the middle of 1977, there should be a number of units on the market in the $1000 range that are fully assembled,[2] include a useful amount of memory and interfacing for low-cost mass storage, and include systems software that make them marginally usable[3] by the applications-oriented non-professional.

Typical configurations

The vast majority of personal computers are 8-bit machines.[4] Memory tends to range from one kilobyte to 64K, with most users having 4K-20K. A majority of home systems appear to use a standard television as their primary alphameric output device. Not uncommonly, they have no hard-copy capability. Phillips-type audio tape recorders are the dominant mass storage medium, usually having a transfer rate in the range of 100-800 bytes/second. Assemblers are in widespread use. Various versions of BASIC form the dominant "high level" language used for most applications.

Differences from industry

A number of noteworthy differences exist between personal computing and professional computing. Within home computers: Price is crucial; speed, capability, and reliability are secondary. Time and effort is essentially free. The primary motivation is entertainment in the broad sense of the word. Essentially, all not-for-profit home computers are purchased out of the family's entertainment budget.

Neighborly, noncompetitive sharing is quite widespread among home computing enthusiasts. Software, assistance, and solutions to problems are readily exchanged among hobbyists.

Attractiveness of the hardware is of minor consideration to computer hobbyists. Although they might wish to have a beautiful enclosure for their computer, they will be more likely to invest limited funds in additional capability than in a walnut cabinet. This characteristic will almost certainly change, however, as there come to be more personal computers than hobbyist computers.

Second-hand equipment is in quite widespread use among hobbyists. This is particularly true of peripherals. To a minor extent, used minicomputers are also appearing in home computing environments.

This is a consumer marketplace. Thus, it requires a much higher level of customer service and responsiveness than has been traditionally available in the small computer industry. Users of personal computers tend to have less expertise and less patience than industrial computer consumers, and tend to expect a level of vendor responsiveness equivalent to that found in other retail consumer markets.

There is no national organization of the personal computing community. There are several regional organizations, however, and 160-200 or more local amateur computer clubs.[5]

The extreme price sensitivity of the marketplace has enabled the widespread development of very small businesses: computer craftspeople. A surprising number of highly reputed products—hardware and software—are being manufactured or assembled in basement workshops or garages. As a small company becomes successful, grows, incurs more overhead costs, and finds it necessary to raise its prices, it thereby makes room for a new, lower-cost garage shop competitor.

HISTORY

The first microprocessor, the Intel 4004, was produced in 1974. It has been estimated that, prior to 1975, there were only 50-200 computers in personal use. Although an Amateur Computer Society[6] has existed for some years, it has remained in the background of the current personal computing movement, and has had little if anything to do with its growth or popularity.

The beginning of hobby computing can easily be dated from January, 1975. In that month, *Popular Electronics*[7] carried a cover story on the Altair 8800 microcomputer kit newly available from MITS,[8] a small electronics kit manufacturer in Albuquerque. Within three months, two computer hobbyist clubs had formed[9] on opposite sides of the country, unknown to one another. In July, 1975, the first retail computer store opened in Los Angeles.[10] By September, the new hobby had its first national magazine.[11] May, 1976, saw the first regional amateur computing convention.[12] Three months later, the first "national" convention was held and drew an estimated 4,500 enthusiasts.[13]

As of December, 1976, the following situation existed: There were 16,000-20,000 computers installed in homes.[14] There were 160-200 local computer clubs, and at least three regional associations of clubs.[15] The largest club had 4,000-5,000 local members.[16] There were 150-400 computer stores.[17] (The number is difficult to ascertain since some "computer stores" are mail order distributors with only a P. O. box. Others are general electronics retailers who happen to carry hobby computers. Still others may be deeply involved in computer retailing but keep a carry-over name implying that they are a hi-fi store or a repair shop.) At least two companies are actively pursuing franchise licensing in multiple states and plan to set up chains of computer stores.

As of December, there were three national magazines[18] explicitly directed to this audience. Two more were about to publish their first issue.[19] Another had published its first and only "monthly" issue in August.[23] Two more periodicals[24] existed that were targeted to the broader topic of popular computing. The largest of all of these[11] had a circulation in excess of 60,000. Two others[21,25] were reporting circulations exceeding 20,000.

Two or three major conventions explicitly concerned with personal and hobby computing were planned for 1977, as of last December. Additionally, a number of smaller trade shows and conferences were being promoted. The First West Coast Computer Faire planned for 7,000-10,000 attendence at a weekend event in San Francisco in April.[26] NCC'77 expected their Personal Computing Section to expand attendance by at least several thousand.

HARDWARE

There are three identifiable levels of computers in personal use. The first level is the tutorial unit costing $100-$400, having a keypad "front panel," a minimal monitor in read-only memory (ROM) or programmable ROM (PROM), and 256-1K bytes of semiconductor random-access memory (RAM). The second level is the hobby computer costing $600-$1200, with a full front panel or a turnkey control panel backed by a good monitor in PROM, and including 4K-29K of RAM. The final level is the small industrial computer priced at $1,000-$1,500 or more with a full front panel and 4K-8K bytes or words of RAM. Used minicomputers also fall in this third category.

Two types of home computers can be identified: kits, and preassembled units. The kits are usually—but not always[27]—less expensive for a given level of capability. They generally require some reasonable degree of hardware and electronics expertise[1] in order to properly construct and debug the hardware. A considerable increase in the number of preassembled units[2] on the market is expected in 1977, and these units are expected to be reasonably cost-competitive with the kits. Several manufacturers have reported that it is as economical to produce assembled units as it is to produce kits and have to staff the large customer service activity necessary to back the kits.

Peripherals

Several traditional computer peripherals are popular among hobbyists, most notably used Teletypes, I/O Selectric typewriters, and paper tape equipment. Some home computers also have joy sticks[28] for two-dimensional input, or digital-to-analog converters, usually used for output of music or voltages to drive CRT's. Fewer systems have real time clocks, or sensors or analog-to-digital converters for real-world input.

A variety of peripherals have appeared that are unique to home computing. TVT's—television typewriters—are in widespread use by hobbyists. These scan a portion of the system's RAM, treating the bytes of data as ASCII character codes and generating a video signal for use with a television or video monitor. Video graphic capability has been available for over a year in the form of a 64x64 color array[28] or 128x128[28,29] black-and-white display. 256x256[29] black-and-white arrays should be economically available by the middle of 1977. That provides half the resolution of a standard U. S. television and is quite adequate for interesting real-time graphics.

Several dot-matrix impact printers are appearing in the hobbyist market. These are priced in the range of $200-$400[30] and print 40-80 columns on regular paper.

Three speech synthesis units were on the market by the end of 1976 and a fourth[33] one was expected in January. They are $400-$750. One[31] is based on a much more expensive unit that has been in industry use for several years. Another,[32] with proper editing, allows not only voice synthesis, but also singing and music synthesis. A speech input experiments unit was also planned for introduction in January.[34]

Other subsystems that are expected on the hobbyist (and industrial) market before the middle of 1977 include several mini-floppy disc systems for $600-$700,[35] microprogrammable microprocessor kits,[36] a 16-bit processor[4] that will fit in many of the hobbyist computers, and 64K byte memory[37] banks on a single circuit board.

Undone products

Several products would be of interest to the personal computing community that this author believes are not yet being planned by any manufacturer. An electronic telephone dialing peripheral would be simple to produce from current components,[38] and could provide hobbyists with the hardware necessary for an electronic telephone book. Acoustic couplers and modems can be easily interfaced to home computers. Then, once the communication protocols are developed, personal computers can communicate with one another, and with central data and program repositories. Somewhat further into the future: interfacing to video tapes and video discs would provide the home computing enthusiast with on-site encyclopedic storage capacity.

SOFTWARE

Hobbyists with tutorial systems often program in machine code and load the programs via the front panel. On hobby and industrial level systems, editors, assemblers, debuggers, and monitors are in widespread use, often based on cassette tape. A variety of BASIC interpreters are available.[20] Interpreters for several string processing languages, PASCAL P-Code, APL, and LISP subsets are expected to become available in 1977. Compilers for BASIC and FORTRAN[37] are also expected to appear in 1977. Several disc operating systems for floppy discs,[39] and cassette operating systems[40] for Phillips-type cassette tape are available, but are little more than simple file systems and program development systems. They have little comparison to the DOS found on midi and maxi computers in business or industry.

1977 should see the introduction of a number of special purpose editors, e.g., for manuscript preparation, word processing applications, and document generation. Spelling checkers should appear around the end of 1977.

A considerable variety of compilers for machine code or pseudo-code to be interpreted are expected in 1977. There has been and will undoubtedly continue to be considerable experimentation and "homebrewing" of programming languages for home computers. Many of these will be highly interactive. Block structured languages are beginning to appear.

There is very widespread demand for business applications software. As of the end of 1976, very little had appeared. A considerable number of people were known to be working on various business packages, however, and these should appear in 1977.

Software problems

Several problems exist for home computing software. One major problem is that of machine independence. Even with machines having identical processors, a variety of design decisions have been made by the hardware manufacturers that impact software. Some assume a given I/O port has a specific function. Many furnish software or firmware—particularly monitors—that make hardware or address assumptions. TVT's commonly make some hardware assumptions concerning the location of the display memory. Few of the programmers developing software for a given CPU give adequate care to isolation of the hardware- and address-dependent aspects of their programs.

In the purely software realm, there are a number of variations of BASIC.[20] Rarely will source code for one version be acceptable without change to another BASIC interpreter.

Control of proprietary software that the hobbyists consider overpriced has been a problem. In these cases, many hobbyists will freely exchange software with little consideration for its proprietary character. Most of the producers of software for this market have chosen to sell it for a large price to the hardware manufacturers who need it to enhance their products. Or, they have chosen to sell it for a very nominal fee—often including source-code listings—and depend on sales volume to obtain an adequate return on their time and effort. It is evident that different marketing practices must be adopted when marketing software to not-for-profit home users than have been used in selling software to a more controllable, for-profit user community.

A possible solution to this problem well may appear in the form of software being stored in masked ROM that plug into an external socket on the computer. Manufactured in sufficient quantity, such firmware units could be priced low enough to be economically competitive with the cost of a tape cassette, plus being far more convenient to use.

USES FOR HOME COMPUTERS

Undoubtedly the question most widely asked concerning personal computers is, "What will they be used for?" In answering this question, the traditional business and industrial uses will be ignored, even though a number of hobby computers are being put to traditional uses in traditional environments.

The most widespread use, currently, is unquestionably for the purpose of playing games.[22] The games are usually single-player games, though an increasing number of multiple-player games are beginning to appear. Some of the games are simple games of chance. Many of them, however, involve simulations of varying degrees of complexity. Many such games have a significant subliminal education value. Most of the games seen through 1976 used only alphameric input and output. An increasing number of games are beginning to appear that use graphic output,[41] and this trend is certain to continue.

Self education concerning computer hardware and programming well may be the next most widespread use of these home computers, to date. Probably a majority of the owners of personal computers spent most of their time, through 1976, learning how to build and debug the hardware, and then learning how to use basic systems software and program. Much of the software work undoubtedly was concerned with learning how to modify some existent software so that it would work with a system that was "almost like" the system for which it was designed. A surprising number of disassemblers have appeared in the hobbyist community, included several that were written in BASIC.

A number of systems include simple analog output equipment, often used for experimentation with computer generation of music.[42] A number of people appear to be experimenting with biorhythm or biofeedback applications. Amateur radio enthusiasts are showing widespread interest[43] in home computers, both for radio applications—e.g., code conversion, antenna control, message processing—and for their own value as a technical hobby. There is considerable interest in word processing applications, however the cost of good-quality hard-copy devices has restrained development in that area. There is also great interest among the physically handicapped, though little work appears to have been completed in this area, so far.

Contrary to what many outsiders appear to expect, virtually no work has been done in the area of automating the house, or kitchen, or sprinkler system, or security system. This is not surprising when one realizes that the interfacing problems are significant and expensive, the need is minimal, and the computers were purchased from entertainment budgets rather than home improvement funds.

Foreseeable personal applications

Black-and-white, and color graphics is an area of considerable and rapidly increasing interest. Economical video interfaces of good resolution should be on the market before the end of 1977. Somewhat further in the future, higher speed processors and interfaces to video tape units will give the personal computer user access to exciting computer graphics and animation facilities.

Considerable interest will be shown in 1977 in experimenting with the several available speech synthesis units. As experimentation in this and the graphics area matures, it is reasonable to expect self-teaching systems for prereader children to begin to appear. Speech units will also prove useful for the handicapped.

Mundane applications such as intelligent music tape systems[20] and electronic phone books[38] may be expected before the end of the year, at least as homebrewed designs. Some experimentation with holographic art[44] and light shows has already begun and will slowly develop in sophistication. Experimentation will considerably increase in applying computers to amateur radio problems. Work will also begin in networking "ham" computers via radio transmission. The hams will need considerable assistance from the computer hobbyists who are familiar with network designs and protocols.

Word processing, manuscript preparation, and letter writing applications will slowly increase as used I/O Selectrics can be found and interfaced to home computers. This application area will explode as soon as someone manufactures an inexpensive, reliable, removable, non-destructive interface that will fit most electric typewriters.

Experimentation will continue and grow in the area of computer music.[42] This year or next year will see the availability of more complex analog output devices which, when coupled with faster processors, will allow very exciting experimentation in the realm of sophisticated electronic music. Input devices appropriate for such music systems will develop more slowly.

As is currently true, there will be unending experimentation with the design and implementation of systems software and homebrewed hardware. This can be viewed as entertaining and intellectually stimulating self-teaching (with all of the problems thereof). There will be significant developments in the areas of floppy disc operating systems, and resident compilers for reasonably interesting high level languages.

The first steps will be taken towards computer networking via unconditioned telephone lines using couplers or modems. Probably, some of the first results of this will be interactive games between users at different sites. *Significant* usage of computer-to-computer communications is unlikely to occur before 1978 or 1979.

Personal computers will see an exponentially increasing use in hardware, software, and mathematics education, both in home and in public schools.

Applications software will continue to be dominated by games and simulation programs of increasing complexity. Simple data processing software will appear in 1977 to perform such chores as maintain the family budget (balancing a checkbook is too trivial), process club and organization mailing lists, keep indices of music albums, stamp collections, etc., and possibly perform dietary intake monitoring. Spelling corrector software may appear by early 1978. Very simple database management systems will appear by late 1977 or early 1978.

The more distant future: three to five years

Electronic libraries, in which one may browse by subject, keyword, or whatever, will appear. These will include the capability of making marginal notes that are stored locally and mapped to a single original of the full-text publications. Prototyping of this system is already under way[45] and should be available in a rudimentary form in 1977.

Various versions of computerized bulletin boards, message centers, want ads, and "switchboards" will become available. A mobile, LSI-11 version of this is currently being implemented.[46] The prototype may be operational by the end of 1977.

An electronic news system into which *all* of the news is fed—at least all of the news concerning, say, the computer or electronics industry—may be operational within five years. There appears to be a considerable problem in extending such a system into the general news media, in that newspaper publishers have a significant vested interest in keeping it from happening. And, the wire services—from whom much of the news flows—are financially tied to the publishers. Once implemented, however, it would mean that everyone would have access to *all* of the news, rather than having an Editor inserted between them and the news source.

IMPACTS ON COMPUTER PROFESSIONALS

The immediate impact is already appearing. There are more jobs in that there are more applications to be programmed, more systems to be developed, more units to be maintained, and more consumers to be served. A rash of computer stores are opening. More and more often, they are being financed by an entrepreneur who is not a computer professional. Yet, they require a person who is knowledgeable of computer hardware and software to adequately serve their customers. It is becoming increasingly simple for a competent computer professional to spin off from a company, develop his own product, and be his own boss through selling that product. Since his overhead is often minimal, he can afford to price his product low enough to make it highly competitive.

The impact, further into the future, is less predictable, and may best be illustrated by some questions: What happens to computer science education when an entering college freshman, interested in science or engineering, has already designed and implemented three compilers, two interpreters, and a parser in a LISP tree? What happens to programming jobs when the average manager can program in three or four high level languages and does so, regularly, on his home computer. What impact will there be on undergraduate electrical and computer engineering programs when the average freshman engineering major not only has been programming since he was in elementary school, but has been repairing the family computers for four or five years, and worked for four summers in the repair shop of the local computer store?

The potential for computer professionals—and anyone owning his own computer—to interact with the society at large is even more interesting: What happens to food marketing when you can link your home computer to the club's grocery price database and run a quick optimal shopping program? What effect will there be when you have access to *all* the news; not just all the news that an Editor feels is fit to print? What effect will there be on the political process when any consumer group or individual voter can trivially ascertain a candidates' *actual* voting record?

As far as this author is concerned, the next several decades hold promise for a great deal of excitement.

REFERENCES

1. Jef Raskin provided a broad-ranging evaluation of computer hobbyist products in "Personal Computers: A Bit of Wheat Amongst the Chaff," *DDJ*,[20] September 1976, pp. 15-17.
2. Fully assembled, good capability machines should be announced by a number of companies before June 1977, including units from ECD Corp., 196 Broadway, Cambridge, MA (news to be released 76-12-1); Apple Computers, 770 Welch Rd., Palo Alto, CA 94304 (prototype was demonstrated, July 1976), and MITS[8] (expected to include a high resolution CRT).
3. Software to support random-access mass storage will still be very limited.
4. Two exceptions are the LSI-11 from Digital Equipment Corp., Maynard, MA, and the 16/8 microprocessor board already prototyped around a custom LSI chip from Western Digital, to be available from the Computer Mart of Orange County, 625 W. Katella #10, Orange, CA 92667.
5. Extensive lists of computer clubs have been published in *PCC*,[22] and *IA*.[21] An up-to-date, computerized list is being maintained by the Computer Faire, Box 1579, Palo Alto, CA 94302.
6. Amateur Computer Society, 260 Noroton Ave., Darien, CT 06820.
7. *Popular Electronics*, 1 Park Ave., New York, NY 10016.
8. MITS, 2450 Alamo SE, Albuquerque, NM 87106.
9. Homebrew Computer Club, Box 626, Mountain View, CA 94042.
10. The Computer Store, now located at 820 Broadway, Santa Monica, CA 90401.
11. *Byte*, 70 Min St., Peterborough, NH 03458.
12. The Trenton Computer Festival, Trenton, NJ, May 2, 1976. It was estimated that there were 1,500 people in attendance, and 45 exhibitors. Sponsors were the Amateur Computer Group of New Jersey, UCTI, 1776 Raritan Rd., Scotch Plains, NJ 07076.
13. The Personal Computing '76 Trade Fair was held in Atlantic City, NJ on Aug 28-29, 1976. It included 103 exhibitors. It was sponsored by the Southern Counties Amateur Radio Association of New Jersey.
14. No "hard data" was available at the time this article was written. Venture Development Corp, 1 Washington St., Wellesley Hill, MA 02181, distributed a survey to approximately 1,000 hobbyists in the summer of 1976, received about 300 responses, and on that basis projected that there would be 18,600 computers in homes by January, 1977. Much more extensive surveys were conducted by *Byte* magazine, *DDJ*,[20] and *IA*[21] in Oct-Dec, 1976, but the results were not available in time for this paper.
15. The Chesapeake Microcomputer Club, Inc., 236 Saint David Ct X4, Cockeysville, MD 21030, has six chapters. The Midwestern Affiliation of Computer Clubs, 14058 Superior Av #8, Cleveland, OH 44116, has eleven affiliates. The SCCS[16] has seven chapters.
16. Southern California Computer Society, 1702 Ashland, Santa Monica, CA 90405.

17. The Computer Faire[5] maintains an up-to-date computerized list of stores and retail distributors.
18. *Byte,*[11] *DDJ,*[20] and *IA.*[21]
19. *Kilobaud,* Peterborough, NH 03458, and *Personal Computing,* Benwill Publishing Corp., 167 Corey Rd., Brookline, MA 02146.
20. *Dr. Dobb's Journal of Computer Calisthenics & Orthodontia,* PCC, Box E, Menlo Park, CA 94025.
21. *Interface Age,* Box 1234, Cerritos, CA 90701.
22. *People's Computer Company,* a bimonthly tabloid, Box E, Menlo Park, CA 94025.
23. *Microtrek,* 333 Dows Bldg., Cedar Rapids, IW 52401. Only the August issue had been published by the end of November, 1976.
24. *People's Computer Company,*[22] and *Creative Computing.*[25]
25. *Creative Computing,* Box 789-M, Morristown, NJ 07960.
26. The First West Coast Computer Faire[5] was sponsored by a number of local and regional educational, professional, and amateur groups interested in personal computing. These included local chapters of the ACM and IEEE, Stanford's EE Department, UC-Berkeley's Lawrence Hall of Science, the Homebrew Computer Club,[9] and the SCCS.[16] As of December, 1976, it expected 200 exhibitors, 100 conference sessions.
27. For example, the Apple[2] is priced below many of the kits of equivalent capability.
28. Cromemco, 2432 Charleston, Mountain View, CA 94043, manufactures excellent, inexpensive joy sticks, as well as the "TV Dazzler" which provides both black-and-white and color TV graphics (of limited resolution).
29. Matrox, Box 56, Ahuntsic Stn., Montreal, H3L 3N5 manufactures a number of alphameric video units and several video graphics arrays.
30. For example, Southwest Technical Products, 219 W. Rapsody, San Antonio, TX 78216, offers a 40-column printer in kit form for $250.
31. The Votrax kit from Federal Screw Works, 4340 Campus Dr. #212, Newport Beach, CA 92660.
32. The CT-1 from CompuTalker Consultants, Box 1951, Santa Monica, CA 90405.
33. A new unit from Logistics Speech, c/o John Ross, 900 Dickson St., Marina Del Rey, CA 90291.
34. Heuristics, Inc., 900 N. San Antonio #C1, Los Altos, CA 94022.
35. For example, a single drive unit from North Star Computers, 2465-4th St., Berkeley, CA 94704, will include a Shugart mini-floppy drive and complete controller and interfacing to many hobbyist computers, and will be priced at $599.
36. Ratheon Semiconductor Division, 350 Ellis, Mountain View, CA 94042.
37. Technical Design Labs, Research Park, Bldg. H, 1101 State Av., Princeton, NJ 08540.
38. The essential element is a solid state binary-to-dial pulse converter available from Collins Radio Group, 4311 Jamboree Rd., Newport Beach, CA 92663.
39. For example, Digital Research, Box 579, Pacific Grove, CA 93950, has an excellent floppy disc operating system called CP/M available for $70, including a "loaded" floppy disc, documentation, a TECO-like editor, transparent debugger, assembler, and a PDP-10-like command language.
40. Digital Group, Box 6528, Denver, CO 80001.
41. Cromemco has "Space War" running on their TV Dazzler.[28]
42. See the *Computer Music Journal,* PCC, Box E, Menlo Park, CA 94025.
43. Chod Harris, an organizer for the American Radio Relay League stated that there were approximately 290,000 "hams" in the U.S. in November, 1976, and estimated that approximately one fourth of them had a serious and active interest in hobby computing. *Kilobaud*[19] is being published by the publisher of *73 Magazine,* the third largest amateur radio periodical in the U.S.
44. Multiplex (holograph images), 448 Shotwell, San Francisco, CA.
45. The Xanadu system, Ted Nelson, Itty Bitty Machine Co., 1316 Chicago Av., Evanston, IL 60201 (being implemented on a PDP-11 in TRAC, Calvin Mooers' trade-marked, copyrighted, and patented string processing language).
46. The Community Memory system, Lee Felsenstein, LGC Engineering, 1807 Delaware, Berkeley, CA 94703.

Operational software for restructuring network databases*

by DONALD E. SWARTWOUT, MARK E. DEPPE and JAMES P. FRY
The University of Michigan,
Ann Arbor, Michigan

ABSTRACT

A high-level "access path" approach to database restructuring is described and contrasted with the "elementary operations" approach taken by most restructuring systems. With the elementary operations approach, restructuring is viewed as a sequence of basic or "primitive" operations which manipulate a source database in order to convert it into a target database. In the access path approach, restructuring is seen as the process of accessing a body of information represented by the source data, and constructing the target database representation of the same information. While the elementary operations approach is useful for restructuring hierarchical databases, it does not generalize well for networks. The access path approach is better-suited to the complex structures possible in network databases.

The access path approach permits the specification of complex restructuring transformations in terms of application-oriented concepts such as access strategies and selection criteria. A non-procedural Network Restructuring Language (NRL) based on this approach is presented, and an example of its use in restructuring is given. The architecture of an NRL-driven Restructurer for network databases is described.

INTRODUCTION

Due to the fact that the database design process is essentially an opaque art,[1] an important tool needed by a Database Administrator is a generalized reorganization facility. With such a facility the designer can adapt existing database structure to conform to new information or processing requirements. For example, a new entity or relationship could be added to the database structure to satisfy a new information requirement, or a new access path could be added to satisfy a processing requirement.

Although some reorganization schemes exist in database management systems, they deal primarily with the physical organization of data, e.g., performing garbage collection, changing the blocking factors, and to some extent, modifying the accessing mechanism. To our knowledge no generalized reorganization facility exists in any database management system and, in particular, we are unaware of a restructuring facility which would provide the capability to change the logical structure of the database.

In the current state of the art, the reorganization facility has developed only within the context of a Data Translator.[2–6] As described in References 2 and 6, a Data Translator is a generalized software system that accepts as input a source database, descriptions of the source (input), target (output) databases and the mapping between the source and target logical structures. Its output is the reorganized target database produced automatically by the generalized software from the input description. The execution of the translator invokes three major modules—Reader, Restructurer, and Writer. While the function of the Reader and Writer modules can be classified primarily as physical reorganization or *reformatting,* the primary purpose of the Restructurer module is to logically reorganize the database or restructure it.[7] Birss and Fry[6] described the physical reformatting capabilities of the Reader and Writer through the logical restructuring capabilities which were embodied in several prototype implementations. In this paper we ~~focus on the development of a generalized restructuring~~ capability for the network class of logical structures.

What is restructuring?

The purpose of restructuring is to transform the logical structure of a database in response to new information or processing requirements. Navathe and Fry[8] developed a categorization of restructuring operations for the hierarchical class of logical structures. Three fundamental logical structure modifications were defined—Naming, Relation, and Combining—which were refined into several lower level restructuring operations. Figure 1 illustrates a few of these hierarchical restructuring operations. The first part, Figure 1a depicts a hierarchical relationship among PRESIDENTS, SPOUSES and CHILDREN. The uppermost box

* Supported by WWMCCS ADP Directorate Defense Communications Agency Contract No. DCA-100-75-C-0064

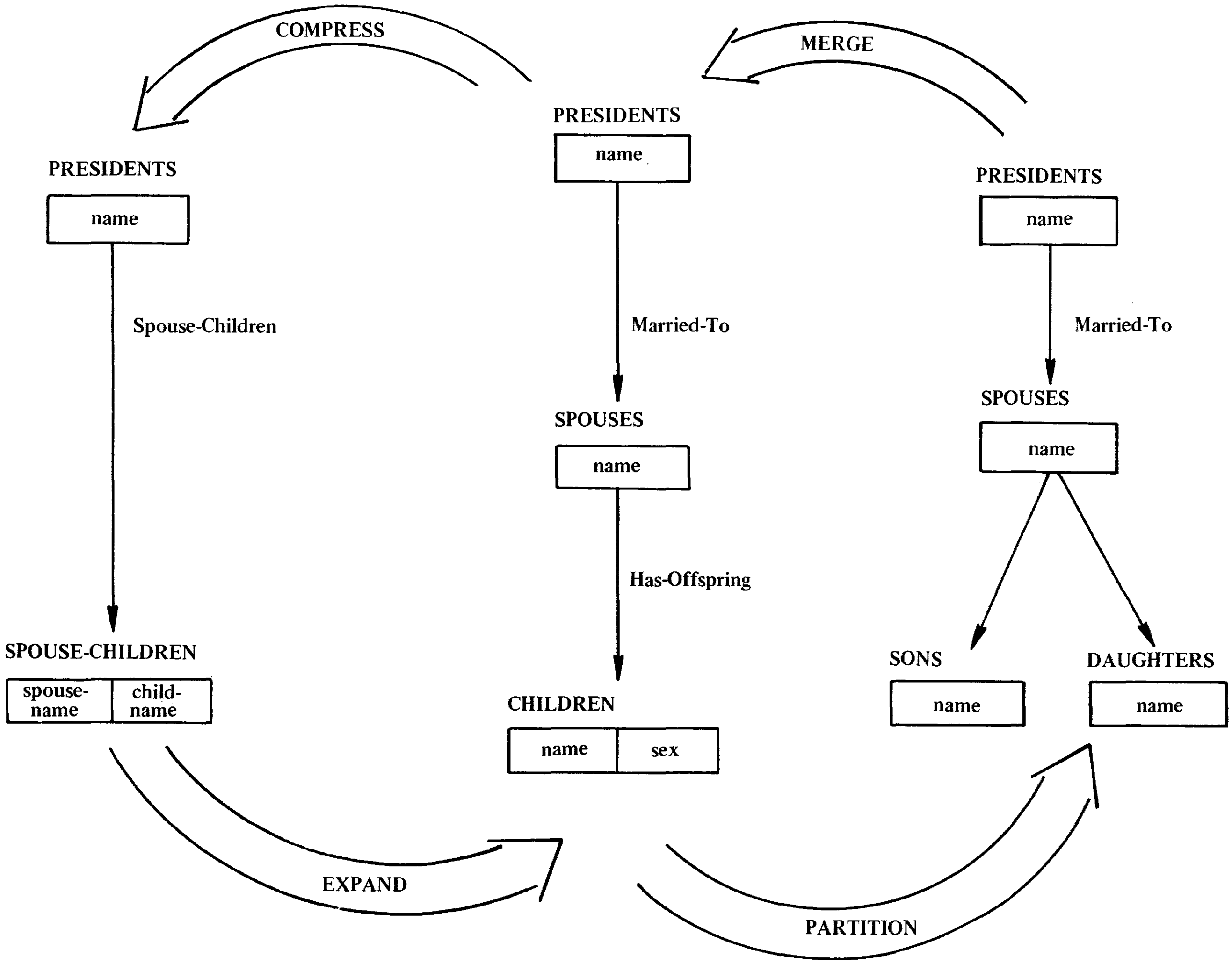


Figure 1—Restructuring operations

or record type,** PRESIDENTS,[4] contains the names of the presidents of the United States. The second level record type, SPOUSES, is related to PRESIDENTS through the set type "Married to" and contains the names of all the spouses for each president. Thus, the set is a one-to-many mapping over two record types. In a similar fashion, the third level record type, CHILDREN, is related to SPOUSES and contains the names of the children born to each SPOUSE.

One restructuring operation is *partitioning* in which the occurrences of a record type are divided into two or more distinct record types based upon the value of one or more data items. This operation is illustrated by the source and target logical structure of Figure 1 where the source record type CHILDREN has been partitioned into two record types, SONS and DAUGHTERS, based on the data item "sex." Notice that the data item "sex" has been eliminated in the logical structure of Figure 1, since this information is now represented by the target logical structure. The dual restructuring operation of partitioning is *merging*. The merging operation consolidates occurrences from two or more record types into a single record type, often adding a data item to the record type to preserve information previously represented by the logical structure. Merging is illustrated by the source logical structure and the target logical structure of Figure 1, where the occurrences of the two record types, SONS and DAUGHTERS, have been merged into a single record type, CHILDREN. Note that the terms "source" and "target" are relative and depend on the direction of the restructuring transformation. Consequently, Figure 1 represents the source logical structure for the partitioning example, and the target database for the

** We follow the terminology of DBTG.[9]

† For a more complete description of the Presidential Database see Reference 10.

merging example. Notice the item type "sex" has been added to contain the information previously represented semantically by the source logical structure of Figure 1. Another form of restructuring involves the *compression* of two or more hierarchical levels into one. This example is illustrated by the source and target logical structure of Figure 1 where the two source records, SPOUSES and CHILDREN, have been compressed into a single record type, SPOUSE-CHILDREN. On the record occurrence level, this operation would be accomplished by replicating the associated SPOUSE record occurrence for every CHILD record occurrence. The dual operation, *expansion* expands one level of hierarchy into two or more by factoring out selected data items. This procedure is illustrated in Figure 1 where the SPOUSE-CHILDREN record type has been factored into two levels.

Network restructuring

Although a complete categorization of such operations has been established for the hierarchical class of structures,[6] restructuring operations are not nearly as easy to classify in the network class of logical structures due primarily to the variety and complexity of the structures involved. For example, an important network restructuring operation, *changing the implementation of many-to-many relationships*, is illustrated by the restructuring transform of Figure 2. The source logical structure represents information on students, their class standing, all courses taken by each student, and the final grade received in each course. The target logical structure represents exactly the same information except that the association between students, course, and grades is achieved using a LINK record type. This example is typical of the restructuring operations which may be posed in a network environment, those which involve several record and set types.

Another example, which might enhance processing, would be the *addition of indexing sets or record types*. This could be achieved by migrating the data item "class" to the record type CLASS-STANDING, which would serve to segment the students into graduate, undergraduate, special, foreign exchange, etc. (Figure 3). Notice that this operation is actually expansion performed on a hierarchical substructure of the target structure of Figure 3, namely the single record type STUDENTS.

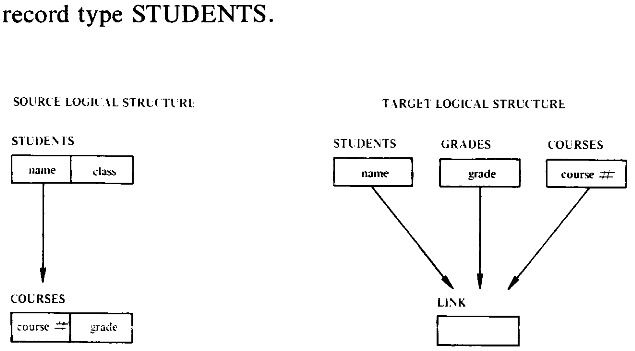


Figure 2—Network restructuring example

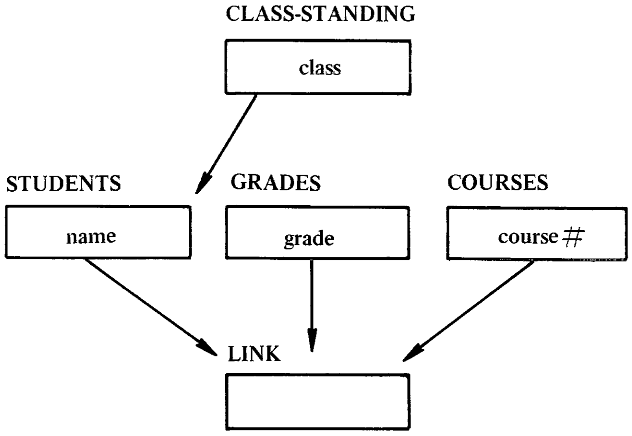


Figure 3—Addition of indexing sets

Yet another important network restructuring capability is the ability of the restructuring process to extract not only data which exists explicitly in the source database, but also that which exists implicitly, i.e., the information which may be inferred from the actual source data. An example of the difference between implicit and explicit information is described in the hypothetical restructuring transform of Figure 4.

The example describes a source database containing two record types, PERSONS and LINK. The PERSONS record type contains name and sex item types, while the LINK record type contains no data but provides relationship information in conjunction with sets PARENTS and CHILDREN. The target structure also contains the PERSONS record type, and construction of target PERSONS records involves the extraction of data explicitly resident in the source file PERSONS record type. In contrast, target record types PARENTS and GRANDFATHERS do not correspond directly to any source file record type; they contain information which is represented implicitly in the source file. Needless to say, this transformation cannot be described by operations on hierarchical substructures.

In the next section, we review the previous work in developing hierarchical restructuring capabilities through specification of elementary operations. The third section comprises a discussion of the access path approach necessary to achieve a network restructuring capability, and the fourth section describes a Network Restructuring Language based on this approach. The implementation of the Restructurer is discussed in the fifth section and the paper concludes with our observations on building and using restructurers.

THE ELEMENTARY OPERATIONS APPROACH

An interesting analogy exists between restructuring systems and high-level query systems. In fact, one can consider a query as a restricted restructuring transformation in which the target is not a database but some other form of

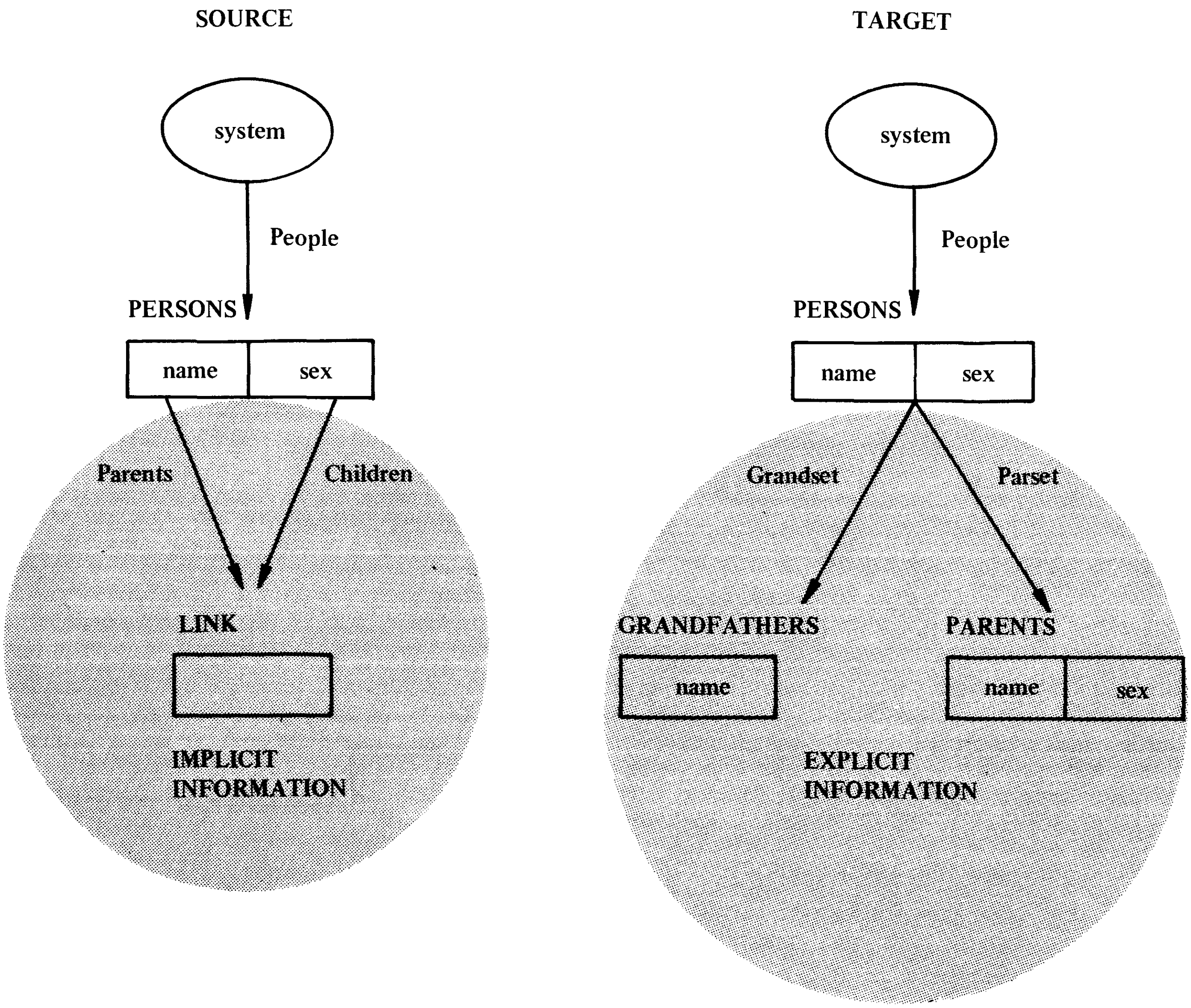


Figure 4—Implicit data extraction

information representation. Efficient, easy-to-use query languages for the hierarchical class of logical structures have been built through the use of elementary operations and have been in existence for some time.[10] These systems allow a user to specify a sequence of elementary operations on hierarchical structures which will accomplish his query. Such languages are easy to learn and use since they employ a few rather simple operations to perform most queries to hierarchical structures.

Several research efforts have addressed the problems of restructuring and restructuring language specification from the elementary operations point of view. CONVERT, a high-level translation language, provides a generalized restructuring capability for hierarchical structures. The approach is based upon the concept of a "data form" in conjunction with a set of restructuring functions called "form operations." Shoshani[11] takes a similar approach to restructuring whereby a set of "conversion functions" is used to specify restructuring operations.

In the elementary operations approach, the source database is considered to be a collection of data in a specific logical structure and format, while the target database is viewed as essentially the same data, but in a different logical structure and format. Restructuring, therefore, is considered to be the process of manipulating the source data to conform to the target logical structure and corresponding format. Consequently, restructuring research using this approach turns toward the development of low-level or "primitive" operations which transform occurrences of one logical structure to another. One advantage of such an approach is that the architecture of the restructuring software system is greatly simplified; it defaults to a set of low-level subroutines which correspond directly to the elementary operations. Thus, a restructuring specification consists of a sequence of primitives which may be directly converted to a sequence of subroutine calls which perform the actual restructuring. Unfortunately, the user is not shielded from any aspects of the restructuring; he must thoroughly understand the function of each operation in order to be able to use it. Consequently, the language, although high-level, still requires the user to treat restructuring essentially as a sequence of low-level steps.

Several other problems arise when the elementary operations approach is applied to restructuring network databases. In general, elementary operations are designed to operate on small, logical substructures consisting of one or two record types and a set, to produce a new, logical substructure. Due to the limited number of operations which may be defined, only a finite number of source substructures can be valid candidates for restructuring. The more complex the structure, the greater the probability that it will contain substructures which are not valid candidates for the set of available operations. Consequently, the restructuring capabilities of elementary operations decrease as the complexity of the structure increases. For this reason, elementary operations are not particularly well-suited to describing restructuring transformation upon complex network logical structures. In addition, a complex restructuring transformation (assuming that it may be performed by the elementary operations) will require the specification of a complex sequence of elementary operations which is difficult to analyze and to understand.

Another class of restructuring transformations that is difficult to accomplish with elementary operations is the extraction of implicit information. We have pointed out that such transformations (as in the example of Figure 4) cannot usually be described by a sequence of operations on hierarchical substructures. Furthermore, although they may be describable by sequences of elementary operations on network substructures, such descriptions are generally long and complicated. There is also reason to believe that, given time, designers of network databases will produce enough intricate methods of storing information implicitly to exhaust any set of elementary operations.

Following our analogy, it has been found to be extremely difficult to generalize the elementary operations of high-level query systems to network databases.[12] The elementary operations approach to developing a query system for network databases tends to be less powerful and much more cumbersome than its hierarchical counterparts. In general, they suffer from the same problems cited above—limited allowable input structures, overly complex specifications, and difficulties with link records and other implicit information storage techniques. The high-level access path approach to restructuring grew out of attempts to develop a restructuring strategy more suited to network databases.

THE ACCESS PATH APPROACH

Following the current trend in host language database systems which process the network structure databases, we chose the high-level "access path" approach to restructuring. The source database is viewed as a body of information, some of which is represented explicitly by data, and some of which is represented implicitly, i.e., may be inferred from the data. Similarly, the target database is considered to contain a subset of information represented by the source; this data is created from information provided by the source data. Executing a restructuring transformation, then, is simply the process of traversing the source database to obtain the information needed to create the target database, and storing it according to the target logical structure. There are numerous consequences of this approach. For one, research in restructuring specification turns toward the development of restructuring specification languages based on the concepts of access strategies and selection criteria. Since this area is closely related to query language development, certain concepts from previous research in this area may be utilized. A second consequence of the access path approach is manifest in the actual restructuring algorithm development. Restructuring technology turns toward the development of algorithms which efficiently and exhaustively access the source database and perform tests upon the data, following externally specified access strategies and test criteria. This is a radical change from the elementary operations approach which tends to develop low-level subroutines.

The most important consequence of this approach is that it produces powerful generalized network restructuring capabilities. Since restructuring is viewed as an operation which accesses information (rather than manipulates data), the approach is unaffected by the logical structure of the database. This independence from the logical structure insures that essentially any database is a valid candidate for restructuring, regardless of the complexity of the logical structure (hierarchical, network, etc.). Furthermore, source and target logical structures need not even remotely resemble each other since the target is derived from information provided by the source rather than from the source structure itself. Also (unlike the elementary operations approach), implicit information may be extracted and restructured as easily as explicit information (as in Figure 4). Thus, explicit information may become implicit and vice versa.

Finally, the system is inherently simple. Assuming that the user has some familiarity with databases and applications, a restructuring specification language based on application-oriented concepts such as access strategies and selection criteria should be easily understood. Furthermore, since all restructuring operations are expressed as information accessing problems, confusion does not significantly increase as the complexity of the restructuring transformation increases. An algorithm designed to carry out this process is also simple and therefore, straightforward and dependable. All restructuring is performed in an identical sequence of steps, regardless of the particular transformation: (1) exhaustively access the source data according to the access specifications, (2) test data based on selection criteria, and (3) create target record occurrences containing the relevant data. For a more complete discussion of the theoretical foundations of this approach, see Reference 13.

NETWORK RESTRUCTURING LANGUAGE

Architecture

The architecture of the Network Restructuring Language is based upon the high-level specification of access paths. The major components of the language describe access strategies and selection criteria. Access strategies are described using an access path statement which specifies the traversal scheme required to obtain data for each target record type. The selection criteria establishes the source (and, indirectly the target) data requirements. It would be beyond the scope of this paper to describe the NRL in great detail. A complete language specification may be found in Reference 14. However, the major points of the language will be described.

The NRL is essentially a block-structured language in which each "block" contains a single *target record statement*. There is one target record statement for each record type in the target database. This structure reflects the access path approach for the following reason: each target record type is considered to represent a certain quantity of information which may be obtained from the source database; consequently, each target record description contains the specifications for the source database accessing scheme as well as the selection criteria. The second level of structure in the NRL is the *target set statement* (see Figure 5 for the NRL structure). Each target record statement includes one or more target set statements which identify the sets of which the target record type is a member. The third level of NRL structure is the *access path statement*. The access path statement specifies exactly how the source logical structure is to be traversed in order to obtain the information necessary to create an occurrence of the target record type. There may be many individual access path statements—one or more for each target set statement—since the information which contributes to the target record type may come from several different source record types.

```
TARGET RECORD STATEMENT
  TARGET SET STATEMENT
    ACCESS PATH STATEMENT
      NEW TARGET ITEM STATEMENT
      SOURCE RECORD STATEMENT
        ITEM QUALIFICATION STATEMENT
        ITEM ASSIGNMENT STATEMENT
        •
        •
      SOURCE RECORD STATEMENT
        •
        •
    ACCESS PATH STATEMENT
    •
    •
  TARGET SET STATEMENT
  •
  •
TARGET RECORD STATEMENT
•
•
```

Figure 5—Structure of NRL specification

There are two types of NRL statements at the fourth level of structure: the *new target item statement*, and the *source record statement*. There may be zero or more new target item statements for each access path statement. Such a statement indicates a target item which receives a constant value each time a target record occurrence is constructed using the specified access path. Since access paths indicate structure in the source database, the new item statement is useful when information represented semantically in the source structure is converted to actual data values in the target structure.

The second NRL statement at the fourth level of structure is the *source record statement*. The source record statement identifies the source record(s) on the access path which will be used in the creation of a target record occurrence, to obtain item values and/or test the source data. There may be one or more source record statements for each access path statement since data may be obtained or examined from several different source record types. Furthermore, each source record type may be optionally assigned an *index number*. The index number is used to identify uniquely a source record within an access path which "cycles" or "loops" back on itself such that the same source record type appears more than once. Finally, there are two statement types at the fifth and final level of NRL structure; they are called the *item qualification statement*, and the *item assignment statement*. Item qualification statements are used to establish the selection criteria used in testing the source data. Consequently, one may specify a constant value to be compared against a source item value which will determine whether or not a target record occurrence is to be created using the current set of source data. The item assignment statement is used actually to obtain the source item values and assign them to the proper target items.

It should be noted that the NRL does not explicitly describe the logical structure of either the source or target database. The Michigan Data Translator obtains appropriate descriptions of the source and target databases independently of the restructuring specifications. The NRL, therefore, describes only the information relevant to the actual transformation from source to target, and assumes that the logical structures of the databases have been previously defined and are available to the Restructurer.

An NRL restructuring example

The NRL example has been chosen to illustrate the capability of the language to specify restructuring capabilities which may not be readily classified in terms of hierarchical structures or elementary operations. The example involves the extraction of implicit information from a network structure and is taken from Figure 4. This example is reproduced in Figure 6 for convenience. The NRL required to accomplish the necessary transformation is documented in Figure 7. The form of the NRL has been simplified slightly for clarity. The block structure of the language may be easily observed. Because there are three target record

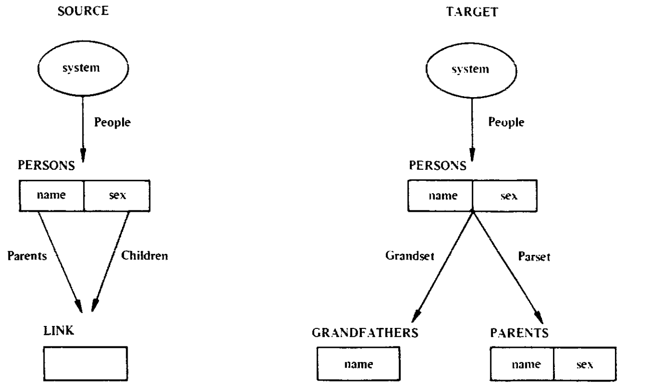


Figure 6—Restructuring example

types, there are three target record statements. Statement #2 begins the NRL description for target record type PERSONS. Since PERSONS is a member of only one set (from the system access level) there is only one target set statement (#3). Statement #4 specifies the source access path to be used to locate the data necessary to create the target record type. ACCESS PATH=PEOPLE (PERSONS) describes an access path from the source SYSTEM node along set PEOPLE to record type PERSONS. Statement #5 is the source record statement to indicate that source record type PERSONS is to be used to obtain information. Statements #6 and #7 are item assignment statements which indicate that the values in source item types "name" and "sex" are to be assigned to the target item types "name" and "sex."

The NRL description for the second record type, GRANDFATHERS, begins with statement #8. As before, there is only one target set statement (#9) because the record type is a member of only one set (GRANDSET). Statements #10, #11, and #12 specify the source access path to be used to obtain the desired information. This access strategy may be summarized as follows. The LINK record type in the source models the relation between parents and children. Given a particular PERSONS record occurrence, one may access all of his/her parents by using the set types PARENTS and CHILDREN in conjunction with the LINK record type. One similarly obtains all grandparents of a person by accessing all parents of parents. The access path statement on lines #10, #11, and #12 describes the essence of this strategy: use set PEOPLE to the PERSONS record type, then set PARENTS to the LINK record type, then set CHILDREN back to the PERSONS record type, then set PARENTS back to the LINK record type, and finally, set CHILDREN back to the PERSONS record type. Notice that the source record type PERSONS is used three times in the access path specification: once as a person, once as a parent, and once as a grandparent. Consequently, the source record statement (#13) must indicate which source group PERSONS is to be used. Since the PERSONS record which represents grandparents is the fifth record on the access path, INDEX=5 makes the necessary distinction. Statement #14 is an item qualification statement which is used to select only male grandparents to yield grandfathers, the desired target record information. Statement #15 assigns the value in the source item type "name" to the target item type "name."

It should be clear that the target record type PARENTS

```
 1)  NRL
 2)     TARGET RECORD PERSONS
 3)        TARGET SET PEOPLE
 4)           ACCESS PATH = PEOPLE (PERSONS).
 5)              SOURCE RECORD PERSONS
 6)                 NAME = NAME
 7)                 SEX = SEX

 8)     TARGET RECORD GRANDFATHERS
 9)        TARGET SET GRANDSET
10)           ACCESS PATH = PEOPLE (PERSONS), PARENTS (LINK),
11)                         CHILDREN (PERSONS), PARENTS (LINK),
12)                         CHILDREN (PERSONS).
13)              SOURCE RECORD PERSONS, INDEX = 5
14)                 SEX QUALIFY IF = 'MALE'
15)                 NAME = NAME

16)     TARGET RECORD PARENTS
17)        TARGET SET PARSET
18)           ACCESS PATH = PEOPLE (PERSONS), PARENTS (LINK),
19)                         CHILDREN (PERSONS).
20)              SOURCE RECORD PERSONS, INDEX = 3
21)                 NAME = NAME
22)                 SEX = SEX
23)  END NRL
```

Figure 7—NRL specification

is created in a manner similar to that of GRANDFATHERS, except that the accessing strategy is simpler and no selection criterion is required.

OPERATIONAL SOFTWARE

There are basically two approaches to implementing the modules of a data translator: generative and interpretive. In the generative approach each module generates an object program which, when executed, performs the desired function of the data translator. The advantage of this approach is that efficient machine code may be generated for each application or function. This efficiency may be irrelevant, however, since transformations are rarely executed more than once. The disadvantage of this approach is that two phases (code generation and module execution) are required to arrive at the final result.

The interpretive approach consists of a table-driven program that is applicable to all potential transformations. The disadvantage of this approach is the possible inefficiency of a general purpose interpreter. However, the program is easier to understand and debug.

Restructurer design decisions

Overall, the basic design objective of the Michigan Data Translator is to provide an operational software system for demonstrating, testing, and validating research results on generalized data translators. The primary design goal for the Restructurer module is the incorporation of general network restructuring capabilities.[15] Due to the prototype nature of the task, and to reduce the implementation time and enhance the experimentation and validation effort, the simplest and most straightforward algorithm was selected. Consequently, little emphasis was placed on execution efficiency. Efficiency development was left to future versions of the Michigan Data Translator.

The decision to use an internal DBMS as an implementation tool was made early in the design process. It reflects the need for an environment in which system tables are shared by translator modules, and change often in size and complexity. In addition, since the Michigan Data Translator was developed in a research environment, the table designs themselves were subject to change. Therefore, some degree of centralization, integrity control, and data independence for system tables was clearly necessary. The translator uses *ADBMS*, a DBTG-type DBMS developed at the University of Michigan by the ISDOS Project.[16] All databases, except the source and target databases, are ADBMS databases. They are manipulated by ADBMS "verbs" which take the form of subroutine calls.

Another design decision to provide generality and to ease the complexity of the Restructurer's implementation resulted in the development and use of an internal data form.[2,13,17] As summarized earlier, the Reader module in the Michigan Data Translator accesses the source database and converts it into the internal format called the source Restructurer Internal Form—source RIF. In addition to maintaining the logical structure of the source database, the RIF database has additional system access sets to every record type which facilitates fast access by the Restructurer.

Restructurer algorithm

The function of the Restructurer algorithm is to provide all the restructuring capabilities discussed in the first part of this paper. Since the Reader and Writer modules isolate the Restructurer from the source and target databases, the Restructurer need be tailored only to handle RIF databases. The Restructurer, then, is essentially on ADBMS-specific data translator and is directed by the contents of the NRL tables. The basic cycle of the Restructurer is divided into five phases as indicated in Figure 8.

During the first phase, Target Control, the Restructurer determines the next step in its traversal of the target logical structure. That is, a target set/record pair is selected for construction, as is a current access path to direct the construction process. The paths that the Restructurer takes in traversing the target database are called "construction paths." They are defined implicitly by the TARGET SET and TARGET RECORD statements supplied by the user in his NRL specification.

During the Source Accessing phase, the Restructurer

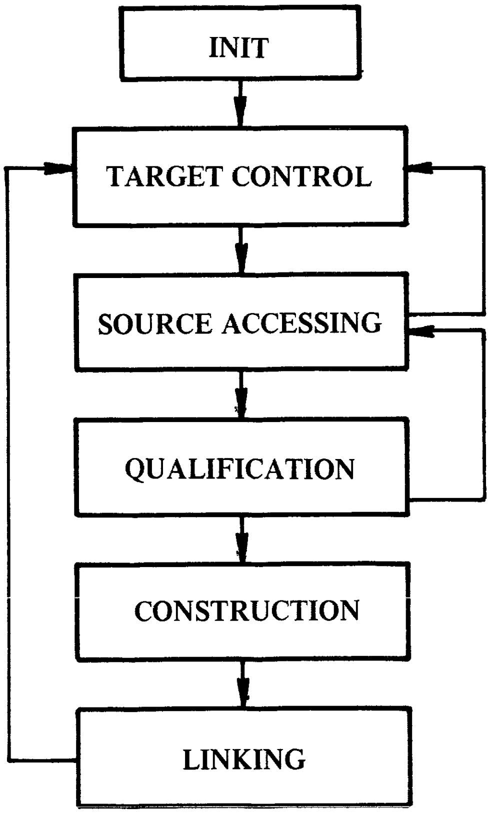


Figure 8—Restructurer block diagram

determines the next step in its traversal of the source RIF, as indicated by the current access path. In essence, an access path is a continuous list of source relations which indicates the location in the source of information required to create a target instance.

The purpose of Source Accessing is to exhaustively search for every potential target record occurrence that could be created from the records on the source access path. When a new record occurrence is found, the Restructurer enters the Qualification phase. If none is found, that is, when the access path has been exhaustively examined, the Restructurer returns to the Target Control phase.

It is during the Qualification phase that the Restructurer implements the user's "selection criteria." The source record occurrences on the current source access path are retrieved and all specific items to be qualified are tested against the values specified in the NRL. If *all* of the occurrences pass, the Restructurer enters the Construction phase. Otherwise, the Restructurer returns to the Source Accessing phase to find another access path.

During the Construction phase, source record occurrences along the current access path are retrieved. Data items are extracted, conversions are performed on them as necessary, and they are stored into a new target record occurrence.

The Linking phase is the most complex phase in the Restructurer cycle. Basically, its job is to connect the newly created target record occurrence on all appropriate target sets. This is no simple task because:

1. at any given moment, all potential parent record occurrences may not exist.
2. a record occurrence may be incomplete because it may obtain partial data from each of several access paths.

During Linking, the Restructurer relies on user-specified primary key items for determining whether instances are unique. Each time a target record occurrence is created, the Restructurer must check all the records in the record type and compare keys with the newly created record. As may be obvious, this is not efficient, but is necessary given this particular implementation. Upon completion of this phase, control is returned to the Target Control phase.

Restructurer implementation

The Restructurer was written primarily in ANSI FORTRAN to simplify coding and maximize program portability. In addition, some routines were coded in assembler language to perform specialized functions not easily accomplished in FORTRAN.

Approximately 5000 lines of FORTRAN code were required to implement the prototype Restructurer. It ran in 60K words on a Honeywell H6000, and required approximately 10,000 CPU seconds to construct a target RIF database containing 2500 record occurrences, an average of 4 seconds/record. Unfortunately, the Restructurer's execution time was found to be proportional to the square of the number of target record occurrences, and an average of 1500 seconds/record was projected for a target database of 150,000 record occurrences.

SUMMARY, PROBLEMS, AND CONCLUSIONS

Two approaches to the construction of restructuring languages and software for network databases have been presented. The elementary operations approach, although effective with hierarchical databases, does not extend well to network databases. The access path approach, in which all source structures are treated in the same way, is more naturally suited to complex network transformations, and powerful restructuring capabilities are obtained.

A Network Restructuring Language (NRL) based on the high-level specification of access paths was developed. It is essentially non-procedural, and its basic constructs—access paths and selection criteria—are familiar to users of network databases. These factors help to make it generally user-friendly.

The architecture of an operational NRL-driven Restructurer was discussed. The internal DBMS used as an implementation tool proved valuable in the construction of the Restructurer. It allows last-minute design changes to be incorporated easily and provides a facility for tuning internal data management.

Experience with the Restructurer has revealed two major problems. First, as noted above, execution efficiency was not a primary concern in its design. It was expected that the prototype Restructurer would be slow but practical to use, and that efficiency enhancements would be built into subsequent versions. Unfortunately, execution time was proportional to the square of the number of the target record instances. The core of the problem was the algorithm chosen for the Linking phase of restructuring, which led to a very large (approximately 3000 FORTRAN statements), very slow Linker module.

Secondly, although the construction of target record instances was easily specified in the NRL, the means by which target sets were established was quite opaque. In fact, it required a user writing NRL descriptions to understand certain features of the Restructurer algorithm. This difficulty compromised the NRL's claim to non-procedurality and user-friendliness.

Both of these problems have been corrected in a second version of the NRL and the Restructurer. The basic NRL structure remains unchanged although access paths are no longer required to be strictly linear—they may be tree-structured—and sets are explicitly established in a way that completely hides the restructuring algorithm from the user. Changes in the Restructurer's algorithm and enhancements to ADBMS, which maintains the source and target databases in their internal forms, have lowered execution time to a linear function of the size of the target database (with what appears to be an acceptable slope). The reader is referred to Reference 13 for theoretical details of the second version, and to Reference 18 for usage-oriented details.

We feel justified in concluding that, using the access path approach, it is possible to build practical, general-purpose restructuring specification languages and restructuring software for network databases. However, since restructuring involves the exhaustive traversal of the source database—with some or all of the source data accessed many times—as well as the complete construction of a target database, it is an inherently slow process, and serious attention must be paid to a restructuring system's execution efficiency.

REFERENCES

1. Novak, D. and J. Fry, "The State of the Art of Logical Database Design," *Proc. Fifth Texas Conference on Computing Systems,* Austin, Texas, October 1976, pp. 30-39.
2. 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,* Denver, Colorado, November 1972, pp. 77-106.
3. 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.
4. Housel, B., V. Lum, and N. Shu, "Architecture to an Interactive Migration Systems (AIMS)," *Proc. 1974 ACM SIGFIDET Workshop on Data Description, Access and Control,* Ann Arbor, Michigan, May 1974, pp. 157-169.
5. Rameriz, J. A., N. A. Rin, and N. S. Prywes, "Automatic Conversion of Data Conversion Programs Using a Data Description Language," *Proc. 1974 ACM SIGFIDET Workshop on Data Description, Access and Control,* Ann Arbor, Michigan, May 1974, pp. 207-225.
6. Birss, E. W., and J. P. Fry, "Generalized Software for Translating Data," *Proc. 1976 NCC,* Vol. 45. AFIPS Press, Montvale, New Jersey, pp. 889-899.
7. Fry, J. P. and D. Jeris, "Towards a Formulation of Data Reorganization," *Proc. 1974 ACM/SIGMOD Workshop on Data Description and Access,* New York, 1974.
8. Navathe, S. B. and J. P. Fry, "Restructuring for Large Data Bases," *ACM Transactions on Database Systems* 1, 2, June 1976, ACM, New York, 1976, pp. 138-158.
9. CODASYL DATA DESCRIPTION LANGUAGE COMMITTEE, *CODASYL Data Description Language Journal of Development,* June 1973, NBS Handbook 113, ACM, New York, January 1974.
10. Fry, J. P. and E. A. Sibley, "Evolution of Database Management Systems," *Computing Surveys,* 8, 1, March 1975, pp. 1-42.
11. Shoshani, A., "A Logical-Level Approach to Data Base Conversion," *Proc. ACM/SIGMOD International Conference on Management of Data,* ACM, New York, 1975.
12. Codd, E. F. and C. J. Date, "Interactive Support for Non-Programmers: The Relational and Network Approaches," *Data Models: Data-Structure-Set versus Relational,* R. Rustin (ed.) Ann Arbor, Michigan, May 1974, pp. 13-42.
13. Deppe, M. E., "A Relational Interface Model for Database Restructuring," Technical Report 76 DT 3, Data Translation Project, The University of Michigan, Ann Arbor, Michigan, 1975.
14. Deppe, M. E., and K. H. Lewis, "Data Translation Translation Definition Language Reference Manual for Version IIA Translator Release 1," Working Paper DT 5.2, Data Translation Project, The University of Michigan, Ann Arbor, Michigan, 1976.
15. Birss, E., M. Deppe, and J. Fry, "Research and Data Reorganization Capabilities for the Version IIA Data Translator," Data Translation Project Technical Report, February 1975, Graduate School of Business Administration, University of Michigan, Ann Arbor, Michigan.
16. Hershey, E. A. and P. W. Messink, "A Data Base Management System for PSA Based on DBTG 71," ISDOS Working Paper No. 88, July 1975, ISDOS Research Project, Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, Michigan.
17. The Stored-Data Definition and Translation Task Group, "Stored-Data Description and Data Translation: A Model and Language," Information Systems.
18. Bodwin, James, et al., "Data Translator Version IIA Release 2 User Manual," Technical Report 76 DT 3.4, Data Translation Project, The University of Michigan, Ann Arbor, Michigan, forthcoming.
19. UNIVAC, UNIVAC 110 Series Data File Converter, Programmer Reference, UP-8070, Sperry Rand Corporation, March 1974.
20. Bakkom, D. E. and J. A. Behymer, "Implementation of a Prototype Generalized File Translator," *Proc. 1975 ACM SIGMOD International Conference on Management of Data,* San Jose, California, May 1975, pp. 99-110.
21. Shoshani, A., "A Logical Level Approach to the Data Base Conversion," *Proc. 1975 ACM SIGMOD Conference,* San Jose, California, May 1975, pp. 112-122.
22. Shu, N. C., B. C. Housel, and V. Y. Lum, "CONVERT: A High-Level Translation Definition Language for Data Conversion," *Comm. ACM* 18, 10, October 1975, pp. 57-67.
23. Lewis, K., B. Driver, and M. Deppe, "A Translation Definition Language for Version II Translator," Working Paper 809, Data Translation Project, The University of Michigan, Ann Arbor, Michigan, 1975.

A multi-level procedure for design of file organizations

by EIVIND AURDAL and ARNE SØLVBERG
The University of Trondheim
Trondheim, Norway

ABSTRACT

This paper describes a multi-level procedure for design of file organizations. Necessary description of the application systems is discussed and a design procedure outlined. The main levels of the design procedure are:

- normalization of messages,
- synthesis of a logical model of the file organization,
- making a "best possible" physical realization of the logical model using a particular DBMS.

The final solution is evaluated through a performance analysis.

INTRODUCTION

The design of large data bases involves a wide range of problems, from the application system specification to the choice of hardware. The work reported here is restricted primarily to the problem of designing a "best possible" file organization given the application systems retrieval requirements, and given a particular database management system ~~(DBMS) for the implementation. A multi-level approach to~~ the design of file organizations is proposed.

Using a multi-level approach to the design of file organizations the following advantages can be achieved:

- transparency of the design procedure,
- elimination of non-effective solutions at an early stage,
- hardware/software selection can be postponed until the appropriate design level is reached.

The multi-level design procedure that is proposed here consists of

- specification of the information processing problem,
- transformation of the specified information structures into a logical model of the file organization,
- modification of the logical model to fit a particular DBMS,
- physical implementation of the modified model using a particular DBMS,
- evaluation of the final solution using a performance analysis.

The various levels of the design procedure are described with respect to the decisions which have to be made on each level, the necessary specifications of the application problem, and the DBMS specifications.

THE INFORMATION SYSTEM SPECIFICATIONS

We shall concentrate on that part of the information system specifications that are relevant to data base design. The specifications must model that part of the real world which we are interested in, the object system. The object system consists of a finite number of objects. An *object* is a unique part of the real world. It is something we are interested in, something we want information about. Objects may be concrete as well as abstract entities, like persons, enterprises, orders etc. Two kinds of features of an object are of interest: the properties of the objects and the relationships between the objects.

Objects of a system can be partitioned into object classes. An object class defines a set of objects which, for our purpose, are supposed to have so many common properties and relationships that we assign one common type of identifier to each object in the set.

Relationships between objects can be described by four different types of binary relations:

1:1 *one-to-one relation* describes a relationship between an object of one class and an object of the same or a different object class.

1:n *one-to-many relation* describes a relationship between one object of one object class and many objects of the same or a different object class.

n:1 *many-to-one relation* is the inverse of the 1:n relation described above.

n:n *many-to-many relation* exists if both the relation and its inverse are 1:n related.

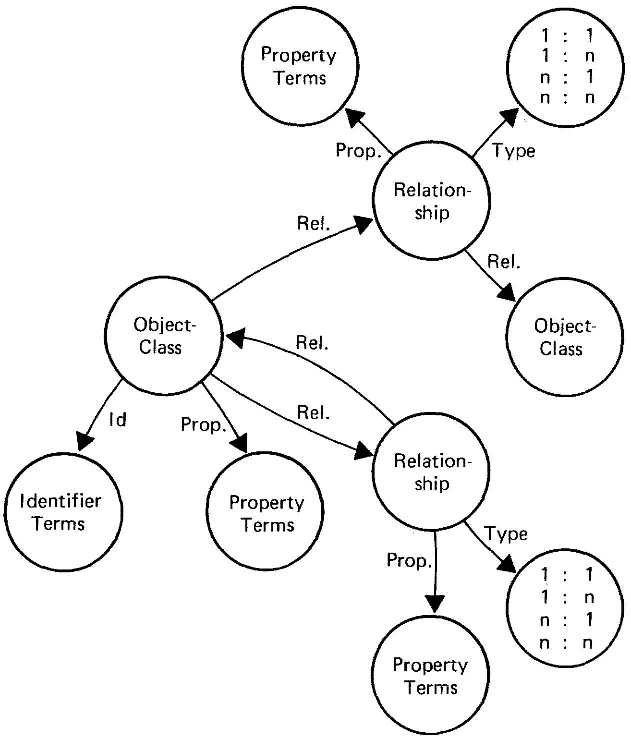


Figure 1—Features of an object

The different features of an object are illustrated in Figure 1. The figure shows how objects are related to other objects, and how identifier-items and property-items are attached to objects. The object description is illustrated with an example in Figure 2. The example shows that a customer may have a number of orders. The customer is identified by "customer no." and described by the property items "name" and "address." One order is identified by "order no" and has the property items "total sum" and "order spec."

In an information processing system, information about real world objects are stored in permanent files in a DBMS. Information about objects is exchanged between different parts of the information processing system by exchanging messages about those objects between processes. A convenient way of specifying this information flow may be obtained using the principle of hierarchical systems partitioning. This will lead to a level-by-level more detailed specification of the application system, where the terminal elements will be subprocesses and logical files.

We shall illustrate our proposal for a multi-level file-design procedure by a case, which will be a simplified model of a warehouse (Figure 3). The system handles customers' orders for goods, it controls quantity in stock, and it produces refill orders for the vendors.

Suppose that a further detailing of the ORDER MANAGEMENT system results in the subsystem structure of Figure 4. The logical files "STOCK FILE" and "CUSTOMER FILE" represent permanent information, while the logical files "ORDERS," "ORDER ACCEPTED," "ORDER REFUSED" and "ORDER REGISTERED" represent message exchange between processes.

The elements of the logical files can be described by message types.

If we suppose that one customer may give several orders, and that one order may consist of a number of orderlines, the CUSTOMER FILE elements can be specified by the message type:

MT(CUSTOMER FILE):
customer no, customer name, customer address,
⟨order no, total price,
⟨order line no, article no, article name,
line price, number ordered⟩⟩

where identifiers are underlined and brackets ⟨ ⟩ represent repeating groups.

Each of the permanent files have to be specified in this way. The permanent files are the basis for the design of the

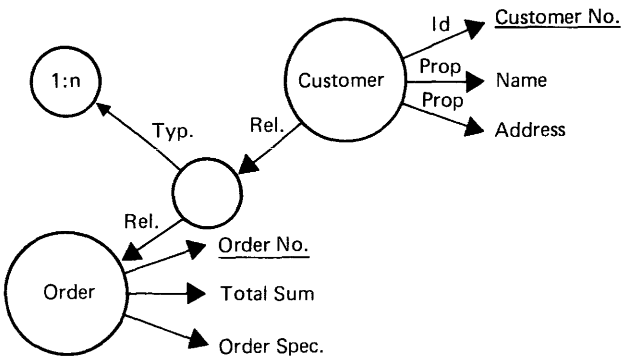


Figure 2

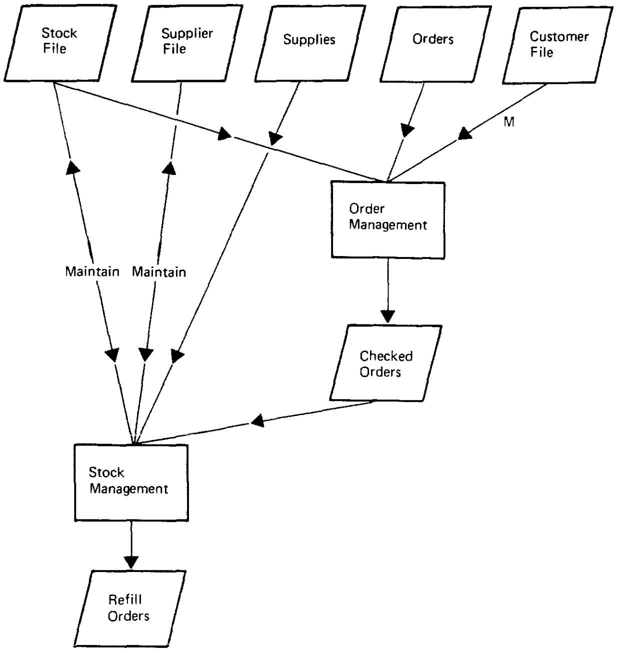


Figure 3

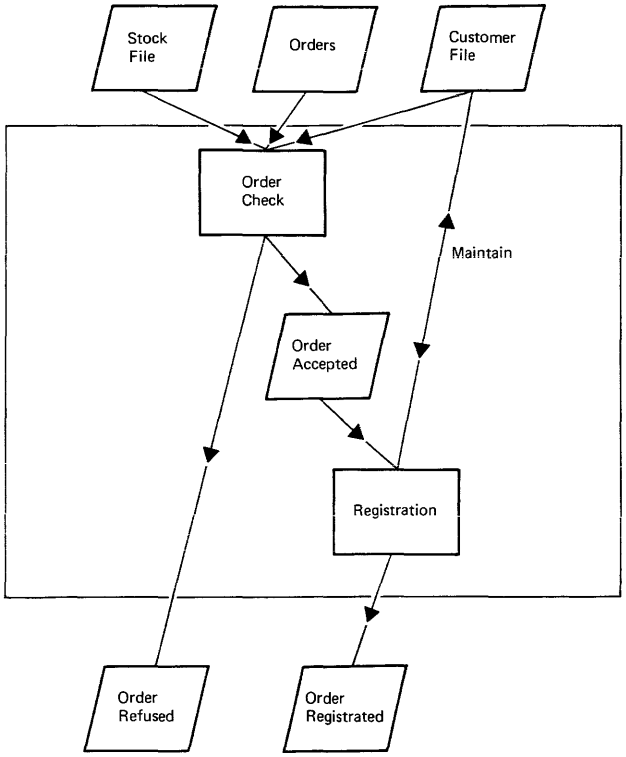


Figure 4—Order-handling system

file organization. Therefore, one has to describe the activity between them and the information system processes. This can be done with special types of processes, called *retrieval processes*. Each information system process may contain a number of retrieval processes.

Each of the retrieval processes must be one of the following four file operations:

READ,	the retrieval process reads element(s) from a permanent file.
UPDATE,	the retrieval process changes one or more property terms of an element of a permanent file.
WRITE,	the retrieval process stores new element(s) into a permanent file.
DELETE,	the retrieval process removes element(s) from a permanent file.

A retrieval process must have exactly specified search keys. The expected result must also be specified. This specification may be a description of which parts of the permanent file shall be read, or it may be a description of the message(s) which shall be stored.

The system of Figure 4 consists of two subsystems, ORDER CHECK and REGISTRATION. The subsystem ORDER CHECK handles orders from the customers. When an order arrives, the CUSTOMER FILE is checked to see if the customer is already registered. If not, some action has to be taken, for instance, to refuse the order, or to produce a customer number for this customer. Before the order can be accepted, an order number has to be produced, and the quantities in stock of the specified articles have to be checked. This subsystem will therefore contain two retrieval processes, one for checking the customer file, and one for checking the stock file. The keys are denoted by K (=key) and the expected result by O (=output).

S1: Check customer-file
Operation: READ
K: —customer name
O: —customer no.
S2: Check stock-file
Operation: READ
K: —article no.
O: —quantity

The system REGISTRATION stores new orders and possible new customers. The registration of new orders can be illustrated as follows:

S3: Order registration
Operation: WRITE
K: —customer no.
—order no.
O: —order no.
—total price
—order-line no.
—article no.
—article name
—line price
—number ordered.
} for each order-line

If all the retrieval processes defined by the application system are described in this way, then the application programs can be considered to consist of retrieval processes working towards a central data base containing permanent files (Figure 5). Through the specification of the application problem, we have now developed a basis for the design of a first logical model of the file organization.

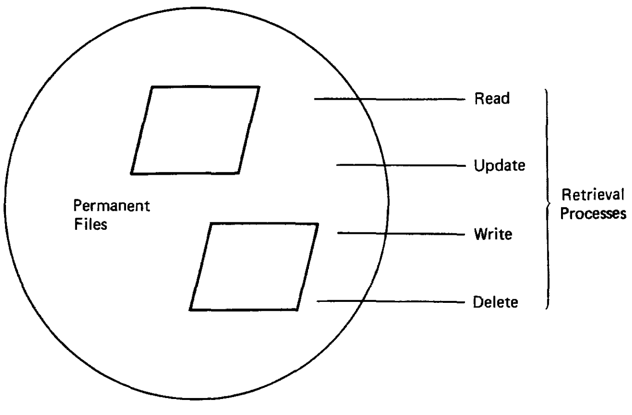


Figure 5—The basis for designing a logical data base model

DESIGNING A LOGICAL DATA BASE MODEL

A logical model of the file organization must satisfy the following requirements:

- It must be entirely based upon the information analysis (i.e., the a priori knowledge of the application problem).
- It must satisfy the requirements specified in the information analysis, i.e., the requirements of information structure and the requirements of retrieval.
- It must involve a "best possible" independence between the designed data structures and the specified application programs.
- It must be easy to realize using any particular DBMS.

The design of the logical model is done in three steps:

(1) Decomposition of the message types
(2) Synthesis of the decomposed messages into a model which satisfies the information requirements and finally
(3) A modification of the model in order to satisfy the retrieval requirements.

The decomposition procedure is similar to the normalization procedure proposed by Codd,[1] which results in an elementary file system.[2] The normalization of message types is a three-step procedure:

1. Elimination of repeating groups, hierarchical structures, or network structures,
2. Elimination of non-full dependence on the primary key,
3. Elimination of transitive dependence between the property terms.

The decomposition will be illustrated with an example. The permanent file CUSTOMER FILE, was described earlier in the following way:

$M1_0$: *customer no,* customer name, customer address, ⟨order no, total price, ⟨orderline no, article no, article name, line price, number ordered⟩⟩

In this case a repeating group is subordinate to another repeating group, and the first step is to eliminate the first repeating group. The message type is therefore split into the message types:

$M1_1$: *customer no,* customer name, customer address
$M1_2$: *customer no, order no,* total price, ⟨orderline no, article no, article name, line price, number ordered⟩

The description of an order is only dependent on "order no," not "customer no." The message type $M1_2$ is therefore split into:

$M1_{21}$: *customer no, order no*
$M1_{22}$: *Order no,* total price, ⟨orderline no, article no, article name, line price, number ordered⟩

An elimination of the last repeating group results in:

$M1_{221}$: *order no,* total price
$M1_{222}$: *order no, orderline no,* article no, article name, line price, number ordered

In message $M1_{222}$ there is a transitive dependence between the property terms. "article name" is dependent on "article no," not the identifier of the message. The message type is therefore split into:

$M1_{2221}$: *order no, orderline no,* article no, line price, number ordered
$M1_{2222}$: *article no,* article name

Using this kind of normalization procedure, the message type $M1_0$ has been decomposed into the five message types $M1_1$, $M1_{21}$, $M1_{221}$, $M1_{2221}$ and $M1_{2222}$. The decomposition was entirely based upon specification of the application systems requirements, i.e., information about objects and message types. Normalization is a reversible process. That means that the original message types can be reconstructed, and therefore no information has been lost.

In the example of Figure 3, the system contains, in addition to the CUSTOMER FILE, also the permanent files STOCK FILE and SUPPLIER FILE. Suppose that the permanent file STOCK FILE contains messages which describe the various articles in stock, and that each article may have several substituting articles and several suppliers. A possible decomposition might be:

$M2_1$: *article no,* article name, quantity, price
$M2_2$: *article no, article no*
$M2_3$: *article no, supplier no*

Suppose the SUPPLIER FILE contains messages which describe the various suppliers and that one supplier may deliver several articles. That supplier may also have several refill orders registered. A possible decomposition might be:

$M3_1$: *supplier no,* supplier name, supplier address
$M3_2$: *supplier no, article no*
$M3_3$: *supplier no, refill order no*
$M3_4$: *refill order no,* number wanted, article no

Some of the message types are binary relations which represent relationships between objects. For example: $M1_{211}$: *customer no, order no,* represents a binary relation between the objects of the object classes CUSTOMER and ORDER. Other messages may describe properties of an object, for example $M1_1$: *customer no,* customer name, customer address.

Examining the message types, one may determine the relationships between the various messages. This will be illustrated with examples:

(1) The message type
$M1_{211}$: *customer no, order no,*
describes an 1:n relationship between messages identified by "customer no" and messages identified by "order no."
(2) In the message type
$M3_4$: *refill order no*, number wanted, article no,
the secondary key, "article no," is used as a property term. This indicates an: 1 relationship between messages defined by "refill order no" and "article no."

The result of such an examination is illustrated in Figure 6. One may observe that there are both a 1:n relationship and a n:1 relationship between "article no" and "supplier no." This implies the existence of a n:n relationship.

The logical model may be represented in a convenient way, using a data structure diagram. The data structure diagram has two basic elements; boxes and arrows. Each box represents a class of records (or a file) and each arrow represents a meaningful relationship between records. Such a diagram may easily be drawn using the normalized message types. Message types may be represented as boxes, and a 1:n relationship is an arrow between boxes. A n:n relationship is represented as a coupling (or a relational file) between boxes.

A data structure diagram based upon the normalized message types of the permanent files CUSTOMER FILE, STOCK FILE and SUPPLIER FILE is shown in Figure 7. This is a logical model which satisfies the requirements of the information structure.

	Customer No.	Order No.	Order No. Orderline No.	Article No.	Refill Order No.	Supplier No.
Customer No.		1:n				
Order No.			1:n			
Order No. Orderline No.				n:1		
Article No.				1:n		1:n
Refill Order No.				n:1		
Supplier No.				1:n	1 :n	

Figure 6—Relationships between identifiers of the normalized messages

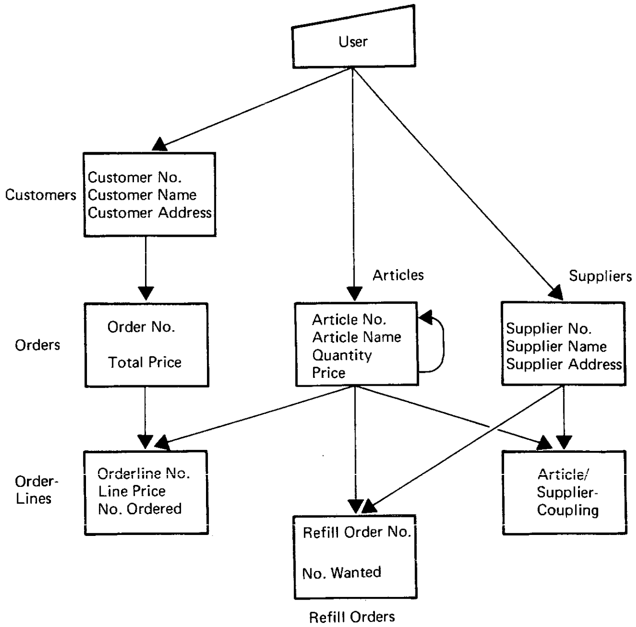


Figure 7—Data structure diagram of the normalized information structure

The next step of the design procedure is aimed at satisfying the retrieval requirements. Suppose that both "customer no" and "customer name" are used as keys by different retrieval processes, and that the response time requirements exclude a sequential processing of the CUSTOMER FILE. A modification of the data structure diagram is then necessary (Figure 8).

A search file consisting of "customer name" is established and 1:n-related to the customer file. Usually, the retrieval requirements may be satisfied establishing search files in a way similar to the one illustrated in Figure 8.

Suppose further examinations of the retrieval processes show that the items "article no.," "article name," "supplier no" and "supplier name" are retrieval keys. Necessary modifications of the logical model may result in a data structure diagram as shown in Figure 9.

A logical model has now been developed in three design steps. The first step, decomposition of messages, was based on message description and object description. The synthesis of the decomposed messages was based upon relation-

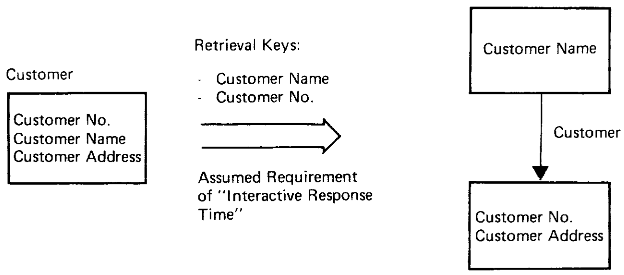


Figure 8—Modification of CUSTOMER FILE to satisfy retrieval requirements

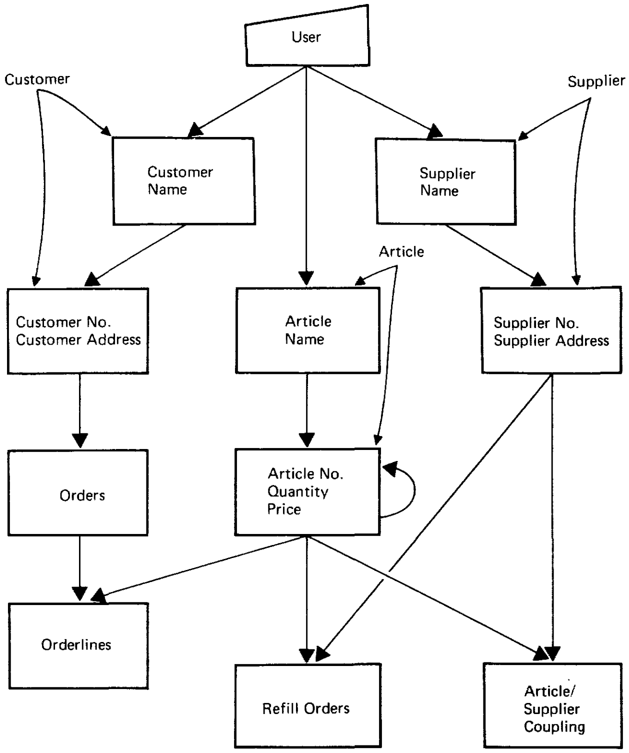


Figure 9—Logical model which satisfies the retrieval requirements

ships between objects, and the resulting model was modified, using information about the retrieval processes and the response time requirements. Thus, a logical model of the file organization has been developed, without making any decisions about any particular DBMS. The entire design process is reversible. Thus, the original information structure can be reconstructed, and the requirements stated in the information analysis is therefore satisfied.

FITTING OF THE LOGICAL MODEL TO A PHYSICAL MODEL

The next step in the design procedure is to select a particular DBMS, and modify the logical model in order to give a "best possible" physical solution. In this paper the efforts will primarily be devoted to DBTG-like DBMS-software products, but most of the problems will nevertheless be of a general character.

The available DBMS may have certain restrictions, like:

- the DBMS does not allow network-structures, only tree-structures,
- the DBMS allows direct access to only a limited number of hierarchical levels,
- the DBMS allows only a certain number of hierarchical levels
- combinations of the restrictions mentioned above.

Therefore, the first modification of the logical model is to satisfy the DBMS restrictions. Substitutions of lists by inverted lists will usually solve these problems.

The next step is to make certain adjustments of the logical model in order to:

- minimize the number of block accesses,
- minimize the data volume,
- minimize the transport of data between memory and the data files.

Unfortunately, these factors are conflicting, and an operation which takes all the factors into consideration must be carried out. One may also notice that the less complex a file organization is, the more it simplifies such operations as initial loading, reorganization and report generating.

The logical model describes the record design and the choice of access paths. Any adjustment of the logical model must therefore either modify the record design or the choice of access paths. A modification of the record design which has to satisfy the information structure requirements must be a choice between storing of duplicate data item values or the use of references. The modification of the access paths is a choice between the use of lists or inverted lists. Duplicate storing of data will be illustrated as shown in Figure 10.

The present version of the logical model describes the customers as illustrated in Figure 10(a). Suppose many retrieval processes are frequently using "customer no" as key and want information about "customer name." Such a situation will result in a large number of accesses from the member records to the owner records. An alternative version of the record layout is illustrated in Figure 10(b). The item "customer name" is stored in both the owner records and the member records. Such a solution will result in a decrease in the number of accesses, an increase in the file size, and an increase in the data read and written. In order to find the best solution one has to estimate the decrease in access costs versus the increase of input/output and storage costs.

Consequently, at this level of the design procedure, there is a need for an analysis tool which may be used in making such estimates. The analysis tool must satisfy the following

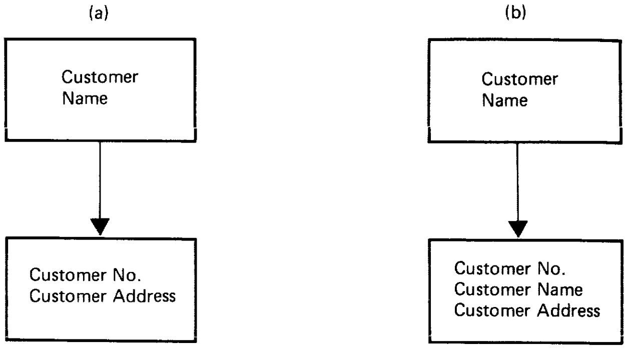


Figure 10—Alternative layouts of the customer file

conditions:

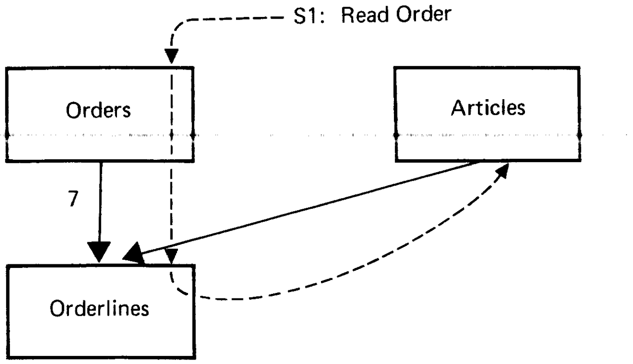
- it must be possible to perform an analysis without making any decisions about physical realization.
- the analysis must be based on a description of the logical model and the retrieval processes.
- the analysis must form a basis for making decisions of the type mentioned above.

The logical model describes the file organization, and the retrieval processes describe the activity against the file organization. Consequently, it is possible to describe the activity between the different record types and the activity between records within the same record type. It will be shown later that such a description is useful in making decisions about logical restructuring or about physical realization.

A part of the order management system described in the previous sections is shown in Figure 11. Suppose the following retrieval process exists:

S1: Read order
Operation: READ
K: —order no
0: —order no
—total price
—orderline no }
—article no } for each
—article name } orderline
—line price }

The process uses "order no." as key, and its access path is illustrated with a dotted line in Figure 11. One may observe that the process results in transitions between the record types ORDERS and ORDERLINES and between ORDERLINES and ARTICLES. These transitions are called retrievals of the first kind. An order will usually consist of more than one orderline, and there will be transitions between the records in the record type ORDERLINES. These transitions are called retrievals of the second kind. Suppose the average number of orderlines per order is seven. The retrieval process S1 then will result in one retrieval of the first kind between user and ORDERS, the same between ORDERS and ORDERLINES, six retrievals of the second kind within the record class ORDERLINES and finally seven retrievals of the first kind between ORDERLINES and ARTICLES.



S1: Read Order

K : - Order No.
O : - Order No.
- Total Price
- Orderline No. }
- Article No. } For Each
- Line Price } Orderline
- Number Ordered }

Figure 11—Illustration of the access path of a retrieval process

In order to describe the activity in the data base, one has to know the frequencies of the retrieval processes, and the scattering factors which are a result of transitions between the different record types. A transition from an owner record type to a member record type results in a scattering factor equal to the average number of member records per owner record. It should be obvious that transitions from user to a record type and transitions from a member record type to an owner record type result in a scattering factor equal to one.

The activity in the data base may be represented in a matrix (see Figure 12):

$\{\omega_{ij}\}$, $i=0,1,\text{---}, n$ and $j=1,2,\text{---},$n

where

n is the number of record types
ω_{oj} is the number of retrievals of the first kind from user to record type no. j.
ω_{ij} $i=1,2,\text{---},n, j=1,2,\text{---},n$ and $i\neq j$ is the number of retrievals of the first kind from record type no. i to record type no. j.
ω_{jj} is the number of retrievals of the second kind within record type no. j.
$\omega_{T.j}=$ $\sum_{i=o}^{n} \omega_{ij}, j=1,2,\text{---},n$ is the total number of retrievals of record type no. j.

Suppose the retrieval process S1 is initiated 100 times a day. The example described in Figure 11, then results in the retrieval matrix shown in Figure 13.

The behavior of the retrieval processes will be described with a few more examples. Figure 14 shows the logical model designed in an earlier section. The numbers within the boxes are the numbers of the specific record types and the numbers on the arrows are the scattering factors for transitions from an owner record to the member records. The access paths of four retrieval processes are drawn as dotted lines. The retrieval process S1 has been described earlier, and a description of the remaining processes shown in Figure 14, may be as follows:

S2: Read customer name
Operation: READ
K: —customer no
O: —customer name

From \ To	1	2	---	j	---	n
0	ω_{01}	ω_{02}				ω_{0n}
1	ω_{11}	ω_{12}				
2						
i	-------			ω_{ij}		
n						ω_{nn}
	$\omega_{T \cdot 1}$	$\omega_{T \cdot 2}$				$\omega_{T \cdot n}$

n is the number of record classes

ω_{oj} is the number of retrievals of the first kind from user to record class no. j.

ω_{ij} i = 1, 2, - - -, n, j = 1, 2, - - -, n and i ‡ j is the number of retrievals of the first kind from record class no. i to record class no. j.

ω_{jj} is the number of retrievals of the second kind within record class no. j.

$\omega_{T \cdot j} = \sum_{i=0}^{n} \omega_{ij}$, j = 1, 2, - - -, n is the total number of retrievals of record class no. j.

Figure 12—Retrieval matrix

The process results in one retrieval of the first kind between USER and CUSTOMERS and the same between CUSTOMERS and CUSTOMERNAMES.

S3: Update quantity
Operation: UPDATE
K: —article no
O: —quantity

The process updates the number of articles in stock, and results in one retrieval of the first kind from USER to ARTICLES.

S4: Remove refill order
Operation; DELETE
K: —refill order no
O: —textstring (deletion OK/not OK)

The process removes a refill order and deletes the relationships to ARTICLES and SUPPLIERS. The result is one retrieval of the first kind from USER to REFILL ORDERS, one from REFILL ORDERS to ARTICLES and one from REFILL ORDERS to SUPPLIERS. Suppose that, in addition to the retrieval processes already mentioned, there are other retrieval processes which complete the user's communications with the data base. A possible retrieval matrix may be the one in Figure 15.

One may observe that ω_{46}=3990 and ω_{64}=0. This indicates that "article no" and "article name" also should be stored in ORDERLINES. The decrease in the number of retrievals has to be paid for with an increase of the data volume.

Such a modification of the logical model results in the data structure diagram shown in Figure 16. Further examinations of the retrieval matrix may lead to other modifica-

To / From	Orders	Orderlines	Articles
User	100		
Orders		100	
Orderlines		600	700
Articles			
Total	Σ	Σ	Σ

Figure 13—The contributions from the retrieval process S1 to a retrieval matrix

tions of the logical model and thus the final solution will be fitted to the selected DBMS in a "best possible" way.

~~PHYSICAL~~ IMPLEMENTATION

To / From	1	2	3	4	5	6	7	8	9	10
0	150	195	570		155	495	5	20	11	
1		150								
2	195	150								
3		380		570						
4				3420		3990				
5						155				
6					1535	75		45		130
7									5	
8						20			55	
9							31			1
10						10			1300	1177

Figure 15—Complete retrieval matrix

The final logical model describes the final record design and the final access paths. The next step in the design procedure is a ~~physical~~ implementation of the logical model. This process consists of four main steps:

- choice of storage structures for the record types,
- physical location of the records,
- implementation of the relationships between the records,
- determination of the file dimensions.

The physical implementation is dependent on the particular DBMS and consequently it is not possible to formulate any general rules to solve the problem. Therefore, in this paper,

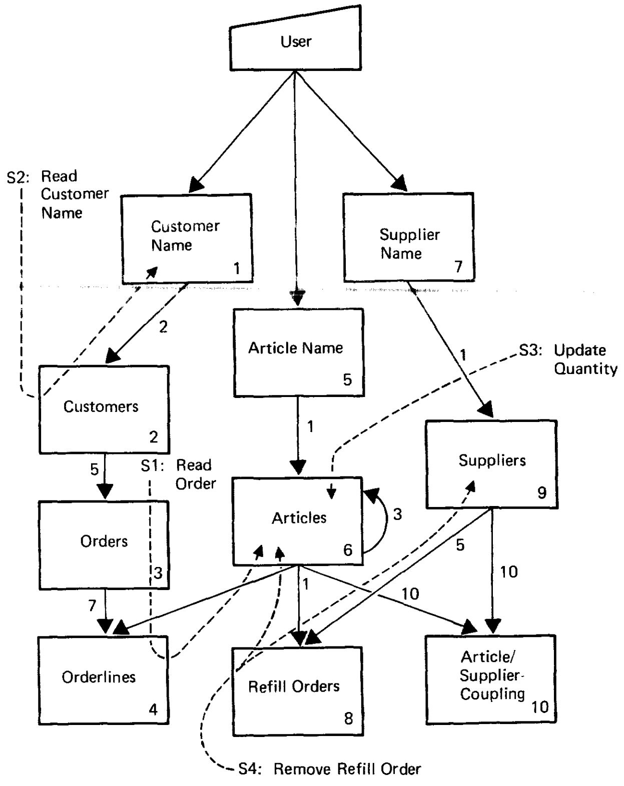


Figure 14

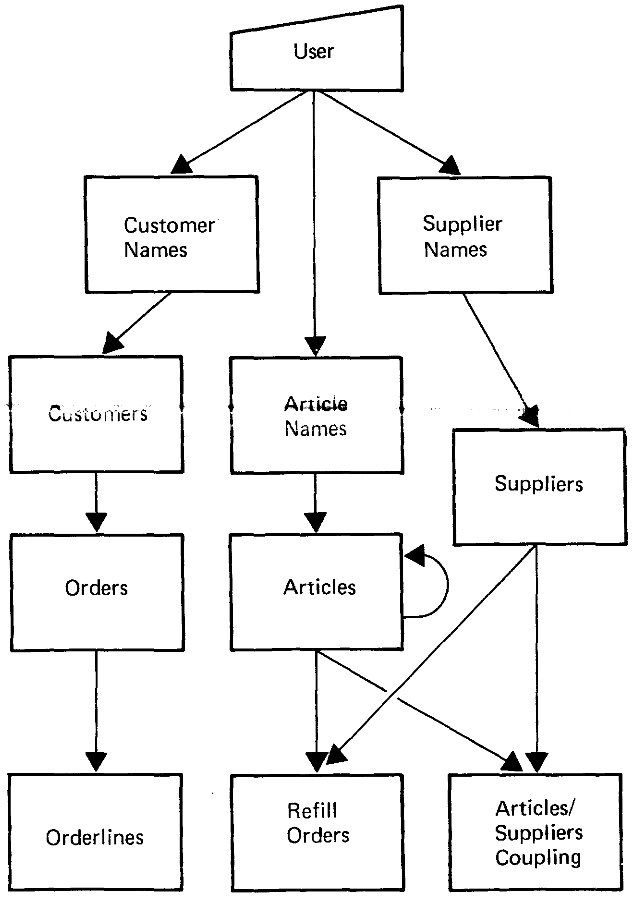


Figure 16

it will be shown how the problems can be solved using a DBTG-DBMS.

Choice of storage structure

A DBTG-DBMS allows the following types of storage structures:

- a direct structure (DIRECT)
- a randomized structure (CALC)
- an index sequential structure (INDEX)
- a sequential structure (VIA)

Usually, there will be no problem in making a choice between these structures. For example a choice between direct access and sequential access is determined by the response time requirements. If direct access is necessary, one should use a direct structured file if possible. If not, an index sequential or randomized structure has to be used. A randomized structure is usually cheaper than an index sequential structure. Therefore, if there is no sequential searching a randomized structure should be used. Of course, these rules are not of general validity, but they may be used in most cases.

Physical location

The physical location may, to some extent, be controlled by the following statements:

(i) $$\left\{ \mathit{WITHIN}\ \text{area-name-1} \left[\begin{Bmatrix}\text{integer-4}\\ \text{data-base-data-name-4}\end{Bmatrix} \begin{Bmatrix}\mathit{THRU}\\ \mathit{THROUGH}\end{Bmatrix} \begin{Bmatrix}\text{integer-5}\\ \text{data-base-data-name-5}\end{Bmatrix}\right]\right\} \ldots$$

The statement may be used to specify in what physical data file(s) the various record types are to be placed.

(ii) *LOCATION* MODE IS *VIA* set-name-1 SET
If a sequential record type has more than one owner record type, this statement may be used to place the member records close to one of the owner records.

(iii) *INTERVAL* IS integer-3 PAGES
This statement may be used to place member records close to their owner record, for example in the same physical block.

The retrieval matrix describes the activity between records and it may therefore be a useful tool in determining the physical locations of the records. High activity between certain records implies that they should have physical locations close together.

Implementation of relationships

The implementation of relationships between records (i.e., set types) are done via list structures in a DBTG-DBMS.

A set may be organized as a one-way or two-way list using the statement

MODE IS *CHAIN* [LINKED *PRIOR*]

The choice of list structure has to be based on the description of the retrieval processes. High activity from particular member records to their owner record indicates the use of a pointer from the member records to their owner record. This can be realized using the statement

[LINKED TO *OWNER*]

Determination of the file dimensions

The final step is to determine the dimensions of the data files, i.e., determine the parameters:

- file volumes
- volumes of overflow areas
- volumes of index tables
- load percents
- block sizes

A "best possible" choice of values for the parameters mentioned above is dependent on the particular DBMS and the particular hardware used. Consequently, such decisions should be made in connection with a performance analysis. This will be further discussed in the next section.

PERFORMANCE ANALYSIS

An evaluation of the final solution should be done using a performance analysis. The most convenient criterion of a good solution is low total cost per time. Various types of costs which affect the choice of file organization are:

- storage costs
- access costs
- data transfer costs
- sorting costs
- reorganization costs

In addition there will be costs for using cpu and memory.

In order to get a better picture of the performance analysis procedure, a simplified case, the one-identifier case, will be considered. In the one-identifier case the entire file organization consists of a single file with records consisting of one identifier item and one or more property items.

In order to develop a cost function, it is necessary to have information about

- the retrieval activity
- the storage structure and the dimensions of the file
- the accounting routine of the particular machine system

The first step is to decide on a storage structure. For a particular storage structure one may decide on block size and load percent. Those decisions will usually be results of local optimizations, depending on the particular computer system that is used. In addition, it is necessary to estimate the volume of internal housekeeping data in the file. Thus, the file may be described by the following parameters:

- number of records
- record size
- block size
- load percent
- size of overflow-area
- volume of housekeeping data

Based on this description, one may estimate the average retrieval length (measured in block accesses). How this can be done for the most common used storage structures is described in Reference 3. Estimates of the average use of memory and CPU for each retrieval process must also be made. The retrieval processes are described with the following parameters:

- frequencies
- use of memory
- use of CPU
- retrieval lengths

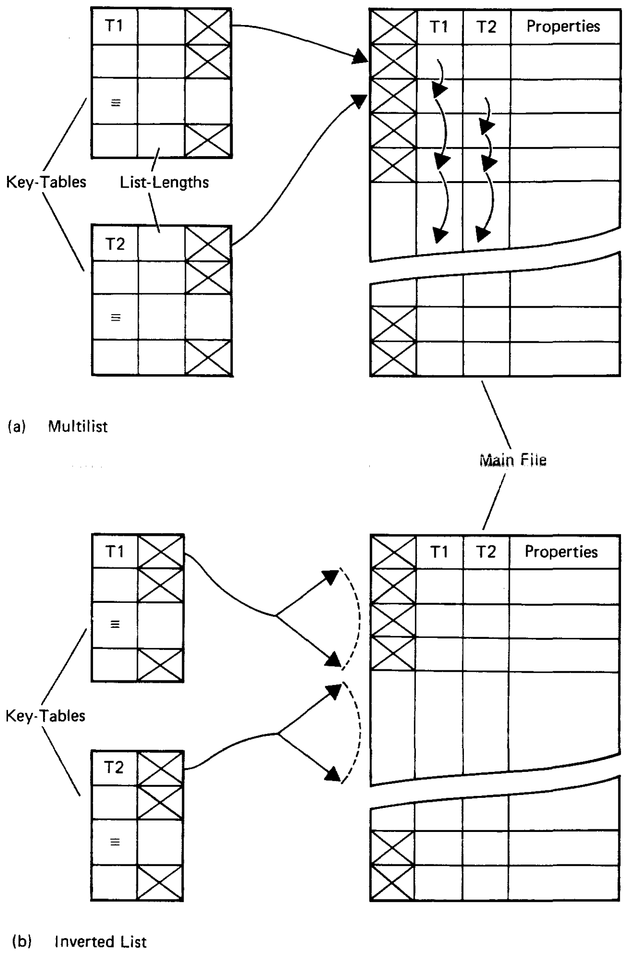


Figure 17—Multilist and inverted list

The parameters mentioned above are used as parameters in the accounting routine of a particular machine system, and a price may be computed.

It is then possible to adjust the file parameters, for example the block size, and look at how these adjustments influence the total price. Thus, a minimization of the cost function may be done in an iterative way.

Searching in the multi-identifier case means that logical conjunctions between the separate identifier items have to be done. The most common ways to implement conjunctions are multilists or inverted lists.

Figure 17(a) illustrates a multilist. T1 and T2 are terms which are parts of the identifier. A multilist consists of key-

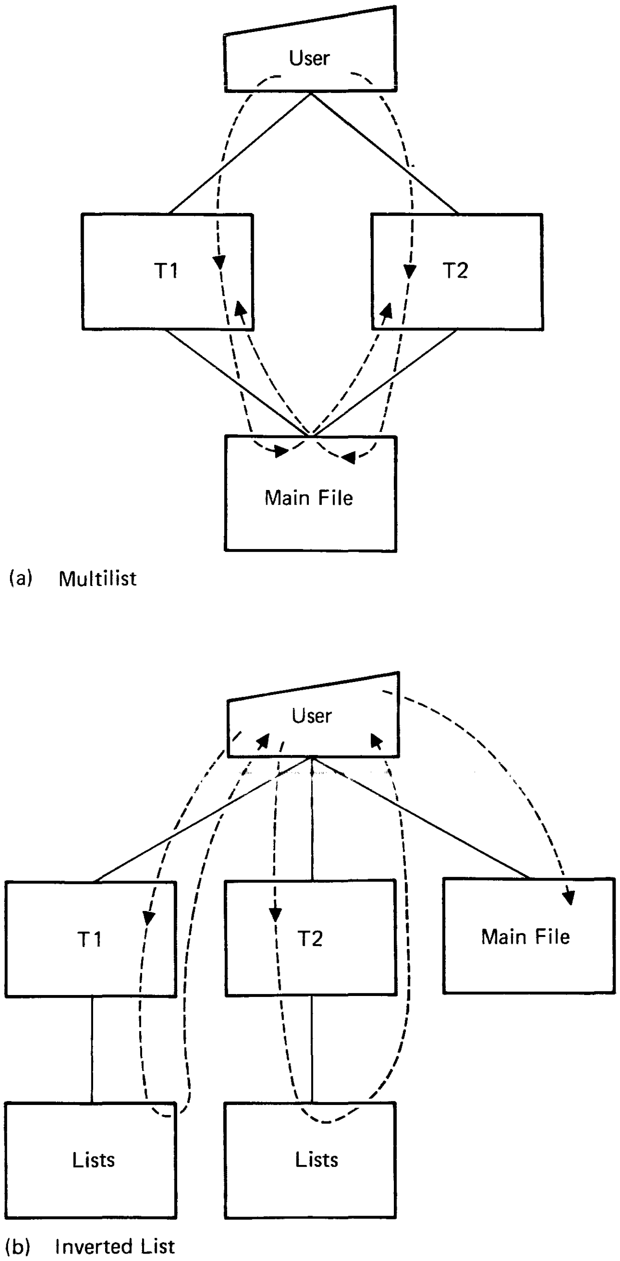


Figure 18—Multilist and inverted list in a data structure diagram

tables and a main file. A key-table consists of one record for each key value; the property parts of the records contain a list address and a list length. The search method is to find the list addresses in the key-tables and then search through the shortest list. The retrieval length will be the number of accesses to the key tables plus the number of accesses to the main file.

Figure 17(b) illustrates an inverted list. An inverted list consists of key tables and a main file. The result of searching in a key table is a record of variable length which contains all the addresses of the records in the main file having specified key value. The search method is to find the record addresses in the key-tables; then logical functions are computed in memory, and the remaining record addresses are found in the main file. The retrieval length will then be the number of accesses to the key tables plus the number of accesses to the main file.

These list structures consist of separate one-identifier-case problems—searching in key-tables and searching in the main file. Therefore, it is possible to develop a cost function using the same parameters as for the one-identifier case.

Fig 18 describes the multilist and the inverted list in a data structure diagram. The access paths are drawn with dotted lines. From data structures of the kinds shown in Figure 18, it is possible to construct any complex data structure, and any complex data structure can be decomposed into simple list structures. It should then be obvious that a performance analysis of a multi-identifier case may be performed using the same parameters as for the one-identifier case. One should, however, notice that the access paths of the retrieval processes have to be described, and that the retrieval length, in the multi-identifier case, may be written as:

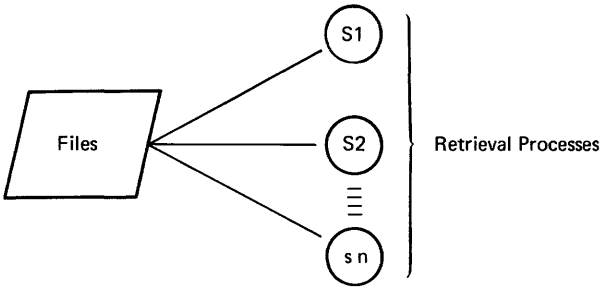
$$a = a_1 + a_2$$

where

a_1=the number of block accesses caused by the retrieval keys

a_2=the number of block accesses caused by the establishing of relationships

Figure 19 gives a summary of the necessary parameters.



(i) Retrieval Process Parameters:
- Frequencies
- Retrieval Lengths
- Use of CORE
- Use of CPU

(ii) File Parameters:
- Number of Records
- Record Volume
- Block Size
- Load Per cent
- Volume of Administration Data

(iii) Parameters Which Describe a Particular Accounting-Routine

Figure 19—Necessary parameters to carry out a performance analysis

CONCLUSION

In the previous sections a multi-level design procedure for design of file organizations has been developed. The design procedure was based on an information analysis of the application system. The information flow was described using the principle of hierarchical systems partitioning, and necessary information was extracted from the information analysis:

- description of objects and relationships between objects,
- description of permanent files, i.e., message types,
- description of retrieval processes.

A logical model was now developed in three steps:

- the message types of the permanent files were normalized,
- a synthesis of the normalized message types into a logical model was made,
- the model was modified in order to satisfy the retrieval requirements.

So far no decisions about any DBMS have been made. The logical model was now modified to fit a particular DBMS in a "best possible" way, i.e., modifying the record layouts and altering the access paths. The record layout was modified making a selection between duplicate storing of data or the use of a reference, and the access paths were altered by making selections between lists and inverted lists.

Therefore, a physical realization in four steps was done, i.e.:

- choice of storage structure,
- physical location of the records,
- implementation of the relationships between the records,
- determination of the file dimension.

The file dimensioning was done in connection with a performance analysis.

The design procedure outlined in this paper still needs a detailed description of the various levels, and further work has to be done in this area.

An experience during the work already done was the strong need for computer based tools for documenting information structures etc., and for analysis of DBMS-schema performance properties. The basis for such a computer-based tool is an automatic documentation of the information analysis. Such documentation systems exist today, for example, CASCADE, which with a few minor adjustments would be well fitted for this purpose. A transformation of the application system into a logical data base model will be the next step. It seems that the method outlined in this paper may be a useful basis for such a transformation algorithm.

Concerning the modification of the logical model, an analysis tool has to be developed. With the information documented through the analysis of the application system it seems possible to develop an analysis tool based upon activity between the records.

Finally, the physical realization using a particular DBMS has to be done. It seems that this may be done in connection with a computerized performance analysis.

Considering the obvious possibilities of automating the design procedure, there seems to be good hope of achieving good results in this area in the future.

REFERENCES

1. Codd, E. F., "Normalized Data Base Structure: A Brief Tutorial," 1971 ACM Sigfidet Workshop, *Data Description, Access and Control,* edited by E. F. Codd and A. L. Dean.
2. Langefors, B., *Theoretical Analysis of Information Systems,* Third Edition, Studentlitteratur, Lund, 1971.
3. Bratbergsengen, Hofstad, Wibe: "Filsystemer og databaser."

An effective method for measurement and analysis of system software performance

by JOHN R. RUMSEY and DAVID W. ABMAYR
Harris Corporation
Dallas, Texas

ABSTRACT

The performance measurement and analysis of software operating systems which extend basic computing machinery is discussed. The description of an external monitoring technique which facilitates the correlation of hardware events with software functions without the need for software monitors is presented. A time related Event is defined to provide the basis for the technique used to implement the monitor system. In addition, Event analysis methods are introduced which allow a software system execution profile to be constructed.

INTRODUCTION

The existence of complex software systems to extend basic computing machinery is well established and procedures for implementing these systems are well known. Many of these extended machines, however, are dynamically responsive so that a predictable system execution path does not always exist. The system path is normally a function of the occurrence of a real-time event or the result of a user system resource request. The occurrence of either of these events is usually asynchronous to the system software. ~~Considering the unpredictable and complex~~ nature of system software execution paths, the empirical determination of these paths for the evaluation of extended machine performance is generally considered a complex and time consuming task.

Several techniques have been proposed for measurement and analysis of the performance of extended machines. Nutt[2] surveys the three classical types of computer system monitors: pure software, pure hardware, and hybrid monitor systems. Generally, hardware monitors are considered to be most adept at detecting a wide variety of events but are limited in their ability to relate the events to a specific software function. Software monitors, on the other hand, are better suited for identifying the source of an event than responding to a large class of events. Svobodova[3] indicates that system measurability can be achieved only with an integrated hardware-software approach in which the external monitoring device complements the internal monitor.

To developers of operating system software, the need to identify and analyze various areas of the software is most important. Items of interest include task dispatching overhead, the response time required to support a disk access, and many other similar functions. It seems, however, inappropriate to burden system software development with the added task of developing and maintaining software monitoring support. Similarly, a hybrid monitor system requires imbedded software to identify certain key functions and has limited flexibility. A technique, therefore, is needed for externally monitoring system software execution.

The objective of this paper is to describe an external monitoring technique which facilitates the correlation of hardware events with software functions without the need for any software monitors, and allows more than one view of machine instruction execution. To completely characterize machine execution, the gathering of several different types of data is required. The data types include:

(1) Time related execution state events—to facilitate analyzing program execution
(2) Selected address references—to allow reference counts to be accumulated
(3) Instruction operation codes—to allow analysis of the ~~distribution of instructions executed~~
(4) Instruction execution and address reference traces—for analyzing branch patterns and to aid software debugging.

Each data type provides a different view of the execution of a sequence of machine instructions. This allows the implementation of a comprehensive program analysis.

EVENT DEFINITION

Conceptually it is desirable to monitor the execution of software with no impact on the system whatsoever. This includes the elimination of any background trace functions or special instructions to identify the occurrence of particular events. To accomplish this goal, it is necessary to define an autonomous device that is capable of capturing states of execution associated with the software system. This, of course, could be an overwhelming task if no method could be devised to concisely represent the set of possible execution states. An approach to this problem has been defined which effectively reduces the set of execution states to a manageable size.

Consider that any function in a program can be associated with one or more address references. To completely classify an address reference, however, it is necessary to identify the reference type, to define the data associated with the address, and to distinguish multiple references with a time value. A set of primitive elements for defining execution states, therefore, can be used to define Events in terms of an address reference five-tuple.

Event=E=(A, R, D, C, T)
where
$A=(a_1, a_2, a_3, \ldots, a_n)$ is an address space
$R=(r_1, r_2, r_3, \ldots, r_j)$ is a non-empty set of address references
$D=(d_1, d_2, d_3, \ldots, d_k)$ is a set of data types associated with an address
C=(Fetch, Read, Write, . . . , I/O) is a set of reference classes
T is a time interval in which two consecutive address references contained in R must be guaranteed to occur
Any Event$_t$ occurs at real-time modulo T.

A profile of the execution of any program can be described in terms of Events. The granularity of the profile is controlled by the magnitude of the Event set. Figure 1 illustrates an application of the Event definitions. A memory map of a skeletal representation of an operating system and associated asynchronous processes is shown. The address space of interest in this example contains the operating system and its associated job scheduling process. The selected Events segment the operating system into specific functions, but the granularity of coverage is very loose. The auxiliary Event associated with clock service guarantees that an Event will always occur within the Event time interval T. Since an Event$_t$ is defined to always occur within an interval real-time modulo T, a time value may be associated with each Event. Events, therefore, not only provide execution path data but also provide sufficient information to determine the time required to execute the functions described by the profile. Therefore, if the address space is defined so that all software functions are included, then the Event set will allow direct determination of processor utilization for any function of interest. For a particular execution of the software, the Events illustrated in Figure 1 are sufficient to describe the relative utilizations in the defined address space.

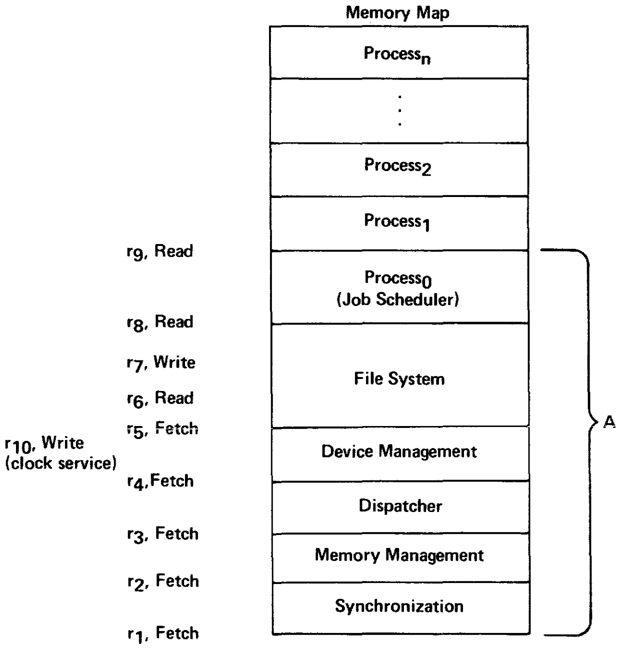


Figure 1—Typical event set

The determination of the absolute address associated with each Event of interest may seem to be an arduous task. This is not the case, however, because Events may be defined in terms of relative displacements associated with the set of software functions to be measured. These relative addresses are then easily converted to match a particular implementation.

IMPLEMENTATION

The implementation of the mechanism required for gathering the various data types is based primarily on a function $f_i(E)$ with a domain consisting of a set of time related events (E). A set of data is collected by applying f_i to E. In particular, for the four data types previously described, the functions are defined as follows:

$f_1(E_i)$ = insert a_i and real-time modulo T
into the data set if
$a_i \in R$
and $c_i \in C_j$
where $C_j \in C \oplus C$*

$f_2(E_i)$ = counter$_i \leftarrow$counter$_i$+1
if
$a_i \in R$
and $c_i \in C_j$
where $C_j \in C \oplus C$

$f_3(E_i)$ = counter$_j \leftarrow$counter$_j$+1
if
c_i=Fetch
where $j=f(d_i)$ which identifies the instruction code

$f_4(E_i)$ = insert a_i into the data set
if
c_i=Fetch

* Notation for unordered cross product taken from Reference 1.

The implementation of these functions requires that the monitoring system must be capable of capturing addresses, verifying reference types, and associating time values with the address data. A device to accomplish these functions is not difficult to implement, consisting only of a logical four-way switch, simple address translation, incremental counting, and the associated memory and function generator. A simplified block diagram is provided in Figure 2.

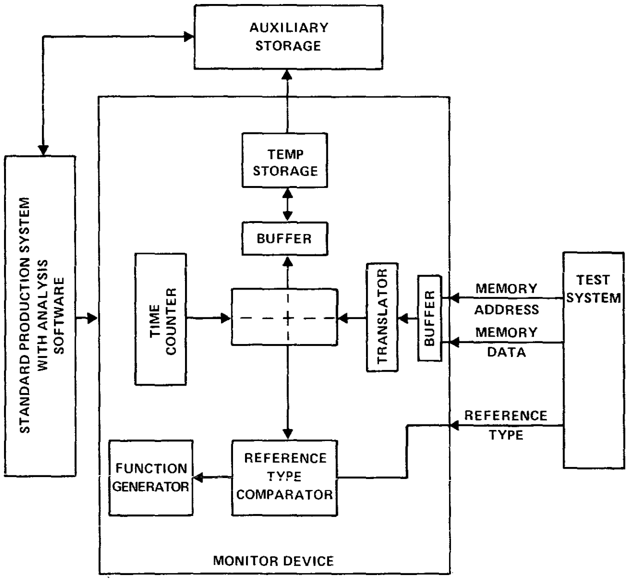


Figure 2—Monitor block diagram

The monitor device is interfaced to a standard production system which specifies the data path in the monitor and performs the analysis of the stored data. Since address, data and reference type can be captured from the Test System, each of the required data types can be accumulated. The function f_1 is implemented by translating monitored memory addresses to specify an appropriate reference type comparison. The Type Comparator, which is dynamically programmable, provides indication of a match between address and reference type. If a match occurs the address data and the current value of the time counter are inserted into a queue in Temporary Storage. Subsequently the queued data is written to Auxiliary Storage. The second function f_2 is also implemented by translating monitored memory addresses. In this case, however, the translated address specifies the location of a counter in Temporary Storage which is to be incremented by one. Function f_3 is similar to f_2 since both specify that a Temporary Storage Counter is to be incremented. The third function, however, invokes translation of memory data if and only if the reference type is a Fetch. The address generated by the Translator becomes a function of the instruction operation code. The last function f_4 is a special case of f_1. The address (without time counter) is queued if the reference type is a Fetch.

EVENT ANALYSIS

Experience with the monitor system has shown that the most extensively used function is f_1, i.e., gathering Events for profiling system execution. While the method for accumulating Events is interesting, the techniques used to analyze the Event sequence to extract system utilization and performance information may be more significant.

For the purpose of analysis, each Event in a sequence of Events may be considered to represent a start, end, or single Event. Start and end Events are used to define intervals, the durations of which are of interest. A single Event has no interval connotation but implies merely a count of occurrences of some address reference by specific type. With respect to intervals, several things must be considered:

(1) Intervals may be nested. Let A represent a set of start Events and B represent a corresponding set of end Events. Then it is permissible for A_j to occur simultaneously with or after A_i but before B_i, provided B_j occurs concurrently with or before B_i. The allowable nesting level depends upon the total number of single Events and Event pairs. The following example illustrates timing calculation. Suppose the following Event sequence occurs:

$$\begin{array}{ccccccccccccccc} A_1 & & A_2 & & B_2 & & A_3 & & A_4 & & B_4 & & B_3 & & B_1 \\ | & \Delta t_1 & | & \Delta t_2 & | & \Delta t_3 & | & \Delta t_4 & | & \Delta t_5 & | & \Delta t_6 & | & \Delta t_7 & | \end{array}$$

Then the intervals associated with each Event pair are as follows:

Pair	Interval Time
A_1 B_1	$\Delta t_1 + \Delta t_3 + \Delta t_7$
A_2 B_2	Δt_2
A_3 B_3	$\Delta t_4 + \Delta t_6$
A_4 B_4	Δt_5

(2) An Event may define

a) the start of n intervals
b) the end of n intervals
c) the start of one interval and the end of another interval
d) the start and end of an interval

Where A_i starts several intervals, the interval count is accumulated only until the first end occurs and is associated only with that specific start-end pair.

(3) The percent utilization associated with an interval is calculated as follows:

If T_{ij} denotes the ith interval time associated with the jth Event pair and only the jth Event pair, then the percentage of time spent in the jth interval is

$$\% = \frac{\sum_i T_{ij}}{\sum_j \sum_i T_{ij}}$$

Considering the above analysis rules, an effective method of determining performance is based upon constructing a profile of system activities.[4] The process of defining a system profile consists of selecting appropriate Events associated with the software system. These Events may then ultimately be used to define intervals of time which collectively account for the total real-time associated with a particular period of system operation. After selected programs are characterized with the profile, performance characteristics may be calculated.

In addition, a data profile may be defined which is a subset of the system profile. A data profile is generated when a data sample accumulated with a system profile is analyzed. Various data profiles may be generated from a single data sample. The one-to-many mapping from system profile to data profile facilitates multiple analyses of a single data sample. For example, data profiles may be generated for selected combinations of Events in the system profile to allow analysis of processor utilization, interpretive instruction execution time, disk system physical access time, etc.

When a data profile is generated certain information relating to intervals is accumulated. The minimum, average, and maximum interval times are maintained with a count of the number of times each interval was entered. In addition, a multipoint interval time distribution is calculated which provides the basis for typical weighted interval residence times.

An example of a system profile is provided in Figure 3. Each item of coverage implies that two Events must be defined to represent the associated interval with the exception of count only items which are implicitly defined by single Events. Associated with each address is the reference type designation.

INTERVAL DEFINITIONS Entry Exit	SOFTWARE FUNCTIONS
F50A,01,F51C,02	USER PROCESS
8C96,02	OPERATING SYSTEM OVERLAY LOADS
80CC,02,80CE,01	PUNCH INTERRUPT SERVICE
7B90,02,7B92,01	TAPE INTERRUPT SERVICE
7820,02,7822,01	PRINTER INTERRUPT SERVICE
7798,02,769A,01	DISK INTERRUPT SERVICE
7404,04	DISK START I/O
7146,02,7148,01	READER INTERRUPT SERVICE
4E4A,04	OVERLAY REQUESTS
440C,02,440E,01	SYSTEM CLOCK
1E1C,01,1E2E,02	OPERATING SYSTEM PROCESS 1
1062,01,1D74,02	OPERATING SYSTEM PROCESS 2
1CC4,01,1CD6,02	OPERATING SYSTEM PROCESS 3
1C4E,01,1C60,02	OPERATING SYSTEM PROCESS 4
1AA8,01,1ABA,02	OPERATING SYSTEM PROCESS 5
19C6,01,1908,02	OPERATING SYSTEM PROCESS 6
1938,01,194A,02	OPERATING SYSTEM PROCESS 7
18D6,01,18E8,02	OPERATING SYSTEM PROCESS 8
1860, 01,1872,02	OPERATING SYSTEM PROCESS 9
0B06,04.0A22,04	WAIT
0AC2,04,0B44,04	DISPATCH
0A8A,04,0AC2,04	SCHEDULER SEGMENT 1
0A84,04,0ABA,04	SCHEDULER SEGMENT 2
0A7E,04,0A70,04	SCHEDULER SEGMENT 3
0A74,04,0A70,04	SCHEDULER SEGMENT 4
0A70,04,0AC2,04	SCHEDULER SEGMENT 5
0A60,04,0A70,04	SVC HANDLER
00BA,01	SVC 15
00B8,01	SVC 14
00B6,01	SVC 13
.	.
.	.
.	.
009C,01	SVC 0

Figure 3—System profile

A data profile may be generated with any subset of the system profile. Utilization summaries extracted from the various data profiles allow an extensive performance analysis. For example, considering a disk based operating system, it is of interest to determine the distribution of the time required to physically access the disk. This is easily accomplished by generating a data profile containing an interval which begins when a Start I/O to the disk is issued and ends when the first instruction of the corresponding interrupt service is executed. A data profile is illustrated in Figure 4. The interval counts have a dimensional value of two microseconds.

Data profiles allow a comprehensive look at any part of the system software. Figure 5 illustrates a summary of the performance of one of the access methods supported by the File System. Data profiles similar to Figure 4 were used to compile the statistics.

The Event analysis method described is certainly not unique. Function f_1 generates a sequence of Events that are written to auxiliary storage. The analysis of these Events is in no way confined to the method described; however, this technique has proven to be one of the most useful. An alternative analysis method has been used which does not consider intervals as nested. The time for any interval is accumulated from the first instance of the start Event until the corresponding end Event is detected. This alternate technique is useful for determining the frequency of occurrence of intervals.

OTHER DATA TYPES AND ASSOCIATED ANALYSES

From the viewpoint of performance analysis, the second most utilized statistics have been derived from machine instruction operation code data. A simple but very useful analysis counts the occurrence of each operation code type. When the data sample is exhausted, a summary is generated indicating the percentage of execution associated with each instruction. Weighted instruction execution times are calculated which clearly define processor utilization in terms of each machine instruction. Excessive use of high cost instructions is quickly identified.

The benefits of generating an address trace concurrent with the execution of system software are immediately

INTERVAL SUMMARY

ENTRY	EXIT	MAX COUNT	MIN COUNT	AVG COUNT	ENTRY COUNT	%TOTAL
7698	7404	31766	409	8610	969	037.08
7404	7698	40279	1174	14604	969	062.92

INTERVAL RESIDENCE DISTRIBUTION

ENTRY	EXIT	MIN GROUP	MIN-AVG GROUP	AVG GROUP	AVG-MAX GROUP	MAX GROUP
7698	7404	409-3142	3142-5875	5875-16328	16328-24046	24046-31766
		040.86%	016.61%	020.04%	014.65%	007.84%
7404	7698	1174-5650	5650-10126	10126-23162	23162-31720	31720-40279
		009.59%	008.66%	079.05%	000.64%	002.06%

Figure 4—Data profile

		System	Buffering Level		Avg Disk Accesses		Avg Access Time[3]	
Functions	Transfer Rate	Utilization	Input	Output	per Transfer	Avg Access Rate[2]	Physical	Software
DATA → DISK	551.56 RPM[1]	28.59 %	0	0	2.34	21.55	28.25	6.24
DATA → DISK	581.69	19.30	0	1	0.42	4.03	24.56	11.77
DATA → DISK	579.87	18.51	0	2	0.22	2.16	30.08	18.10
DISK → DISK	381.49	23.54	0	0	3.83	24.34	33.68	5.02
DISK → DISK	638.25	24.42	0	1	1.89	20.08	40.17	5.41
DISK → DISK	744.57	27.21	0	2	1.72	21.29	36.54	4.80
DISK → DISK	452.80	22.07	1	0	2.86	21.61	38.53	4.72
DISK → DISK	478.36	22.48	2	0	2.68	21.40	38.68	4.78
DISK → DISK	1087.45	26.72	1	1	0.95	17.21	45.30	3.17
DISK → DISK	1360.39	27.79	2	2	0.56	12.62	60.30	2.95

[1] RPM – Records per minute

[2] Accesses per second

[3] Milliseconds

Figure 5—File system performance statistics

apparent to programming personnel. Performance data, however, related to programming methodology can also be extracted from address data. Consider the simple problem of searching an array A for a value x. Examples 1a and 2 of Reference 2 provide two solutions with differing efficiencies. The second is considered the more efficient but it also may execute a greater number of Branch instructions. The increase in the number of Branch instructions executed may or may not have an effect on performance, depending upon the environment. For example, if the architecture of the processor executing the software is based on a pipeline technique, then it is possible that a high frequency of Branch instructions could certainly affect system performance. Branch analysis of address trace data, therefore, could reasonably be used to predict performance problems.

CONCLUSIONS

The performance monitoring technique described has been implemented and successfully used to evaluate the performance of a general purpose, disk based operating system and related asynchronous processes. The most extensively utilized operating system functions were quickly identified. Critical real time dependent functions were isolated and the functions for which execution times were marginally acceptable were rewritten to provide sufficient safety margins. Each release of the operating system or associated applications is currently profiled to ensure consistently reliable extended machine performance.

REFERENCES

1. Korfhage, Robert R., *Discrete Computational Structures*, Academic Press, Inc 1974.
2. Knuth, Donald E., "Structured Programming with go to Statements," *Computer Surveys*, Vol. 6, No. 4, December 1974, pp. 261–301.
3. Nutt, Gary J., "Tutorial: Computer System Monitors," *Computer*, Vol. 8, No. 11, November 1975, pp. 51–61.
4. Svobodova, L., "Computer System Measurability," *Computer*, Vol. 9, No. 6, June 1976, pp. 9–17.
5. Warner, Dudley C., "System Performance and Evaluation—Past, Present and Future," *AFIPS Conference Proceedings*, Vol. 41, Part II, 1972, pp. 959–964.

The Navy Fortran validation system

by PATRICK M. HOYT

Department of the Navy
Washington, D. C.

ABSTRACT

The FORTRAN Compiler Validation System (FCVS) developed by the Department of the Navy tests the conformance of those elements of the FORTRAN language which are contained in the logical intersection of the American Standard FORTRAN, X3.9-1966, and the elements proposed for the subset language in the draft proposed American National Standard Programming Language FORTRAN.

This paper discusses the development of the FORTRAN Compiler Validation System and presents the rationale for the FCVS. The design criteria for the FCVS and a description of the test production is explained. The capabilities of the Executive System are described as well as the future developments anticipated for the FCVS because of the adoption of the revised FORTRAN Standard and the impact of the CODASYL FORTRAN Data Base Facility.

INTRODUCTION

FORTRAN is one of the oldest of the higher level programming languages with its roots in IBM in 1954.[1] Standardization for the FORTRAN language began in May 1962 under the direction of the American Standards Association Committee X3.4.3.* In 1966, two standards were published for the FORTRAN language: American Standard FORTRAN, ~~X3.9-1966[2] and American Standard Basic FORTRAN,~~ X3.10-1966, which is a proper subset of the first Standard.

The FORTRAN Compiler Validation System (FCVS) developed by the Department of the Navy tests the conformance of those elements of the FORTRAN language which are contained in the logical intersection of the American Standard FORTRAN, X3.9-1966, and the elements proposed for the subset language in the draft proposed American National Standard Programming Language FORTRAN.[3]

One of the principal reasons for developing validation systems is the principal criteria given for developing the FORTRAN Standard: "Interchangeability of FORTRAN programs between processors."[3] The FCVS was developed as a tool to enable users to acquire FORTRAN compilers which meet the ANSI language specifications. The availability of FORTRAN compilers conforming to the Standard enhances the interchangeability of FORTRAN programs.

The FCVS consists of FORTRAN audit routines, their related test data, and an executive routine (EXECUTIVE) which prepares the audit routines for compilation and execution. Each audit routine consists of series of tests of FORTRAN language elements, and supporting procedures which indicate the result of executing these tests. Because the routines were designed to run on any computer system purporting to support FORTRAN, the assumptions used to write the audit tests are very restrictive. Only the simplest forms of GO TO, Arithmetic IF, WRITE, and assignment statements are used to write the support code required for each test. A complete discussion of the FCVS test philosophy and a full description of each of the language element tests are contained in the document FCVS DETAILED TEST SPECIFICATIONS.[4]

A SOURCE PROGRAMS file of audit routines with appropriate implementor-defined parameters inserted into the source code is produced by the EXECUTIVE. The EXECUTIVE is a FORTRAN program included in source form in the FCVS LIBRARY. Once installed, the EXECUTIVE is used each time that an audit routine or series of audit routines is selected from the FCVS LIBRARY. Basic inputs to this process are the FCVS LIBRARY (a file of all of the audit routines, the EXECUTIVE and related test ~~data), and a series of control inputs to select and/or update~~ the audit routine source code.

A FORTRAN compiler, in a particular computer configuration/operating system environment, is tested by the compilation and execution of each audit routine. If a compiler rejects some language element by giving fatal diagnostic messages or terminating the compilation, then the EXECUTIVE is used to eliminate the source code containing that language element. The audit routine is then recompiled and executed. Output reports (TEST RESULTS) produced by the execution of each routine indicate whether the code generated by the compiler passed or failed each test of the routine. The TEST RESULTS together with the compilation listings constitute the raw data from which the Department of the Navy produces a Validation Summary Report (VSR). The VSR itemizes the areas where the FORTRAN compiler being tested does not conform with the American National Standard FORTRAN specifications.

* The American Standards Association (ASA) has since changed its name to the American National Standards Institute, Inc. (ANSI). The FORTRAN Committee is now known as X3J3.

HISTORY

A study of available FORTRAN validation systems was performed in August 1973. This study analyzed the U. S. Navy FORTRAN tests developed by Captain Grace Hopper of the Navy Programming Languages Section,[5] and the National Bureau of Standards FORTRAN tests developed by F. E. Holberton and E. G. Parker.[6] The study concluded that the major flaws in these validation routines were that all the test results were listed on a printer and required careful examination of the test results by the user, and these test routines required many manual changes to the source code when preparing them for execution on a given computer system.

At this time it was decided that the FCVS developed by the Software Development Division must evaluate the results of the language tests within the tests themselves, and print PASS or FAIL for each test in the same manner as the COBOL Validation System. In 1973 a three stage project was designed to:

(1) extract and modify existing tests and routines;
(2) add PASS/FAIL/DELETE support code to make the routines self-measuring; and
(3) build a complete FORTRAN validation system based on a set of simple assumptions and the self-measuring techniques used in implementing the second stage.

Due to lack of available resources, the FCVS project remained in abeyance until February 1975 when the decision was made to pursue the third stage as the initial effort. The scope of the FCVS project was to adequately test all of the elements of the FORTRAN language based on the specifications in American Standard FORTRAN, X3.9-1966.

DEVELOPMENT OF THE FCVS

The FCVS project was broken into five major phases as follows:

1. Systems Analysis and Design Phase—
   - develop the matrix of language elements to be tested
   - develop the list of basic test assumptions, programming and naming conventions, EXECUTIVE routine functions and requirements, test and implementation procedures.
2. Program Analysis and Design Phase—
   - produce detailed specifications for each audit routine.
3. Coding and Debugging Phase—
   - write boiler plate for TEST RESULTS format
   - code three programs to test the basic assumptions
   - code and debug an estimated thirty elementary routines
   - code and debug an estimated twenty advanced routines
   - test data to be prepared as required.
4. Integration and Testing Phase—
   - write detailed specifications for the EXECUTIVE, then code and debug the EXECUTIVE routine
   - integrate the EXECUTIVE, audit routines and any test data onto the FCVS LIBRARY
   - test the final integrated FCVS as a system.
5. Documentation and Release Phase—
   - update all documentation to reflect final FCVS specifications then release and distribute through NTIS.

Based on this scope of the FCVS project, eighteen (18) manmonths were estimated for completion of the project. Two computer specialists were assigned to share equally the responsibilities of the entire project. It was estimated, based on the experience gained in developing the CCVS74 audit routines, that the two computer specialists could devote half of their available time to the project. The FCVS project was to begin October 1975 and was scheduled for completion on July 1, 1976.

Work proceeded on schedule until January 1976. Very little progress was made on the FCVS during January and February 1976 as the available manpower was devoted to higher priority projects. In March 1976, two major decisions were made. The number of tests in any one routine were limited to thirty (30), since the TEST RESULTS report could then be printed on a single page (approximately 56 lines). The draft proposed American Standard FORTRAN (X3J3—pending), which had been distributed for public comment, was analyzed with respect to the language elements identified in American Standard FORTRAN, X3.9-1966. It was decided that the FCVS version 1.0 then being developed would test the conformance of those elements of the FORTRAN language which are contained in the logical intersection of American Standard FORTRAN, X3.9-1966, and the elements proposed in the subset language of the draft proposed American Standard Programming Language FORTRAN. The previous arbitrary classification of elementary versus advanced language elements was deemed obsolete since the proposed Standard contained a subset language.

The FCVS was designed to build the statement tests from a basic set of FORTRAN language features which are assumed to function correctly. The remaining language features are tested using these basic language elements. The assumptions were made with the goal that these routines would be executable on most minicomputer systems as well as on the larger computer configurations.

The basic assumptions are listed below and the references to X3.9-1966 are enclosed in parentheses.

(1) Six character symbolic names (3.5 and 10.1) and five digit statement labels (3.4) are permitted.

(2) Comment lines (3.2.1) do not affect a program in any way.

(3) Execution of the unconditional GO TO statement (7.1.2.1.1) GO TO k causes the statement identified by the statement label k to be the next statement executed.

(4) Branching to a CONTINUE statement (7.1.2.6) causes the statement following the CONTINUE statement to be the next statement executed.

(5) The assignment statements (7.1.1.1)

integer variable = integer constant (5.1.1.1)
integer variable = integer variable
real variable = real constant (5.1.1.2)
real variable = real variable

function correctly.

(6) The arithmetic IF statement (7.1.2.2) functions correctly: IF (e) k1, k2, k3 where e is an arithmetic expression (6.1) of the form

integer variable + integer constant
integer variable − integer constant
real variable + real constant
real variable − real constant

and k1, k2, and k3 are statement labels.

(7) The simple formatted WRITE statement (7.1.3.2.3) functions correctly: WRITE (u,f) k where u is a logical unit number (7.1.3.1), f is a FORMAT statement label, and k is a list (7.1.3.2.1) of integer and real variables.

The format statement contains nH Hollerith field descriptors (7.2.3.8), nX blank field descriptors (7.2.3.9), Iw numeric field descriptors (7.2.3.6.1), and Fw.d numeric field descriptors (7.2.3.6.2).

(8) In order for the output report to have the correct format, the use of the first character of a formatted record for vertical spacing must function correctly (7.1.2.4).

Two characters which are used in printing the report are:

CHARACTER	VERTICAL SPACING BEFORE PRINTING
1	Advance to first line of next page
blank	One line

In addition to the preceding basic assumptions, the following minimum capabilities are assumed for the routines:

(1) Integer variables consist of at least 16 bits of which one is a sign bit.

(2) The system output device has at least 56 characters per line.

(3) Real variables contain at least 16 bits in the mantissa and 8 bits in the exponent.

In order to appreciate the changes in scope made during the FCVS project, it is essential that one understands what was considered elementary versus an advanced audit routine in the original identification of the tasks. The following list shows the language element areas originally chosen for elementary vs. advanced.

Also shown is a column for whether a given language element area was tested in version 1.0 of the FCVS.

LANGUAGE ELEMENT AREA	ORIGINAL LEVEL	VERSION 1.0
Comment lines	Elementary	Tested
Reference format blanks in variables statement labels	Elementary	Tested
continuation of lines	Elementary	Tested
FORTRAN reserved words	Elementary	Tested
Simple Subroutine call	Elementary	Tested
Subroutine calls another routine	Elementary	Tested
Intrinsic functions	Elementary	Tested
DATA statement	Elementary	Tested
BLOCK DATA subprogram	Elementary	Deferred to a later version
Blank COMMON	Elementary	Tested
Labeled COMMON	Elementary	Tested
EQUIVALENCE statement	Elementary	Tested
EQUIVALENCE and COMMON	Elementary	Tested
DO loops—simple format	Elementary	Tested
CONTINUE statement	Elementary	Tested
Arithmetic IF statement	Elementary	Tested
Logical IF	Elementary	Tested
Unconditional GO TO statement	Elementary	Tested

LANGUAGE ELEMENT AREA	ORIGINAL LEVEL	VERSION 1.0
Assigned GO TO	Elementary	Tested
Computed GO TO	Elementary	Tested
TYPE statement	Elementary	Tested
Integer arithmetic tests	Elementary	Tested
Arrays—fixed dimensions, simple constant and variable subscripts	Elementary	Tested
Repeated calls to a subroutine	Elementary	Tested
Inline arithmetic statement functions	Elementary	Tested
FUNCTION subprogram	Elementary	Tested
Multiple RETURN statements	Elementary	Tested
Logical data	Elementary	Tested
Logical expressions	Elementary	Tested
Simple sequential file I/O	Elementary	Tested
Character set	Elementary	Tested
Subroutines sharing COMMON	Advanced	Added**
Nested DO loops and extended range of a DO statement	Advanced	Added**
DO index tests	Advanced	Added**
Real arithmetic tests—adjustable accuracy	Advanced	Deferred
Double precision data	Advanced	Deferred
Complex data	Advanced	Deferred
Arrays—arithmetic expressions for subscripts	Advanced	Deferred
EQUIVALENCE with COMMON and DIMENSION arrays	Advanced	Added**
EXTERNAL statement	Advanced	Deferred
REWIND	Advanced	Added**
ENDFILE	Advanced	Added**
BACKSPACE	Advanced	Added**
READ	Advanced	Added**
WRITE	Advanced	Added**
I/O with implied DO loops	Advanced	Added**
Variable logical unit numbers	Advanced	Added**
Binary READ and WRITE unformatted	Advanced	Deferred
Scaling in FORMAT statement	Advanced	Deferred
F, E, I FORMAT field descriptors	Advanced	Added**
Evaluation of expressions—many variables, arithmetic, relational and logical	Advanced	Deferred
Assignment rules for expressions with a change in data type	Advanced	Added**
Variable dimensioned arrays in subprograms	Advanced	Deferred
External procedure names as arguments in intrinsic function references	Advanced	Deferred

** Added elements were included in Version 1.0. Remaining advanced elements are not included in Version 1.0, however, these elements will be tested in future versions of the FCVS.

Additional manpower resources were added to the FCVS project in late April 1976 as the number of routines to be written had increased from fifty (50) to seventy-five (75).

Description of statement tests

The statement tests in the FCVS were built carefully from the foundation of the basic assumptions. There was a systematic increase in the complexity of the language features tested as succeeding FORTRAN audit routines were developed. Language features other than those in the basic assumptions were not included in a test until they had been thoroughly tested themselves. This method provides for the cross checking of test failures and allows for the precise identification of problem areas due to nonconformance to the language specifications or other compiler errors and deficiencies.

The first several routines in the FCVS test the language elements in the basic assumptions. Their correct execution ensures that the failure of any test in the remainder of the routines is due to the improper implementation of the language feature being tested.

A description of the first few tests for the Arithmetic Assignment Statement is included to show how the tests build upon previous tests. An Arithmetic Assignment Statement is of the form:

variable name = arithmetic expression.

The simplest form for the arithmetic assignment statement is:

integer variable = integer constant.

The first audit routine which tests arithmetic assignment statements contains tests of the above form where the integer constant is unsigned, positively signed and negatively signed. The unsigned and positively signed constants increase in absolute value in succeeding tests to a maximum of 32767, and the negatively signed constants decrease in value to −32766.

The next form of the arithmetic assignment statement to be tested is:

integer variable = integer variable.

In order to test this form the statements from the previous tests setting an integer equal to a constant must be used. The source code lines for these tests are:

integer variable 1 = integer constant
integer variable 2 = integer variable 1,

where the integer constant assumes the values previously tested. This process is continued with tests of arithmetic assignment statements of the form

integer variable = integer variable + constant,
integer variable = integer variable − constant,
integer variable = integer variable + integer variable,
integer variable = integer variable − integer variable.

By developing tests in this manner, if a problem with a language element appears in a particular construct, the problem is easily identified in all other tests which employ the same type of construct.

Test support code

The tests in the FCVS contain support source code which checks the results of the language features tested and produces output indicating the results of each test. The support source lines also contain statements which are executed if a test must be deleted in order to compile a program. If the compiler cannot handle a particular language feature which is being tested, that code is deleted by placing a C in column 1 of the source lines for that test. During execution, the program falls immediately into the test deletion source lines.

Section 9.2 of the 1966 FORTRAN Standard states "A program part may not contain an executable statement that can never be executed."[2] Since the test deletion code is only executed when a test is deleted, this specification required several IF statements to be added to the support code. The IF statements refer to statement labels which begin lines of source code which are not executed if the language element tested performs correctly.

An example of the source lines for two tests of the arithmetic assignment statement is given in Figure 1 to show the test construction and the support code common to each test. Figure 2 contains the same tests but test number 227 has been deleted in this example. In the execution of these tests on a given system, the Executive System will replace the X02 in the WRITE statement with the implementor-defined logical unit number for the printer.

Audit routine output report

The output report for each audit routine indicates whether the individual tests in the routine passed, failed or were deleted. A summary of the results for each routine is printed at the end of the output report. Figure 3 is an example of the output report for the audit routine FM004. This report shows that two tests in this routine failed, and the computed and expected results are given for these two tests. The comment lines within the program or the program documentation would have to be consulted to determine what language elements did not conform to the language specifications and thus caused these tests to fail.

The executive system

The FCVS source library tape contains system independent source programs with implementor-defined aspects such as logical unit numbers yet to be resolved. The Executive System was developed to build compilable programs from the FCVS source library tape. The purpose of the Executive System is to handle the implementation problems which occur even with programs written in Standard FORTRAN.

The Elementary Executive Routine was written for execution on a minicomputer system and contains only those capabilities expected of a system with limited resources. Because of this, the Executive Routine is written in FORTRAN using only language elements and features included in the basic assumptions.

~~The Elementary Executive Routine permits the selection~~ of a program from the FCVS source library tape by program identifier and the building of a compilable program file. Resolution of implementor-defined logical unit numbers and update capabilities by source line are performed as the program file is built. The update capabilities include inserting a source line, replacing a source line, deleting a source line, and changing a source line to a comment line by placing a C in column 1.

Testing

During July and August 1976, the audit routines comprising version 1.0 of the FCVS were tested on four systems:

- UNIVAC 1108 Field Data compiler under EXEC-8
- Data General NOVA 800 under RDOS version 3.0

```
C
C      TEST 225 THROUGH 234 USE PARENTHESES TO GROUP ELEMENTS IN AN
C      ARITHMETIC EXPRESSION.
C
 2271  CONTINUE
       IVTNUM=227
C
C      ****  TEST  227  ****
C            INTEGER VARIABLE=(2+INTEGER VARIABLE)+4
C
       IF (ICZERO) 32270, 2270, 32270
 2270  CONTINUE
       IVONO1=3
       IVCOMP=(2+IVONO1)+4
       GO TO 42270
32270  IVDELE=IVDELE+1
       WRITE (X02, 80003) IVTNUM
       IF (ICZERO) 42270, 2281, 42270
42270  IF (IVCOMP-9) 22270, 12270, 22270
12270  IVPASS=IVPASS+1
       WRITE (X02, 80001) IVTNUM
       GO TO 2281
22270  IVFAIL=IVFAIL+1
       IVCORR=9
       WRITE (X02, 80004) IVTNUM, IVCOMP, IVCORR
 2281  CONTINUE
       IVTNUM=228
C
C      ****  TEST  228  ****
C            INTEGER VARIABLE=2+(INTEGER VARIABLE+4)
C
       IF (ICZERO) 32280, 2280, 32280
 2280  CONTINUE
       IVONO1=3
       IVCOMP=2+(IVONO1+4)
       GO TO 42280
32280  IVDELE=IVDELE+1
       WRITE (X02, 80003) IVTNUM
       IF (ICZERO) 42280, 2291, 42280
42280  IF (IVCOMP-9) 22280, 12280, 22280
12280  IVPASS=IVPASS+1
       WRITE (X02, 80001) IVTNUM
       GO TO 2291
22280  IVFAIL=IVFAIL+1
       IVCORR=9
       WRITE (X02, 80004) IVTNUM, IVCOMP, IVCORR
```

Figure 1—Example of source lines for test of arithmetic assignment statement

- Digital Equipment Corporation PDP 11/70 under RSX-11M
- General Electric FORTRAN IV compiler under the MARK III timesharing system.

Milestones

The following chart shows the actual milestone completion dates to develop the FCVS.

Matrix to Identify FORTRAN Language Elements	31 OCT 75
Programming Procedures Document	21 NOV 75
FCVS Test Plan	26 MAR 76
EXECUTIVE Routine Specifications	28 MAY 76
FCVS Test Specifications—Working Papers	11 JUN 76
EXECUTIVE Routine Completed	26 JUN 76
Version 1.0 Test Routines Completed	04 JUL 76

FCVS Detailed Test Specifications Manual Version 1.0	09 JUL 76
Testing Completed—FCVS LIBRARY Tape Version 1.0 Produced	13 AUG 76
FCVS User's Guide Manual Version 1.0 Completed	13 AUG 76

SCOPE OF THE FCVS

The purpose of the FORTRAN Compiler Validation System is the testing of a compiler's conformance to the FORTRAN language specifications. The tests in the FCVS are "positive" in that only statements permitted by the

```
C
C      TEST 225 THROUGH 234 USE PARENTHESES TO GROUP ELEMENTS IN AN
C      ARITHMETIC EXPRESSION.
C
 2271  CONTINUE
       IVTNUM=227
C
C      ****  TEST  227  ****
C            INTEGER VARIABLE=(2+INTEGER VARIABLE)+4
C
       IF (ICZERO) 32270, 2270, 32270
 2270  CONTINUE
C      IVON01=3
C      IVCOMP=(2+IVON01)+4
C      GO TO 42270
32270  IVDELE=IVDELE+1
       WRITE (X02, 80003) IVTNUM
       IF (ICZERO) 42270, 2281, 42270
42270  IF (IVCOMP-9) 22270, 12270, 22270
12270  IVPASS=IVPASS+1
       WRITE (X02, 80001) IVTNUM
       GO TO 2281
22270  IVFAIL=IVFAIL+1
       IVCORR=9
       WRITE (X02, 80004) IVTNUM, IVCOMP, IVCORR
 2281  CONTINUE
       IVTNUM=228
C
C      ****  TEST  228  ****
C            INTEGER VARIABLE=2+(INTEGER VARIABLE+4)
C
       IF (ICZERO) 32280, 2280, 32280
 2280  CONTINUE
       IVON01=3
       IVCOMP=2+(IVON01+4)
       GO TO 42280
32280  IVDELE=IVDELE+1
       WRITE (X02, 80003) IVTNUM
       IF (ICZERO) 42280, 2291, 42280
42280  IF (IVCOMP-9) 22280, 12280, 22280
12280  IVPASS=IVPASS+1
       WRITE (X02, 80001) IVTNUM
       GO TO 2291
22280  IVFAIL=IVFAIL+1
       IVCORR=9
       WRITE (X02, 80004) IVTNUM, IVCOMP, IVCORR
           . . .
```

Figure 2—Example of test deletion procedure

```
FORTRAN COMPILER VALIDATION SYSTEM
DEPARTMENT OF THE NAVY
ADPE SELECTION OFFICE
SOFTWARE DEVELOPMENT DIVISION

PRE-RELEASE FORTRAN 1966—LIMITED DIS-
TRI.

FOR OFFICIAL USE ONLY—COPYRIGHT 1975

TEST    PASS/FAIL    Computed    Correct
----------------------------------------
 21     PASS
 22     PASS
 23     FAIL            0           1
 24     PASS
 25     PASS
 26     PASS
 27     PASS
 28     PASS
 29     PASS
 30     FAIL           -2           2
 31     PASS
 32     PASS
----------------------------------------
        END OF PROGRAM FM004
         2 ERRORS ENCOUNTERED
        10 TESTS PASSED
         0 TESTS DELETED
```

Figure 3—Example of audit routine output report

Standard are included. There are no "negative" tests of incorrect statement formats which a compiler is suppose to flag as errors.

The FCVS also does not test vendor extensions to the language specifications, and does not perform an error analysis on the results of executing the Basic External Functions supplied by FORTRAN processors. The FCVS is not designated to measure the efficiency of the object code generated or the performance characteristics of a FORTRAN compiler.

FUTURE FCVS DEVELOPMENT

X3J3 has developed a draft proposed revised FORTRAN Standard consisting of a full language and a subset language to replace American Standard FORTRAN, X3.9-1966. X3J3 has also recommended withdrawal of X3.10-1966, Basic FORTRAN since a FORTRAN subset is defined in the revision to X3.9-1966. The proposed revision is in the process of being accepted by ANSI and it is anticipated in the "near" future there will be a new FORTRAN Standard.

A study of the new draft revised Standard and associated appendices reveals that programs conforming to the 1966 Standard will also conform to the revised Standard. The changes to FORTRAN from X3.9-1966 to the X3J3 revision were made "only when such changes were necessary to correct an error in the previous standard or to add to the power of the FORTRAN language in a significant manner. In addition, such changes were only considered when it was felt that the change would not affect a significant number of programs."[3]

The FCVS developed for the 1966 Standard will be the foundation for an FCVS for the complete revised Standard. Major additions to the current FCVS will be required to test the new language features in the revised Standard. The motivation and philosophies previously described for the current FCVS remain essentially intact in developing a compiler validation system for the complete revised language Standard.

The FORTRAN Data Base Committee of CODASYL is developing a data base facility to allow a FORTRAN user to manipulate data bases. The data base facility is based on both the CODASYL Data Base Facility and the revised FORTRAN Standard. A working document of the FORTRAN Data Base Committee, CODASYL FORTRAN Data Base Facility Journal of Development,[7] describes a set of data manipulation language statements and data definition language statements "intended to be in the spirit of FORTRAN."

If the FORTRAN data base facility is accepted by the FORTRAN Community then data base validation routines would be developed for inclusion in the FCVS. The growth in the use of data base concepts for large and small scale computer systems makes validation techniques for host language interfaces important.

CONCLUSIONS

The FORTRAN Compiler Validation System provides a tool for measuring a compiler's conformance to the FORTRAN language specifications. Properly administered, the FCVS will promote improvements and eliminate compiler deficiencies from vendor supplied software. The FCVS will be used by the ADPE Selection Office, Department of the Navy, in the procurement process. It is an important addition to procurement procedures and the FCVS will ensure the selection of computer systems with compilers that support the FORTRAN Standard.

The FCVS is now available to the user community. Any comments or suggestions on the FCVS will be appreciated and should be addressed to:

Department of the Navy
Software Development Division
ADPE Selection Office
Washington, D. C. 20376

REFERENCES

1. Sammet, J. E., *Programming Languages: History and Fundamentals*, Prentice-Hall, Incorporated, 1969.
2. American Standard FORTRAN, X3.9-1966, American National Standards Institute Incorporated, New York, 1966.
3. American National Standards Committee X3J3, Draft Proposed ANS FORTRAN, ACM Sigplan Notices, Vol. 11, No. 3, March 1976.

4. FCVS Detailed Test Specifications, available from the National Technical Information Service, US Department of Commerce, 5285 Port Royal Road, Springfield, Virginia, 22151, reference ADA030211.
5. Hopper, Captain Grace Murray, USNR, "U. S. Navy FORTRAN Tests," March 1971, unpublished Department of the Navy Documentation.
6. Holberton, F. E. and E. G. Parker, "NBS FORTRAN Test Programs," U. S. Department of Commerce, National Bureau of Standards, NBSIP 73-250, June 1973.
7. CODASYL FORTRAN Data Base Facility, CODASYL Journal of Development, Version 1.0, August 1, 1976, published by CODASYL FORTRAN Data Base Committee.

A two-step approach to the validation of software engineering methodologies*

by GRUIA-CATALIN ROMAN
Washington University
St. Louis, Missouri

ABSTRACT

A two-step approach to the experimental validation of software development methodologies is proposed. The first stage consists of the evaluation of the programming effort required for the reconstruction of some existing medium-sized program which (1) has known development and maintenance costs and (2) is anticipated to undergo numerous developments in the immediate future. The second step is shown to involve a comparative study of (1) the programs' performance, (2) the degree of expertise required to maintain the two programs, (3) the maintenance costs, and (4) the average adaptation times necessary to integrate a new programmer into the maintenance team.

The approach is demonstrated for a method called *program control restructuring* that was used in the reconstruction of a complex 10,000 line biochemical simulation system.

THE EXPERIMENTAL ENVIRONMENT

Program control restructuring is a method that was designed to be a useful tool for constructing large programs. It provides good quality design and documentation, expedites the implementation process, improves the readability and flexibility of the program, diminishes the frequency of logical errors, and limits the knowledge needed for making program alterations. Furthermore, program control restructuring was conceived as a method that performs well, especially in those adverse circumstances where the program is not an end in itself, but a research tool. In such conditions the program is under continuous development, only partially specified, and exposed to frequent changes in the design specifications. Moreover, programming is often done by transient personnel, hired on a temporary or part-time basis to accomplish specific developments, or by the researchers themselves, who may have some limited computer science training.

No theory is capable of establishing to what degree program control restructuring is able to control such extreme circumstances as those provided by the environment described above. Only practice can offer the final validation. This is the reason why the project SIMBIOR was conceived—the first practical experiment using program control restructuring. SIMBIOR is comprised of two distinct steps. The first one includes the rewriting (based on the program control restructuring approach) of a large and complex biochemical simulation system[4] which for convenience will be referred to as BSS. The second step involves the study of the maintainability, flexibility, and performance of the new program, called BIOSSIM, over a two-year period characterized by intensive developments and numerous changes in personnel.

That SIMBIOR is indeed a fair trial for program control restructuring may be justified by BIOSSIM's size and complexity but, more importantly, by the fact that the environment in which it was constructed and later used is the very same one described in the beginning of this section. The key to a convincing validation of program control restructuring is to be found in the hardships it had to overcome. From the very beginning of the project, it was clear that BIOSSIM had to be constructed with limited financial and no technical support. While satisfying strict efficiency requirements, it also had to meet demanding deadlines. Moreover, the environment was to have no negative effect upon the quality of the product.

Facing these demanding circumstances was a modest and inexperienced programming team. Its members had never before coded more than several hundred lines; neither had they ever used any systematic approach or well-defined methodology. The situation was made worse by the fact that all of them were employed part-time, a circumstance which could cause a team that required considerable interaction to become very ineffective.

SIMBIOR gained financial support based upon the preliminary evaluation of program control restructuring as a software engineering methodology. A tentative schedule was adopted. It was estimated that the construction of BIOSSIM would take eight months followed by two months of testing and would require a three-person team working fifteen to twenty hours per week. There was also a general agreement that the new version would be less efficient and need more core since it was intended to be machine

* This research was partially supported by NIH grants (GM-16501, HL-15622, and RR-15.

independent (BSS was not) with the accent on flexibility and clarity. However, the disadvantages were viewed as insignificant in comparison to the gains.

To validate program control restructuring, SIMBIOR had to achieve the following main goals:

(1) a. to adapt the method to the use of FORTRAN and to the specifics of the problem under implementation (to show both generality and adaptability);
    b. to meet the deadlines established for completion of BIOSSIM (to demonstrate high productivity and good utilization of available human resources);
(2) to prove the program's maintainability and flexibility during subsequent developments carried out by transient programmers.

The following sections will describe the test program, the methodology, the two validation steps, and the conclusions of the experiment.

THE TEST PROGRAM

A short description of BSS may help the reader gain a basic appreciation of its complexity and value. BSS was designed to be a biochemical research tool. It is used by a number of biologists and biochemists to build, validate, and experiment with various metabolic models. Its main capability is that of predicting the state (a state is defined by the concentration vector of all chemical species involved in the model) in which the system is to be found at various future points in time. (Time is viewed as being continuous.) BSS accepts a formalized description of a biochemical system, simulates its behavior, and displays its state at certain selected moments in time. Dynamic and state alterations to the system are permitted.

The original version of BSS was written by one person within a year's time and has undergone much development and expansion for more than ten years, reaching a length of approximately 10,000 cards of highly efficient code. The system, initially conceived on a PDP-6 and later run on a PDP-10 computer, was reorganized two times and transferred to an IBM 360. The first attempt required more than one year during which minor alterations were made and debugging took place. The most recent transfer entailed six months of intensive effort made by a team of two persons. The users became increasingly aware of the inadequacies of the system (high development and maintenance costs) as the demand for new facilities and portability intensified.

Because BIOSSIM is intended to be internationally distributed, it is written in FORTRAN. It is assumed that most installations support this language and most users are likely to have some FORTRAN experience. The language choice is also considered as appropriate from the experimental point of view. Since the methodology is language independent, the use of FORTRAN can emphasize the generality of the method even for those that tend to be very hostile toward languages such as FORTRAN. A good approach must be able to function satisfactorily in any environment even when it is known to work better under certain conditions than others.

THE METHODOLOGY

A complete description of program control restructuring and the motivation behind it may be found in Reference 8. For the purpose of this exposition, it suffices to present at this point only a simplified view of program control restructuring. A control restructured program may be viewed as a set of subprograms organized top-down in a tree-like hierarchy. Each subprogram in turn is represented by a set of procedures or routines organized on two levels. The first level is represented by a control block while the second one contains several modules.

The sole function of the control block is that of calling the various modules that belong to the same subprogram and the control blocks of lower level subprograms. The control block is not permitted to alter any data. However, it is allowed to test the value of a special integer variable called the control variable. The modules, on the other hand, are restricted from calling any other procedures. They perform specific transformations over the data to which they have access. Among the modules of each subprogram, there is a distinguished one called the initialization-documentation module. This module is the first ever to be called by the control block and plays the role of setting all pertinent variables to their initial values. Such a program structure is presented in Figure 1.

While other aspects of program control restructuring will be introduced in the next section as they are needed, it must be pointed out now that the key idea behind program

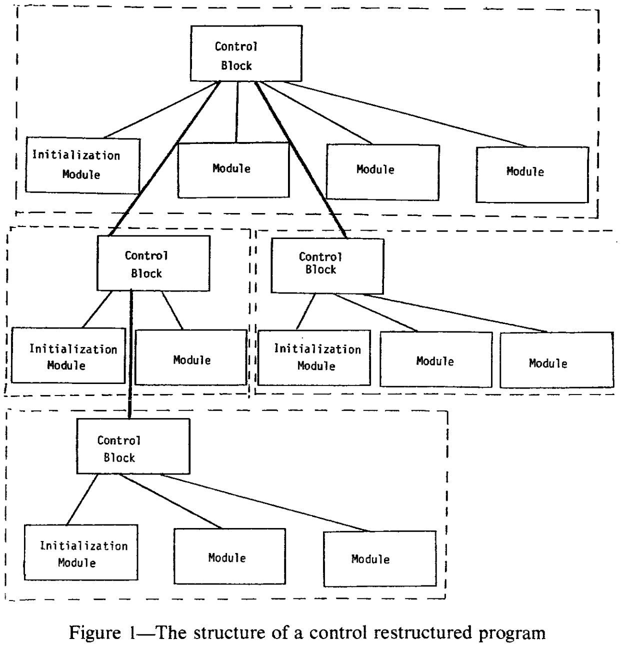


Figure 1—The structure of a control restructured program

control restructuring is that of relying heavily on standardization (based upon language and problem dependent criteria) in order to achieve very high productivity, good quality documentation, and inexpensive development and maintenance.

THE RECONSTRUCTION

This section is dedicated to describing the construction of BIOSSIM, the control restructured version of BSS. As previously stated, this was the first stage of the project SIMBIOR and was intended to demonstrate the impact that program control restructuring has upon the productivity and quality of software production in the context of a most imperfect and demanding environment. At the same time, this section provides a concrete example of the way in which program control restructuring may be adapted to a specific problem and given circumstances. The reader interested in employing this approach will find here very useful guidelines.

Prior to starting the actual design of BIOSSIM, a preliminary study of the problem and environment was conducted in an effort to organize the project and select the appropriate standards for program structure. A short and well-structured weekly meeting was viewed as being necessary for progress reports and handing out programming assignments. However, the only precise and correct communication channel was the documentation itself. Each programmer was expected to work independently and require little or no interaction with anyone else. Regulations regarding the usage of auxiliary storage, updating of listings, and precautionary procedures were also established.

The first step in selecting the project standards was data structuring. The process resulted in the adoption of the following types of variables:

- Control variables—as previously presented, they are used by the control blocks to determine the sequencing of modules and may be altered by the modules only.
- General variables—they are global variables constant for significant portions of the program and grouped in labeled commons based upon their functions.
- Local variables—they are represented by those variables local to a given module with the exception of the do-loop indices.
- Do-loop indices.
- Debugging variables—they are globally defined flags stored in a labeled common and used to control the activity of the debugging modules.
- Workspace variables—they are globally defined arrays (in labeled commons) upon which the transformations are applied. The structure of the workspace variables is standardized.

Standards were also imposed upon the modules. They were chosen in an effort to assure not only low connectivity but also reduced maintenance and development costs, small probability of error, and simplified verifications. The most important standardization rules refer to restrictions regarding access to certain classes of variables by certain types of modules, the preservation of the invariant properties of the workspace variables, and the definition of a maximum complexity for each type of module.

Due to certain specific needs of the program, a number of deviations from program control restructuring were accepted as absolute necessities. The introduction of a group of modules called heterogeneous routines is one concession made to assure good portability of the program. They are not actual modules and therefore may be called from within any module. The heterogeneous routines include specific short machine-dependent sections which could be part of the modules, but the designer felt the need to single them out. Transferring the program from one system to another should require only the rewriting of the heterogeneous routines. Later experiments proved this to be true—the transfer cost was reduced by 80 to 90 percent.

The use of FORTRAN determined not only superficial changes in terminology (e.g., control block became control routine) but also the solutions adopted for a number of aspects defining the implementation standards. A first set of language dependent decisions refer to naming and labeling conventions. One-letter prefixing was selected as the simplest way to provide the programmer with immediate access to the semantics of the named element. Thus, each class of modules or variables has a unique letter by which it is identified in an unambiguous manner (e.g., X2OUT must be a control block on the second level). Furthermore, groups of labels have been associated with specific usages or constructs.

The second type of language restrictions involves the usage of constructs within the modules. These conventions have been designed primarily to assure the generation of intelligible code and to simplify program verification. In order to produce block-structured routines, backward and intersecting GOTO paths have been eliminated. As a result, the computational flow is naturally divided into logical blocks.

The need to subdivide BIOSSIM into a number of subprograms was justified by the complexity of the program, the incomplete nature of the program description, and the instability of the specifications. Being a research and development project, the construction of BIOSSIM required that many key decisions be taken or reversed at times when the program implementation was in an advanced state of completion. The division of the system into subprograms diminished the impact of such circumstances and allowed the designer to perform all necessary adjustments at a low cost and in a very short time. The criteria applied in separating the program into subprograms allowed for the design and implementation of the various subprograms taken as a group to be completed in an arbitrary order, which was by no means top-down or bottom-up. However, each subprogram in and of itself was independently designed and implemented in a strictly top-down fashion.

The only documentation preceding SIMBIOR was the user's guide, which provided the initial specifications for BIOSSIM. The first element of documentation produced during the course of the project was the introduction to the programmer's guide. It included a detailed and complete description of the methodology and program structure adopted by SIMBIOR. Subsequently, the programmer's guide accumulated information regarding the algorithms used, references, debugging facilities, etc.

While maintaining the traditional external documentation sources, program control restructuring places a special accent on developing a powerful internal documentation scheme. Each routine is both an active program segment and a documentation source of a specific character determined by the role it plays in the structure. From the documentation point of view, each control block is a "conversational dictionary." The conversational dictionaries indicate the information which may be accessed by each module and the way it can use the variables made available. The comments preceding each call also include information regarding the purpose of the call, in what circumstances it is made, and the effect it may have upon the control variable.

Similarly, each documentation module becomes a "data set dictionary." Among them, the documentation module of the top subprogram is distinguished as the "main data set dictionary" while the others are referred to as the "regional data set dictionaries." The main data set dictionary contains all the global variables used by the program accompanied by a complete description of such things as their purpose, format, and initial values. In the cases where a variable is not documented at the top level or is used in a particular way by a group of modules, references to the places where the pertinent information exists are included. If an initialization takes place outside the documentation routine, an explanation is provided. Those variables that are not documented in the main data set dictionary appear in one of the regional data set dictionaries.

Regarding module documentation, each module contains, immediately after the necessary declarations, a list of all local variables, their rationale, and their initial values. This enumeration is vital for quick comprehension of the code and rapid checking of the type of variables used and is referred to as "the pocket dictionary."

The duality between program structure and documentation scheme promotes the understanding, in a clear and precise way, of the logic of the program and the use of variables. No assumptions have to be made by the reader at any point in time. All information is there; to reach it requires an effort comparable to finding a word in a dictionary.

The quality of this fully standardized documentation was validated by its use in the design and implementation of the subprograms. By making the program self-explanatory to a large degree, the documentation scheme used by program control restructuring eliminated the need for numerous incomprehensible documents which rarely form a unified documentation system anyway. As a result, it was no surprise that developing the documentation parallel to and as part of the program in a natural and totally interdependent manner registered an unusual success.

The construction of each subprogram always followed the same pattern: first, the control block was designed and its correctness informally established; the implementation of the control block followed; the design and coding of the modules came next; and, finally, the testing and debugging of the subprogram took place. The verification of each subprogram, prior to implementing the control block, was found to be extremely useful both in eliminating design errors and also in preparing correct design specifications for the modules. Although the proofs were carried out in an informal manner, their effectiveness exceeded the expectations. All logical errors were completely eliminated during the verification stage. Furthermore, subprogram verification forced the designer to define the interfaces in a very precise and careful manner—one more factor that contributed to reducing the number of coupling errors to zero.

VALIDATION OF THE RECONSTRUCTION STEP

The distribution of the work responsibilities among the members of a programming team plays an important role in achieving full utilization of the human resources. A concept that proved itself very helpful in reaching this goal was the *ideal team* notion. The ideal team is a team in which, for each indivisible project responsibility, one person is assigned. The structure of the ideal team may easily be determined by analyzing the processes involved in implementing a specific method. In most circumstances one does not expect to have the resources necessary to form an ideal team. However, by knowing its structure the allocation of responsibilities among the available human resources may be done properly so as to assure the highest possible productivity. Each member of a real team may assume the role of one or more members of the ideal team depending upon his experience and qualifications. On the other hand, the duties of each member of an ideal team should never be assigned to more than one person. If this simple rule is not obeyed, confusion, increased need for communication, and lower productivity will result. An example of an ideal team is the chief programmer team concept,[2] which was devised and tested in its pure form by Mills and Baker. The differences between their approach and program control restructuring are reflected in the structure of the respective ideal teams. The ideal team for program control restructuring requires a chief designer, a documentation secretary, several programmers, and technical support personnel.

The CHIEF DESIGNER needs to have an excellent command of the problem to be solved and a strong structuralist orientation. His work duties should consist of the following:

- the selection of the design and implementation standards to be adopted;

- the general design of the system and its correctness proof;
- the writing of control blocks;
- the specification of work assignments for the team;
- synchronization of the work;
- the ultimate decision as to choice of programming solution;
- the supervision of any changes to the documentation;
- design of test cases;
- the testing of the whole system.

The DOCUMENTATION SECRETARY has to take care of the maintenance and updating of

- the module library;
- the documentation modules;
- the external documentation files;
- the manuals and archives;
- the backups;
- the distribution of information regarding any changes in design or documentation to all interested parties.

No request is to be serviced by the secretary unless approved by the chief designer.

The PROGRAMMERS have the duty to code, test, and debug the different modules or subprograms whose specifications are given to them by the chief designer. It is important that they respect all conventions and restrictions upon which the system is based. In order to generate unity of style, they need to be able to adopt easily the programming style of the group. A good understanding of the methodology helps in taking advantage of the available documentation. The programmer is not allowed to use any information or data not indicated in the specifications.

A TECHNICAL SUPPORT TEAM with experience in the kind of work in which the project is involved becomes an important factor in accelerating the work and in decreasing the number of typing and minor syntactic errors.

Material conditions did not permit the employment of the three-person team estimated as being necessary to meet the deadlines set by the schedule. The project had to adapt to the situation by reassigning the work duties of the third person to the two available persons. One of the team members assumed the responsibilities of chief designer and documentation secretary and a small part of the programmers' work load. The remaining programming needs and most of the required technical support were responsibilities of the other person. The total amount of time available was only thirty-five to forty-five hours per week. However, the project was developed on a very reliable time-sharing computer, the PDP-10. A noninteractive machine would require that the work be somewhat differently organized due to the possibility of slow turnaround.

Because of the limited human resources, the modules were not proven correct before coupling them together in a subprogram. Therefore, the need for a well-defined systematic testing procedure appeared to be critical for eliminating any errors existing in the modules. The subprogram testing (theoretically unnecessary if the modules were guaranteed to be correct) was structured as a combined subprogram and module evaluation. Although the modules, when compiled for the first time, were usually full of typing errors, the testing proved to be less expensive than one would expect. An appropriate selection of the modules, the use of invariant properties, the very effective and precise documentation, and the separation of the very simple control from the transformations are some of the reasons that reduced testing costs. Another cost decreasing factor was the fact that the design of each subprogram was concerned with minimizing the number of interfaces. Consequently, the drivers and stubs needed to simulate the subprogram's connections to the rest of the program were few and easy to implement. The testing procedure followed the pattern of the subprogram verification. It was aimed at showing that each transformational routine preserved the invariant properties of the variables involved and efficiently achieved its purpose. The analysis moved sequentially from one routine to the next and attempted to be exhaustive. Special attention was given to those conditions that determined the termination of the subprogram.

The fact that a large portion of the time was spent in designing and coding rather than debugging has to be attributed exclusively to simplicity, clarity of design, and the use of correctness proofs. All the debugging was strictly at the level of a programmer's duties. No errors occurred in coupling the subprograms together. All errors were local and, as it was, none of them required much effort to be corrected.

The means by which the subprogram testing and debugging were performed were the DDT facility available on the PDP-10 computer and BIOSSIM's built-in debugging facilities. DDT provides the programmer with breakpoint, display, and change commands. At the same time, BIOSSIM is able to input a set of debugging flags designed to control the activity of the debugging modules. These modules, built for specific purposes, are stored in a file separated from the program itself. They are called from the control blocks and are permitted to trace or display any variable but prohibited from altering any data belonging to the program. The removal of a debugging module cannot have any side effects. However, the debugging modules may be kept in the program (if properly documented) since their activation occurs only upon the programmer's request.

The control routines and the control variables made debugging simple and inexpensive. Tracing the control variable is the best way to analyze the behavior of the program. By knowing the successive values of the control variable, one can easily determine from looking at the control routine what calls have been made and in what order. Furthermore, the low connectivity among BIOSSIM's modules made the detection of error origin very rapid and eliminated the risk of unpredictable side effects when the correction was performed.

At the beginning of the project, there was little doubt that a better program would be produced. There were, however, questions about the time needed to do it and a general consensus that the program would suffer a considerable loss of efficiency. Nevertheless, BIOSSIM ran faster and required less core. Furthermore, a construction time of four months by two programmers employed half-time exceeded the expectations of the best experts familiar with the program. The reader is reminded that one person worked one year on the initial design of BSS; since then programs have been added to it during a period of more than ten years. It is reasonable to consider that the intensity of work was similar in both cases. Productivity is the factor that generated the significant differences in time and effort. With a modest team like the one used to build BIOSSIM, one may justifiably submit that the remarkable success (at this point only a reduction of time, effort, and cost) is entirely due to the methodology developed.

MAINTENANCE VALIDATION STEP

Upon the completion of BIOSSIM, the project SIMBIOR entered its second validation stage: development and maintainability tests. The results reported in this section cover the first two years since the completion of BIOSSIM.

The programmers employed to alter BIOSSIM were hired sequentially on a temporary basis to provide assistance with the various developments requested by the biological research team. Only one of them was part of BIOSSIM's programming team; the others had no previous experience with program control restructuring or BIOSSIM. They all were exposed to a two-day training session in which the principles of program control restructuring, the standards, and the documentation scheme were explained. Starting with the third day and from then on, they were given programming assignments of significant difficulty, which they handled with surprising success. Information which could be obtained from the documentation was intentionally withheld from them as a way of forcing them to make use of the documentation scheme. The brief adaptation period was especially surprising since it was shorter than the most optimistic expectations. Experience with BSS showed that the accommodation period varied from one to two months and only after four months could one assign the programmer to make alterations of some major significance. Contrast this with the fact that some programmers who worked productively for BIOSSIM were able to stay with the project only three months! Furthermore, while working on BIOSSIM, it was very rare for programmers to make serious errors, and in no case did their alterations produce any unexpected side effects.

Their work has been considered to be very satisfactory and, indirectly, they evaluated program control restructuring as being equal to the claims it makes. Furthermore, the transient programmers were faced with major developments in the short time they were associated with SIMBIOR. Among the new facilities offered to the user were better online and post-simulation scope plotting and an extension to the simulation language allowing one also to include FORTRAN-like statements in which chemical names, fluxes, etc., are used instead of variables. This capability extended the power of the language to a degree never before achieved. Consequently, it allowed the research team to use the language to implement a heart model which previously was produced by tedious hand coding. The price for implementing this so-called SIMFOR capability was only two weeks of work. Another major achievement was a machine-independent compaction of the partial derivatives matrix for the case where the initial sparseness is known in advance. This reduced the differential equation solving time by as much as seventy-five percent. A last surprise was the construction in two weeks, instead of two months, of a very much needed program which finds consistent values for heavily underdetermined environmental inputs. The complexity of these developments fully probed the flexibility of design introduced by program control restructuring.

The experiment was a success that fully justifies the proposed methodology. BIOSSIM was built as a control restructured, machine-independent program for large distribution. It proved to accommodate rapid and unpredictable developments. The methodology was shown to assure low-cost maintenance and expansion as long as any changes made to the program respect the conventions established when BIOSSIM was initially constructed. Its style and structure cooperate in preserving its entity.

CONCLUSION

The proposed experimental validation procedure reflects the recent changes of attitude among those involved in the development of software—the realization that there are two distinct productivity requirements that must be satisfied by any new methodology. The first one is associated with the construction phase of the system, while the second relates to the maintenance and future development costs. Subsequently, however, the validation itself must satisfy an important acceptance criterion—the experimental environment must be a realistic one. Any experimental validation is relative to the environment in which it was carried out.

REFERENCES

1. Aron, J. D., *The Program Development Process: The Individual Programmer,* Addison-Wesley Publishing Co., 1974.
2. Baker, T., and H. Mills, "Chief Programmer Teams," *Datamation,* 19, 12, 1973, pp. 58-61.
3. Dahl, O.-H., E. W. Dijkstra, and C. A. R. Hoare, *Structured Programming,* Academic Press, New York, 1972.
4. Garfinkel, D., "A Machine Independent Language for the Simulation of Complex Chemical and Biochemical Systems," *Computers and Biomedical Research,* 2, 1, 1968, pp. 31-44.
5. Kernighan, B. W. and P. J. Plauger, "Programming Style—Examples and Counterexamples," *ACM Computer Surveys,* 6, 4, 1974, pp. 303-319.
6. Mills, H. D., "Syntax-Directed Documentation for PL360," *CACM,* 13, 4, 1974, pp. 216-222.
7. Parnas, D. L., "On the Criteria to be Used in Decomposing Systems into Modules," *CACM,* 15, 12, 1972, pp. 1053-1058.
8. Roman, G.-C., *Program Control Restructuring—A Software Engineering Methodology,* Ph.D. Dissertation, University of Pennsylvania, 1976.
9. Yourdon, E., *Techniques of Program Structure and Design,* Prentice-Hall Inc., 1975.

Surveying the billion dollar chasm—How educational differences continue to force corporate and data processing executives apart

by ROBERT S. HOBERMAN
The INSCO Systems Corporation
Neptune, New Jersey

ABSTRACT

Many problems occurring in the use of computers are, in fact, not machine problems, but people problems centered around failures to communicate objectives and to understand and carry out responsibilities. These problems have caused computers to become as much a frustration to management as a useful tool.

Corporate executives, expert in their areas, but naive to or disinterested in specific data processing efforts, have trouble defining what they want. They often defer key judgments on the design of information systems to data processing professionals who usually lack business training, experience and perspective. Results include the d.p. professional's use of comfortable techniques that may alter what the user actually needed and, collectively, produce gross inefficiencies in a company's data processing effort.

Much of this is fostered by limited education on both sides. Corporate and data processing groups appear to be growing apart rather than coming to some point of conciliation.

For the 25 years or so the computer industry has existed, there has been great concern about surrendering certain management controls to computers, coping with changes in staff and organization structures due to automation, and controlling the spiralling costs of information. Companies, pushed hard by manufacturers and software vendors, regard scientific management and automated management information systems as now essential to the success of their businesses. Yet the implementation of most of these systems, regardless of the size or wealth of the companies for which they are being built, will fail before it begins.

Few businesses in the 1970's, except for some small, owner-run firms, exist outside the direct influence of computers. Most medium and large-sized companies have their own computers. Many supplement their data processing capabilities by using the facilities of service bureaus. Yet, because of ineffective organizational and communications structures which often result in competition and misunderstandings between corporate and data processing staffs, there are numerous cases where brilliant data processing people link up with brilliant businessmen to produce streams of meaningless detail. It is accepted in many companies that computer systems are not expected to be finished by their original deadlines or within their original cost estimates.

The perspective of what constitutes a deadline is different. To the corporate executive it is the date on which he expects the report. To the data processing executive it is "x" days after he has received the final input needed for the system. And, even where deadlines are not at issue, many computer systems produce long and complex reports which some corporate executives study to cull out what meaningful information they can. More often they are confused by detail and the intended purpose of the reports, many of which go largely unused.

Shipments of computers during 1975 were estimated by the U.S. Department of Commerce to run about $11 billion.[1] Network Information Services, including leasing operations, used computer sales, timesharing, software support and education and security services, were expected to exceed $7 billion.[2] At the same time, the number of installed computers was expected to rise to about 93,000 and the number of computer terminals installed in industry was expected to jump to 1.25 million.[3]

Studies by Fritz Machlup of Princeton University in 1958, and by FORTUNE Magazine in 1963, showed that the percentage of the U.S. Gross National Product spent on knowledge had risen from 30% to 43% in that five year period.[4] There appear to have been no formal studies since that time, but it is obvious that the investment in information and information technology is staggering.

Despite the magnitude of the investment and a failure rate of as much as 40% attributed to computer installations in business and industry, there appears to have been little management thought directed toward optimizing investments in knowledge. Narrowness of views of top level executives and data processing professionals alike, and lack of understanding of the sweeping impact computers have on business suggest there are significant flaws in the education and communications ability of both groups.

This article is drawn from a study whose purpose was to determine how the education of corporate and data processing executives contributes to the waste of time and money and the failure associated with computers to produce useful information in line with immediate goals or corporate objectives.

Many problems occurring in the use of computers are, in fact, not machine problems, but people problems centered around failures to communicate, understand and carry out responsibilities.

It may be said with reasonable certainty that:

1. Corporate and data processing goals are often separately identified and not in alignment.
2. Top managers often lack either the interest or the perspective to positively influence the use of computers in their companies.
3. The limited education provided to corporate executives too often is oriented towards teaching them how to program a computer rather than how to apply its use to their business.
4. Data processing people attach loyalties to a closed circle within their own "industry" rather than to the company or industry in which their employers are engaged.
5. Mutual understanding between data processing and corporate executives is limited by the vague and suspicious image each has of the other.

In the study, which included an extensive literature review as well as original research, 494 corporate and data processing executives were contacted. They represented 462 companies, nearly all listed in the FORTUNE Double 500 Directory. Of the sample, 267 were corporate executives and 227 were data processing executives. A total of 179 (36.2%) persons returned questionnaires or sent letters. The response by industry and sizes of companies represented by the respondents are provided in Tables I, II, IIA, and IIB. The questionnaires sent to corporate and data processing executives are detailed in Appendix A.

The average company responding to the study, according to data supplied by the participants, probably has a data processing staff of 270 persons, average equipment rental of over $200,000 per month and an annual data processing budget of over $6 million. These figures are conservative and do not include participants from very large companies who did not supply data relating to the sizes of their installations.

Preliminary figures from the Bureau of Labor Statistics of the U.S. Department of Labor, sets the non-Agricultural work force of the United States at 78,817,000 in 1976[5] and projections carry it to 107.7 million in 1985.[6] Of this work force, the 1976 FORTUNE 500 Industrial Companies employ 14,412,992; 6,481,693 more are employed by the group consisting of the fifty largest commercial banks, life insurance companies, financial companies, retail companies, transportation companies and utilities. Additionally, figures for the FORTUNE Second 500 Industrials show they employ 1,860,002.[7] Thus, the companies from which the sample for this study was drawn represent about 28.9% of the estimated non-Agricultural work force of the United States.

Accordingly, using data from a 1975 study by Hitchcock Publishing Company, the data processing work force of the United States would be about 1.3 million of whom 385,004 then probably are employed by the company groups represented in this study. Table III shows the author's calculations based on data from the Hitchcock study and the FORTUNE Magazine figures.

The author's estimate of 1.3 million data processing employees compares reasonably to a 1973 study by the American Federation of Information Processing Societies which showed an approximate data processing work force of 1,000,000.[8] These calculations do not take into effect the recession and the effects of unemployment since there appear to be no figures available on the size of layoffs in the data processing field. They are assumed here to be of a similar or lesser proportion than layoffs of the general work force.

Industrial companies in the FORTUNE 1,000 would comprise 63.4% of the data processing work force employed by the company groups represented in this study. The study group was weighted heavily to industrial companies (75.3%) and, of the questionnaire returns, 66.5% came from industrial companies. Returns from the six other study group industries ranged from 2.8 to 6.7% of the total; since

TABLE I—Questionnaire Response by Industry

Company Type	Corporate Executives			D.P. Executives			Total		
	Mailed	Received	%	Mailed	Received	%	Mailed	Received	%
Industrial	199	64	32.1	173	55	31.8	372	119	32.0
Banking	10	4	40.0	9	5	55.6	19	9	47.4
Insurance	10	6	60.0	12	6	50.0	22	12	54.5
Financial	10	2	20.0	6	3	50.0	16	5	31.3
Retailing	10	4	40.0	8	3	37.5	18	7	38.9
Transportation	10	8	80.0	8	4	50.0	18	12	66.7
Utility	10	6	60.0	11	5	45.5	21	11	52.4
Consultant	8	4	50.0	—	—	—	8	4	50.0
Total	267	98	36.7	227	81	35.7	494	179	36.2

TABLE II—Response by Size of Data Processing Staff

Company Type	<100	100- 300	300- 500	500- 1000	>1000
Industrial	51.4%	24.8%	10.5%	03.8%	04.8%
Banking	00.0	44.4	33.3	22.2	00.0
Insurance	25.0	25.0	16.7	25.0	00.0
Financial	50.0	25.0	00.0	00.0	25.0
Retailing	50.0	50.0	00.0	00.0	00.0
Transportation	58.3	16.7	00.0	25.0	00.0
Utility	00.0	36.4	36.4	09.1	09.1
Consultant	25.0	25.0	00.0	25.0	25.0
Composite	42.9%	27.0%	12.3%	08.6%	04.9%

they employ from 2.7 to 10.9% of the data processing work force, the sample appears reasonably representative of the industry. Table IV shows the details.

ANALYSIS OF RESULTS

Because the industrial companies are so significant to both the data processing field and this study, it seems particularly relevant to show a comparison between one form of results obtained from this study and similar results obtained in a Booz Allen Hamilton study in the late 1960's which showed the median expense for computer equipment by 108 manufacturing companies to be $5,600 per $1,000,000 of sales.[10]

A sample of 29 companies, ranging in size from $100 million to $3 billion, was drawn randomly from a list of industrials that responded to the study. These companies had 1974 annual sales totalling better than $30 billion and had approximate 1974 annual data processing expenditures of about $176 million. Their median expense, then, for data processing had inched up in the last ten years to about $5,900 per $1,000,000 of sales.

This appears to track with Booz Allen's finding that computer investments were rising steadily from 4% of total investment in plant and equipment in 1961 to 10% in 1968. Additionally, John Diebold suggested in 1970 that the percentage of computer investments might reach 12% by the mid-70's.[11] And this for an industry that employs less than 2% of the total United States work force!

Despite the enormity of the expense, the study shows there is confusion over how much impact computers actually have on corporate executives' decisions. While corporate executives say computer reports have a limited use in daily decision-making, data processing executives perceive the use as very high. And, while both corporate and data processing executives agree that their companies have improved significantly since the advent of computers, sizable groups on both sides find the efforts inconsistent.

Forty-five percent of the data processing executives say corporate executives have minimal-to-poor knowledge of data processing; ten percent more say they find corporate executives disinterested in data processing; another 35% say that corporate executives either delegate the control of systems development projects or remain out of the picture entirely.

Thirty percent of the corporate executives agree with this despite the warnings of consultants like Withington, Diebold, Brandon, Wight and others that computers represent major investments that ought to be closely managed by top management.

W. Blake Thompson, senior vice president of financial planning at Allegheny Airlines offered this comment in the study, "A close control must be maintained over the data processing organization and a complete cost vs. benefit analysis must be made of major applications. In general, an aggressive manufacturer sells unneeded equipment to willing buyers (data processing managers) unless the top management of a company is in a position to understand what they are being told. In addition, inefficient operations are paved over with excessive manpower and equipment."

Joseph Orlicky, IBM's leading consultant to the manufacturing industry, says that "top business executives are willing to spend money but not their time to improve the systems efforts of their companies."[12]

The question is, "Why?"

In some cases, corporate executives probably still are awed by the technology and jargon of data processing; in others, top executives simply find computer systems too complex and frustrating. They prefer to stick with areas they know well rather than show ignorance or weakness by getting into discussions of computer applications.

Yet one-third of the corporate executives who responded to this study said they are not satisfied with the quantity of

TABLE IIA—Response by Monthly Equipment Rental (In $000's)

Company type	50	50- 100	100- 200	200- 300	300- 400	400- 500	>500
Industrial	38.1%	22.9%	13.3%	06.7%	03.8%	01.9%	09.5%
Banking	00.0	00.0	22.2	22.2	11.1	11.1	33.3
Insurance	08.3	08.3	08.3	16.7	00.0	16.7	16.7
Financial	25.0	25.0	00.0	00.0	00.0	00.0	25.0
Retailing	33.3	50.0	00.0	00.0	00.0	00.0	00.0
Transportation	25.0	16.7	16.7	08.3	08.3	00.0	25.0
Utility	00.0	00.0	36.4	18.2	27.3	00.0	09.1
Consultant	00.0	25.0	00.0	00.0	00.0	25.0	25.0
Composite	28.8%	19.6%	14.1%	08.6%	05.5%	03.7%	12.9%

TABLE IIB—Response by Annual Budget (In $ millions)

Company type	<2	2-4	4-6	6-8	8-10	10-12	12-16	>16
Industrial	40.0%	18.1%	12.4%	05.7%	03.8%	03.8%	02.0%	09.5%
Banking	00.0	00.0	22.2	33.3	00.0	00.0	11.1	33.3
Insurance	08.3	25.0	08.3	00.0	00.0	16.7	08.3	16.7
Financial	50.0	25.0	00.0	00.0	00.0	00.0	00.0	25.0
Retailing	33.3	33.3	16.7	00.0	00.0	00.0	00.0	00.0
Transportation	41.7	25.0	00.0	08.3	00.0	00.0	08.3	16.7
Utility	00.0	18.2	18.2	00.0	27.3	00.0	09.1	18.2
Consultant	00.0	00.0	00.0	25.0	25.0	00.0	00.0	25.0
Composite	31.9%	18.4%	11.7%	06.7%	04.9%	03.7%	03.7%	12.9%

information they get and 25% more say they are not satisfied with the quality. On top of this, only slightly more than half say the costs of developing new computer systems are consistently within budget.

If their attitude is so confused and they are so unwilling to get involved with an activity that may represent 12% of their companies' investments, one must question whether they would be similarly uninvolved in the building of a new plant or in the marketing of a new product whose overall annual cost may be far less.

No small part of the problem rests with data processing executives. According to this study, more than 80% of them think corporate executives are pleased with quality and quantity and nearly 95% of them apparently feel that corporate executives are well satisfied with the data processing organization, generally. They also seem to think that corporate executives think costs are consistently under control.

The president of a large FORTUNE 500 company summarized his feelings about questions of quantity in computer reports by writing across the questionnaire, "Too much junk!"

The manager of Data Processing Standards, Controls and Training at a major oil company says, "Most users are reasonably satisfied—but only because they don't appreciate how much better they could do."

The Director of Information Services for another large industrial firm says, "Data processing is O.K. However, many systems demand or provide excessive information to the manual action areas."

While data processing executives say they would like corporate executives to play a much larger role in systems development, they also suggest that corporate executives don't know enough about data processing to contribute very much.

The data processing manager of a manufacturing company commented in this survey that, "Management is unknowledgeable and not interested enough to become more knowledgeable."

Another data processing executive, from a much larger manufacturer, comments that "Conventional functions still view the data processing operation defensively despite the fact that most major companies' systems are 50-75% computerized. This could be termed business 'culture lag'."

Perhaps most directly pointing to the feelings that corporate executives show toward data processing is this quote from the senior vice president—finance—of a major pharmaceuticals manufacturer:

"A computer is nothing but a glorified adding machine—a good manager gets things done through people and uses

TABLE III—Author's Calculations of Probable Relative Sizes of Data Processing Staffs (by Industry)

	Company type	Total #[7] Employees	% D.P. to[9] Total Staff	Probable D.P. Staff	% Profile of Probable D.P. Staff
Top 500	Industrial	14,412,992	@ .015	216,195	56.2
2nd 500		1,860,002	@ .015	27,900	7.2
	Banking	458,597	@ .027	12,382	3.2
	Insurance	421,961	@ .099	41,774	10.9
	Financial	381,779	@ .027	10,308	2.7
	Retailing	2,652,959	@ .011	29,183	7.6
	Transportation	980,401	@ .011	10,784	2.8
	Utilities	1,585,996	@ .023	36,478	9.5
	Total	22,754,687 Therefore,	@ .169	385,004	100.1
	(Total non-Agricultural U.S. Work Force)	78,817,000	@ .169	1,332,007	(Total U.S. D.P. Work Force)

TABLE IV—Relationship Between Questionnaire Response and Estimated Percentage of Data Processing Work Force Employed

Company type	Estimated D.P. Staff	% of Total D.P. Staff	% Returns of Questionnaires
Industrial	258,299	64.4	65.3
Banking	12,369	3.1	5.0
Insurance	41,935	10.4	6.7
Financial	10,893	2.7	2.7
Retailing	29,772	7.4	3.9
Transportation	10,111	2.5	6.7
Utilities	38,373	9.5	6.1
Total	401,752		

all he can. Computer people can get, gather and shape information but a manager must *himself* know what he needs to run a company or division. There has been too much glory for the programmers and operators when they really know little else but the machine."

For all the criticism and dialogue, there is little communication. Corporate executives recognize in large proportions (74.4%) that data processing efforts in their companies would improve if they would learn more about data processing. They think data processing planning could be better, but as top executives of their firms—people for whom planning is a daily function—they remain largely aloof from data processing projects despite their expressed good intentions.

Repeatedly, this study shows good intentions coupled with bad performance. Executives recognize the value of a course in Computer Concepts—80% stress that it is worthwhile—yet 40% have never taken such a course. IBM Corporation has a course which is generally close to what corporate executives say they want, and the American Management Association makes some attempts to keep senior-level executives abreast of changes in technology and their impact on corporate life. There are few other meaningful courses available from manufacturers, consultants and universities, and the problem of getting senior-level executives to class, additionally, is difficult. But even when these executives have the chance to learn from their own staffs—and to ask their own questions—they don't.

The majority of corporate executives who responded to this study say they think joint meetings would help communications, yet not one industry group indicates that even 50% of the companies it represents hold joint meetings at all. A large number of corporate executives think having the data processing managers attend corporate officers' meetings would be a fine idea, but they say practically no one does it. Of the insurance executives responding, not one, according to his own statements, has ever taken a Computer Concepts course!

There is general agreement among data processing executives that they would welcome more management involvement in data processing. Half of the data processing executives say they would like to understand management goals better and two-thirds of them would welcome the opportunity to attend corporate officers' meetings. Yet their proposed solutions for improvements seem almost too blameless.

Just as corporate executives have done little to prepare themselves to communicate with or to guide processing executives, data processing people have been reticent about acquiring business knowledge. Perhaps they do wear rose-colored glasses which allow them mostly to view corporate executives as pleased enough with the contributions of data processing to the company. Perhaps they have become resolved to being regarded as perpetual underlings despite their control over a resource which is rapidly becoming the heart of communications in industry.

It is clear that data processing executives feel—with the possible exception of those in the banking industry—that senior-level management does not consider them knowledgeable enough in business to move into corporate management. They are strong in their belief that senior management views them as technicians with insufficient backgrounds in business. Yet they do precious little to correct the situation. Few data processing executives appear to have had much formal training in business subjects but perhaps more striking is the fact that few seem to care. Fewer than 20% express any desire to take business courses other than those that directly tie into their current jobs. Their apathetic present, it seems, is causing them to have little future.

Perhaps what writers have written and the 179 respondents to this study have said in retrospect is that the most common grounds for communication between corporate and data processing executives is their mutual unwillingness to take the first step to convince each other that they, in fact, have a great deal to learn from each other and a great deal more they could accomplish together.

One respondent to the study writes, "The most needed ingredient is an interactive, synergistic relationship between user and data processing people based on trust and mutual respect. Then each brings his own expertise. Neither need be expert in the other's field. The whole becomes greater than the sum of its parts."

Computers, in many ways, have become as much a frustration to management as a useful tool. Numerous writers point out that, instead of watching and controlling this investment, many senior executives have retreated to the safety of their expertise in other business areas, and, simply, delegated responsibilities for use of computers to lower-ranking executives. Corporate executives have trouble defining what they want; data processing personnel must reconcile technical factors that make a system work and that may alter what the user thinks he wants. Computer systems force organization and lines of communications changes which are disturbing to many corporate executives. This is particularly emphasized by major data base efforts, their placement in organizations and the accountability they require.

McKinsey and Company discovered in the 1960's that no company got effective use from its computers unless there was active participation by top executives. Alarmingly, 18 of 27 of the largest computer users were found to have marginal returns on their data processing investments. Things haven't changed much.

Top management's attention should be focused not only on the huge investments represented by computers, but also, on bringing into alignment plans for computer systems and plans for the business in general. Both computer systems plans and business plans should be reviewed in exactly the same manner.

Failure of top management to participate actively has an effect on the makeup and functions of the whole business organization, as well as its ability to adapt to change.

In the long run, no company can be better than its computer system. The question becomes: Who actually is designing and approving the computer systems that are central to the company—top executives or data processing people? The responsibility falls too often to data processing specialists who do not have the same business experience, training or overall perspective of corporate executives.

When responsibility for the design of systems falls to data processing executives, they tend to resist changes in systems and programs and, instead, tend to produce what is more comfortable for themselves, rather than what is most effective for the user.

McKinsey found that when proposals for new data processing applications were challenged as to their profitability, good answers rarely were available.

CONCLUSION

Poor definition and inadequate dissemination of corporate objectives seriously affect the definition of the company's information systems. Management not only must refuse to abdicate responsibility for computer systems, but should make a point of learning enough about computer systems to control them effectively.

Computer courses available to executives are generally too technical to be of any significant value. The purpose of training executives in data processing should be to encourage them to think of ways of improving existing applications and guiding the development of new ones that can provide relevant, useful information for decisions that minimize operating problems and optimize the potential for profit. It is absurd to teach corporate executives to program and then expect them to get intelligently involved in data processing planning. The bits and bytes of programming are at totally opposite ends of the "big picture" perspective corporate executives must keep. Interest, understanding, comfort and appreciation come from being able to read and use output, not from understanding how a program is written and run.

Data processing professionals, on the other hand, tend to limit their education to the extreme of bits and bytes. While some have moved into corporate executive jobs, most must be better educated in business matters before they can expect much personal growth in their companies. Consultants differ sharply on the likelihood of data processing professionals rising to the top of their organizations. Booz Allen Hamilton showed that data processing executives more often than not report to financial officers. Therefore, considering their lack of training and pertinent experience, there is little chance they could ever develop in areas of finance enough technical expertise to enable them eventually to succeed their bosses.

To follow the thinking of Robert Katz, at the first level of management the emphasis must be on technical and human skills. (For data processing people rising within a data processing career path, technical skills can mean data processing skills; for the data processing executive looking to succeed a financial executive, the required technical skills are most apt to change to financial skills—skills he usually does not possess.)

At higher levels of management, Katz sees the requirement for technical skills diminishing and the new emphasis placed on human and conceptual skills.

At the top, conceptual skill becomes the most important skill of all for successful administration.[13] (For top-level corporate executives, lack of conceptual skills may well extend to their views of the communications structure of their organization and of the impact of computer applications.)

Orlicky says it would be far wiser for an executive to learn something about systems and computers rather than depend on technical experts to learn the business.

According to corporate executives, if they were to take data processing courses, those they rate most important are: Computer Concepts, Using Models, Data Base Concepts, Project Planning and Estimating Techniques. Data processing executives agree except that they think a course in Systems Analysis would benefit senior executives more than the course in Estimating Techniques.

The five most important business courses for budding company presidents—according to corporate executives—are: Economics, Financial Accounting, Decision-Making, Finance and Writing. The five most important business courses for future data processing managers—selected by data processing executives—are: Decision-Making, Problem Solving, Business Information Systems, Writing and Financial Accounting. Ironically, fewer than 20% of the data processing executives responding to this study say they have any interest in pursuing business training.

Perhaps the greatest irony, though, is that we have evolved into a knowledge society for whom information processing technology has extended the speed, breadth and depth of our abilities to make decisions. As our ability to gather, interpret and disseminate information becomes more advanced, we should be reaching points of proficiency and optimization in business that will ensure continued competitiveness and profitability.

Certainly the caliber of people in business is improving constantly. The Dun & Bradstreet Directories show an impressive number of graduate degrees among the top executives of many corporations.

In terms of technology, we are at a highly advanced state in which we are capable of processing data at billionth-of-a-second speed and printing it at more than 1,000,000 characters per minute.

Nearly everyone employed in a white collar job is at least a high school graduate; certainly most people have had post-high school training in colleges, business schools or the military.

Why, then, is it that companies of all sizes and all industries, run by intelligent, educated people, staffed, as well, with educated and intelligent people at all levels, and supported by incredibly fast and reliable equipment, rarely can put together these resources into smooth and proficient organizations?

In most cases, computers are intended to provide the closest thing to perfect information within the constraints of time, available input and cost effectiveness. Certainly, no executive denies the value of good information and every executive wants his decisions based on the best information available.

Most astute executives also are aware of the communications filters that exist in every organization. Information that rises to the top is rarely pure; it is colored by individuals at every level who determine what is relevant, what is politically right and what they feel the individual above them ought to know. Usually, the information released from each level is that which can be defended or is championed by the individual who serves as the filter at that level. So it is with computer systems when top executives fail to assign direct accountability or, worse yet, fail to lend their insights, experience and knowledge to the thinking that results, ultimately, in computer systems which provide information essential for key management decisions.

By executives' recognition and by their own admission, computer professionals basically are technicians with limited expertise, at best, in business matters outside the data processing environment.

When a top corporate executive delegates design responsibilities down through his organization, there must be a point at which those given the responsibilities think they are asking for—or demanding—information that those above might want to know. Communication with data processing specialists may be dangerously dictatorial or it may run to any degree of delegation. It is, unfortunately, not uncommon for systems designers to be left with the responsibility for determining how flexible a system ought to be, the kind and format of data to be handled and the kind and formats of reports to be issued on the subject and within the time schedule set by the user. Most systems designers, given this opportunity, will revert to previous successes—implemented systems on which complaints were minimal or at least not loud—and will use methods and procedures which have worked before.

The end result, predictably, and frequently, is that the user receives reports on subjects of importance to him in a format that he and his staff may find difficult to understand. In many cases, such reports contain infinitely more data—and pages—than they need to contain—and they are more a source of confusion than help. Data in most systems exists, not because it is necessary, but because some corporate executive or some data processing professional thinks someone, someday, may need it. Instead of producing clean, informative and useful reports, then, such systems introduce nothing more than a printout of a filter system and, as such, contribute little to the decision-making capabilities of high-level executives.

The point comes back to someone at the top knowing—or wanting to know—what computers in his company are being used for, what information is being produced, and what decisions are now able to be made that, perhaps, could not have been made within previous systems. They need to ask—or advise on—how these information systems and decisions can be further improved. Most importantly, they must create the communications climate that first informs those below them of their interest in what kinds of usable information are being produced in return for data processing expenditures, and they need to establish the mechanism for feedback to ensure that their interests and desires are, in fact, being satisfied.

Two specific conversations I participated in may serve to illustrate the communciations gaps that exist and how these gaps inhibit the education and potential responsiveness of the people concerned.

A vice president of a multi-billion dollar financial company was asked about scheduling a course in computer concepts and applications for the top executives of his company.

"Top down training makes the most sense," he advised, "we'll start the program with vice presidents, as a group, and work down to supervisory levels."

I reminded him politely that it seemed more important and appropriate to have the chairman and president trained than anyone else, and therefore, to start at the very top.

"In this company," he said, "no one suggests to the chairman that he needs training!"

I suggested that, given the company's annual data processing investment of over $20 million, that we might invite the chairman to attend.

"No, you won't or you won't train anyone in this company," he answered threateningly.

"Perhaps you need to consider," I suggested, "that better use of your computers can mean more business but, also, failure to control the direction and performance of your computer systems can lead to long-term problems. If computers fail to keep you competitive or if you lose control over the information you need to run your business—or if for any reason someday you couldn't get it—your ability to compete effectively in your markets could be sharply diminished. Doesn't that make the subject important to the chairman?"

He answered simply and directly, "When and if the chairman wants training he will tell us. That is highly unlikely. . .".

In another conversation, a manager of systems and programming said that he would like to take courses in Accounting and Statistics.

"Why don't you?" I asked.

"Can't afford them," he answered.

"Doesn't your company have a tuition assistance program?" I asked, sure that it did.

"Sure it does," he answered, "but they'd never go along with me taking these kinds of courses. I'm supposed to know this stuff somehow. I'll wait until I don't need the money for something else. Then I'll do it and, if they find out about it, it'll be because I'm answering their questions smarter."

Those conversations were real. They may or may not be typical. Both persons worked for large, FORTUNE-listed companies.

Their attitudes and fears may not accurately reflect the thinking of the chief executives of their companies, but they do point to basic failures in communicating corporate thinking and in top management's sensitivity to such communciations problems.

The questions to resolve, I think are what education management ought to get, but perhaps even more fundamentally, how can management be made to recognize the impact of poor communications on the information systems of their companies?

A corporate officer of a major oil company, which is one of the largest users of computers in the world, wrote incredulously, that no one individual in his company is versed enough in their widespread use of computers to answer questions about their costs and effectiveness. If that is true, then certainly, it ought to raise questions about communications and the placement of controls in the organization.

The solution to the problems discussed in this study rest in the basic thinking of top level corporate executives who, to again paraphrase Robert Katz, need to be able to see computers in clear, conceptual terms as central resources and who may not be equipped to do so.

If training is a partial answer, then business schools, early in the careers of executives, must teach concepts and applications and challenge young executives to think creatively about the uses of computers. Courses should not be technical and should not be aimed at winning the sympathy of executives towards technical problems. And they should be taught by businessmen, not technicians.

If communications is a partial answer, then executives need now to look at their chains of command and their charts of organization from a communications viewpoint. They must determine at what levels decisions are being made on what information is to be produced, and they must identify the filters. Top managers should be sure they are getting enough direct information and should look for ways in which computer systems might be used better to produce clearer, more useful information.

Perhaps even more importantly, executives need to look at what computer systems exist in their organizations and why. They ought to review them thoroughly enough to be satisfied they are cost justified.

REFERENCES

1. U.S. Department of Commerce, *U.S. Industrial Outlook—1976,* U.S. Government Printing Office, Washington, D.C.
2. Quantum Science Corporation, *Network Information Services Management Action Summary,* New York, 1969.
3. Frost & Sullivan, Inc., "Markets for Insurance Computer Systems & Services," New York, 1974.
4. Sanders, Donald H., *Computers and Management,* New York, McGraw-Hill, 1970, p. 3.
5. Newspaper Enterprise Association, *The 1977 World Almanac,* New York.
6. U.S. Department of Labor, *Occupational Outlook Handbook,* 1975-76 Edition, Bureau of Labor Statistics, Washington, D.C.
7. Time, Inc., "Fortune Double 500 Directory," New York, 1976.
8. American Federation of Information Processing Societies, "The State of the Comupter Industry in the United States," Montvale, N. J., 1973.
9. Marketing and Research Services Department, Hitchcock Publishing Co., "1975 Salary Survey," *Infosystems,* June 1975.
10. Dean, Neil J., "The Computer Comes of Age," *Computers and Management,* New York, McGraw-Hill, 1970, p. 194.
11. Diebold, John, *Business Decisions and Technological Change,* New York, Praeger Publishers, 1970.
12. Orlicky, Joseph, *The Successful Computer System,* New York, McGraw-Hill, 1969.
13. Katz, Robert L., "Skills of an Effective Administrator," *Harvard Business Review,* September-October 1974.

APPENDIX A—Questionnaire Results

QUESTIONNAIRE
to
Corporate Executives

If you care to comment on any point, please feel free to use the back of any page or to add your own pages. All comments will be greatly appreciated, particularly if they can be quoted.

1. What is your position in the company?
   3.7% Chairman of the Board 14.3% President 6.1% Executive Vice President
   11.0 Senior Vice President 26.8% Vice President (Staff)
   9.8% Vice President (Line) 3.7% Director of Planning 8.5% Controller
   16.1% Other (please specify)

2. Who has final responsibility for approving major applications of computers to areas of your company's operations?
25.6% President 42.7% Officer Immediately Above Areas Being Affected
7.3% Controller 3.7% Data Processing Manager
20.7% Other (please specify)

3. To whom does your company's data processing manager report?
13.4% President 48.8% Controller or Financial V.P. 18.3% Administrative V.P.
0 Personnel Director 19.5% Other (please specify)

4. Who has the main responsibility for deciding what data processing education corporate executives should have?
23.2% President 18.3% Administrative V.P. 3.7% Training Director
20.7% Data Processing Manager 0 Internal Consultant
23.2% Other (please specify)

5. How much a part of your daily decision-making is a direct result of computerized reports?

11.0%	None	30.5%	26-50%	0%	76-99%
52.4%	1-25%	3.7%	51-75%	0	All

6. Are you generally satisfied with the computerized information you receive?
73.2% Yes 24.4% No

7. How would the effectiveness of your decisions change if you no longer received computerized information?

Greatly Improved	Improved Somewhat	Unchanged	Weakened	Considerably Weakened
1.2%	0%	18.3%	48.8%	30.5%

8. Consultants claim that 40% of all companies operated more efficiently with manual systems than they do now with computer systems. How do you feel?
4.9% Strongly Agree 9.8% Agree 13.4% No Opinion 46.3% Disagree 25.6% Strongly Disagree

9. How do you generally assess your company's performance with computers compared to its performance before computers?

Much Less	Less	About	Generally	Much More
Efficient	Efficient	the Same	More Efficient	Efficient
1.2%	2.4%	13.4%	57.3%	20.7%

10. How do you rate your own knowledge of data processing?
8.5% Expert 11.0% Fluent 42.7% Generally Knowledgeable 30.5% Informed 6.1% Light 1.2% None

11. How do you rate your involvement in the development or redesign of information systems?

Take Total	Take Part	Take Part	Approve	Delegate	Do Not
Charge	Throughout	Periodically	Main Steps	Control	Take Part
2.4%	11.0%	28.0%	29.3%	20.7%	8.5%

12. How do you generally rate the involvement of your company's other executives?

Take Total	Take Part	Take Part	Approve	Delegate	Do Not
Charge	Throughout	Periodically	Main Steps	Control	Take Part
0%	7.3%	31.7%	29.3%	23.2%	4.9%

13. If you could select the business courses for the college freshman who will be president of your company in 30 years, how would you rate these? Please indicate which courses you have taken and which you would like to take.

	Wasteful	Low Priority	Useful	High Priority	"Must" Knowledge	I Have Taken	I Would Like to Take
Economics	0%	0%	25.6%	32.9%	35.4%	84.1%	1.2%
Financial Accounting	0	0	25.6	34.1	34.1	76.8	3.7
Business Policy	0	6.1	34.1	25.6	20.7	51.2	11.0
Management Psychology	1.2	8.5	37.8	22.0	24.4	56.1	9.8
Marketing	0	2.4	37.8	35.4	18.3	61.0	14.6
Finance	0	1.2	25.6	40.2	24.4	72.0	2.4
Management Theory	2.4	14.6	36.6	12.2	25.6	50.0	8.5
Problem Solving	1.2	6.1	32.9	35.4	15.9	41.5	18.3
Principles of Auditing	11.0	43.9	28.0	4.9	2.4	41.5	6.1
Theory of the Firm	1.2	35.4	26.8	11.0	7.3	24.4	13.4
Writing	1.2	3.7	25.6	36.6	25.6	62.2	8.5
Decision-Making	0	7.3	25.6	32.9	28.0	34.1	17.1
Public Speaking	3.7	4.9	25.6	42.7	17.1	52.4	9.8
Managerial Accounting	2.4	9.8	35.4	31.7	13.4	56.1	7.3
Business Information Systems	1.2	8.5	41.5	31.7	9.8	37.8	18.3
Operations Research Techniques	2.4	29.3	41.5	11.0	4.9	31.7	22.0
Statistics	2.4	14.6	51.2	17.1	6.1	64.6	8.5
Business Cycles & Forecasting	0	15.9	43.9	22.0	7.3	45.1	15.9

14. If you were planning a curriculum for senior-level corporate executives to learn computer concepts and applications, how would you rate these? Please indicate which courses you have taken and which you would like to take.

	Wasteful	Low Priority	Useful	High Priority	"Must" Knowledge	I Have Taken	I Would Like to Take
Computer Concepts	2.4%	3.7%	20.7%	20.7%	43.9%	62.2%	7.3%
Programming Concepts	19.5	29.3	25.6	12.2	3.7	36.6	8.5
Operating Systems	20.7	25.6	30.5	7.3	3.7	22.0	11.0
Hardware Configurations	19.5	35.4	19.5	8.5	2.4	20.7	6.1
Data Base Concepts	2.4	15.9	32.9	29.3	7.3	25.6	17.1
Introduction to Teleprocessing	8.5	22.0	43.9	13.4	0	23.2	14.6
Using Models	6.1	13.4	35.4	24.4	11.0	31.7	20.7
Estimating Techniques	7.3	30.5	39.0	8.5	4.9	14.6	14.6
Project Planning	2.4	13.4	43.9	20.7	7.3	15.9	12.2
Algorithmic Processes	23.2	36.6	22.0	0	1.2	3.7	7.3
Data Communications	6.1	32.9	31.7	13.4	0	14.6	11.0
Systems Analysis	6.1	22.0	40.2	11.0	7.3	20.7	9.8
Systems Design	4.9	35.4	26.8	13.4	2.4	18.3	11.0
Computer Systems Architecture	29.3	35.4	13.4	6.1	0	7.3	4.9
Real-Time Systems	8.5	35.4	30.5	9.8	1.2	14.6	8.5
Systems Simulation	18.3	35.4	23.2	3.7	1.2	11.0	3.7
COBOL	42.7	28.0	8.5	1.2	1.2	11.0	1.2
Assembler Language	47.6	28.0	4.9	1.2	1.2	11.0	2.4
FORTRAN	42.7	29.3	8.5	2.4	0	15.9	1.2
Computer Organization	14.6	23.2	31.7	8.5	4.9	13.4	4.9
Other	0	0	0	0	0	0	0

15. How satisfied are you with the amount of information you get from your company's computer systems?

Very Satisfied	Satisfied	Uncertain	Dissatisfied	Very Dissatisfied
8.5%	58.5%	8.5%	15.9%	2.4%

16. How satisfied are you with the quality of the information you get from your company's computer system?

Very Satisfied	Satisfied	Uncertain	Dissatisfied	Very Dissatisfied
13.4%	61.0%	7.3%	14.6%	2.4%

17. Which of these most closely summarizes your overall feelings about your company's data processing effort? (Check as many as apply.)
14.6% Highly Efficient
18.3% Inconsistent
6.1% Insensitive to My Needs
68.3% Generally Effective
18.3% Intelligent
24.4% Highly Dependable
9.8% Invaluable
1.2% Worthless
8.5% Too Slow
_____Other (Please specify)

18. How well do your company's data processing personnel understand your activities?
7.3% Not Well at All 11.0% Minimal 48.8% Generally O.K. 29.3% Quite Well 2.4% Expertly

19. How would you generally rate their knowledge of other corporate areas?
4.9% Not Good 18.3% Minimally 46.3% Generally O.K. 28.0% Quite Good 0 Expert

20. Which of these applies to your company's data processing organization?
(Check all that apply.)
Overall performance is
81.7% Satisfactory 13.4% Unsatisfactory
Implementation of new applications is
36.6% Rarely on Schedule 61.0% Usually on Schedule
Costs are consistently
54.9% Within Budget 35.4% Over Budget
Communications between data processing people and users is
67.1% Good 28.0% Not Good
Turnover among the most competent data processing personnel is
11.0% Too High 85.4% Under Control

21. Data processing would do a much better job in my company if (check all that apply.)
14.6% applications were more realistic.
28.0% there was more corporate management control.
24.4% data processing people had more knowledge of our business.
24.4% data processing operated as a cost/profit center.
43.9% planning were better.
32.9% data processing management had more business knowledge.
74.4% corporate managers knew a little more about data processing.

22. How could communications between corporate and data processing managers in your company be improved? (Check all that apply. Please circle those that are done in your company.)

Say	Do	
46.3%	17.1%	Frequent meetings
18.3%	3.7%	Swap jobs occasionally
13.4%	3.7%	Attend the same courses
15.9%	4.9%	Have the data processing manager attend all corporate officer meetings
17.1%	3.7%	Have your people visit the computer room periodically
67.1%	13.4%	Have joint staff meetings to explain new applications and responsibilities

__Other? ____
____
____

23. How well do your company's data processing personnel identify with the company and its goals?
0 Not at All 28.0% Not Enough 18.3% Don't Know 42.7% Strongly 9.8% Very Strongly

24. Can data processing professionals realistically expect to succeed into corporate (non-data processing) management in your company?
42.7% Yes 31.7% Don't Know 25.6% No

25. Have data processing people ever moved into corporate management jobs in your company?
35.4% Yes 9.8% Don't Know 52.4% No

If so, into what job(s)? ______________________________

____

If not, what holds them back? (Check all that apply.)
35.4% Their training is too technical and has nothing to do with the rest of the business.
4.9% They are not interested in corporate management jobs.
31.7% They don't have the necessary business background.
7.3% They have enough trouble running their own shop.
__Other? ______________________________
____
____

26. About how large is your data processing organization?

People	Equipment	Overall Budget
42.9% Less than 100	28.8% Under $50,000/month	31.9% Under $2 million
27.0% 100-300	19.6% $50,000-100,000/month	18.4% $2-4 million
12.3% 300-500	14.1% $100,000-200,000/month	11.7% $4-6 million
8.6% 500-1,000	8.6% $200,000-300,000/month	6.7% $6-8 million
4.9% More than 1,000	5.5% $300,000-400,000/month	4.9% $8-10 million
	3.7% $400,000-500,000/month	3.7% $10-12 million
	12.9% More than $500,000/month	3.7% $12-16 million
		12.9% More than $16 million

27. What is your company's main area of business?
65.3% Industrial 4.1% Retailing
4.1% Banking 8.2% Transportation
6.1% Insurance 6.1% Utility
2.0% Financial 4.1% Consultant Other (please specify)

QUESTIONNAIRE
to
Data Processing Executives

If you care to comment on any point, please feel free to use the back of any page or to add your own pages. All comments will be greatly appreciated, particularly if they can be quoted.

1. What is your official title?
17.3% V.P., Data Processing 27.2% Director, Management Information Systems
8.6% Director of Data Processing 8.6% Manager of Data Processing
4.9% Manager, Systems and Programming
32.1% Other (please specify)

2. Who has final responsibility for approving major applications of computers to areas of the company's operations?
17.3% President 38.3% Officer Immediately Above Areas Being Affected
13.6% Controller 24.7% D.P. Manager
25.9% Other (please specify)

3. To whom do you report?
8.6% President 18.5% Administrative V.P. 3.7% Personnel Director
40.7% Controller or Financial V.P.
28.4% Other (please specify)

4. Who has the main responsibility for deciding what data processing education corporate executives should have?
12.3% President 4.9% Training Director 0% Internal Consultant
11.1% Administrative V.P. 46.9% D.P. Manager
21.0% Other (please specify)

5. How much do you estimate that senior executives' decision-making is a direct result of computerized reports?
1.2% None 24.7% 26-50% 6.2% 76-99%
39.5% 1-25% 22.2% 51-75% 0 All

6. Are your company's executives generally satisfied with the computer output they receive?
81.5% Yes 13.6% No

7. How would the effectiveness of their decisions change if they no longer received computer reports?

Greatly Weakened	Weakened	No Change	Somewhat Improved	Greatly Improved
34.6%	53.1%	6.2%	1.2%	0%

8. Consultants claim that 40% of all companies operated more efficiently with manual systems than they do now with computer systems. How do you feel?
2.5% Strongly Agree 6.2% Agree 9.9% No Opinion 34.6% Disagree 42.0% Strongly Disagree

9. How do you rate your knowledge of business management subjects?
9.9% Expert 39.5% Fluent 39.5% Generally Knowledgeable 6.2% Informed 0% Light 1.2% None

10. How do you rate the involvement of senior management in your company in the development or redesign of systems?
0% Take Total Charge 7.4% Take Part Throughout 24.7% Take Part Periodically
27.2% Approve Main Steps 30.9% Delegate Control 3.7% No Interest in Taking Part

11. If you were planning an unlimited curriculum for senior corporate executives to learn enough about computer concepts and applications to work closely with you and your people, how would you rate these?

	Wasteful	Low Priority	Useful	High Priority
Computer Concepts	4.9%	8.6%	42.0%	38.3%
Programming Concepts	32.1	29.6	22.2	8.6
Operating Systems	44.4	34.6	13.6	1.2
Hardware Configurations	24.7	43.2	22.2	2.5
Data Base Concepts	4.9	17.3	49.4	22.2
Introduction to Teleprocessing	14.8	27.2	39.5	11.1
Using Models	3.7	21.0	44.4	23.5
Estimating Techniques	17.3	32.1	32.1	11.1
Project Planning	8.6	22.2	35.8	25.9
Algorithmic Processes	43.2	35.8	9.9	3.7
Data Communications	17.3	38.3	34.6	3.7
Systems Analysis	12.3	30.9	35.8	14.8
Systems Design	17.3	35.8	30.9	9.9
Computer System Architecture	43.2	37.0	11.1	1.2
Real-Time Systems	18.5	34.6	38.3	1.2
System Simulation	30.9	35.8	22.2	2.5
COBOL	72.8	18.5	1.2	1.2
Assembler Language	79.0	13.6	1.2	0
FORTRAN	71.6	21.0	1.2	0
Computer Organization	22.0	27.2	31.2	11.1%
Other	0	0	0	0

12. If you could select the business courses for the college freshman who will be your company's data processing manager in 20 years, how would you rate these?

	Wasteful	Low Priority	Useful	High Priority
Economics	2.5%	16.0%	50.6%	19.8%
Financial Accounting	0	7.4	35.8	46.9
Business Policy	1.2	6.2	46.9	35.8
Management Psychology	0	7.4	38.3	44.4
Marketing	1.2	22.2	53.1	13.6
Finance	0	12.3	56.8	21.0
Management Theory	1.2	11.1	42.0	35.8
Problem Solving	0	8.6	18.5	63.0
Principles of Auditing	3.7	22.2	44.4	18.5
Theory of the Firm	6.2	21.0	42.0	16.0
Writing	0	6.2	33.3	50.6
Decision-Making	0	1.2	27.2	61.7
Public Speaking	0	9.9	34.6	45.7
Managerial Accounting	0	7.4	48.1	34,6
Business Information Systems	0	3.7	30.9	55.6
Operations Research Techniques	4.9	28.4	25.9	28.4
Statistics	2.5	19.8	42.0	24.7
Business Cycles & Forecasting	3.7	19.8	44.4	21.0

13. How satisfied are d.p. users in your company with the amount of information they get from computer systems?

Very Satisfied	Satisfied	Uncertain	Dissatisfied	Very Dissatisfied
6.2%	74.1%	12.3%	4.9%	0%

14. How satisfied are d.p. users in your company with the quality of information they get from computer systems?

Very Satisfied	Satisfied	Uncertain	Dissatisfied	Very dissatisfied
16.0%	67.9%	12.3%	3.7%	0%

15. Which of these most closely summarize your overall feelings about corporate management's involvement in the data processing effort? (Check all that apply.)

2.5% Highly Efficient
42.0% Inconsistent
4.9% Insensitive to My Needs

50.6% Generally Effective
30.9% Intelligent
11.1% Highly Dependable
____Other (please specify)

11.1% Invaluable
1.2% Worthless
9.9% Disinterested

16. How well do your people understand the workings of your company?
2.5% Expertly 37.0% Quite Well 40.7% Generally O.K. 17.3% Minimally 0% Poorly

17. How would you rate executives' knowledge of data processing in your company?
0% Expert 13.6% Quite Good 39.5% Generally O.K. 38.3% Minimal 6.2% Not Good

18. How do you think your company's corporate executives would rate the data processing organization?
Overall performance is
93.8% Satisfactory 4.9% Unsatisfactory
Implementation of new applications is
27.2% Rarely on Schedule 70.4% Usually on Schedule
Costs are consistently
76.5% Within Budget 21.0% Over Budget
Communications between data processing people and users is
80.2% Good 17.3% Not Good
Turnover among the most competent data processing personnel is
6.2% Too High 90.1% Under Control

19. Data processing could do a much better job in my company if (check all that apply.)

29.6% application requests were more realistic.
56.8% corporate managers got more involved.
37.0% corporate managers had more knowledge of data processing applications.
29.6% the department operated as a cost/profit center.
51.9% planning were better.
32.1% data processing staff knew more about user applications.
65.4% users took part in systems development all the way through the project.
46.9% we had a clearer understanding of management goals.
2.5% corporate managers were less involved.

20. How could communications between corporate areas and data processing be improved? (Check all that apply.) Please circle those that are done in your company.

Say	Do	
50.6%	28.4%	Frequent meetings
33.3%	7.4%	Swap jobs occasionally
24.7%	14.8%	Attend the same courses
64.2%	23.5%	Have data processing mangers attended corporate officer meetings
35.8%	16.0%	Have Your people visited user areas periodically
58.0%	34.6%	Have user personnel visit computer room periodically
75.3%	38.3%	Have joint staff meetings to explain new applications and responsibilities

__Other? ______________________________
____
____

21. Data processing people frequently are accused of being loyal only to the data processing profession, not necessarily to the company they work for. How well do your company's data processing personnel identify with the company and its goals?
2.5% Not at All 34.6% Not Enough 7.4% Don't Know 46.9% Strongly 4.9% Very Strongly

22. Can data processing professionals realistically expect to succeed into corporate (non-data processing) management in your company?
54.3% Yes 19.8% Don't Know 22.2% No

23. Have data processing personnel ever moved into corporate management jobs in your company?
53.1% Yes 6.2% Don't Know 38.3% No
If so, into what job(s)? ______________________________
____

If not, what holds them back? (Check all that apply.)
13.6% Their training is too technical and has nothing to do with the rest of the business.
11.1% They are not interested in corporate management jobs.
18.5% They don't have the necessary business background.
34.6% Corporate management apparently doesn't consider them good candidates.
__Other? ______________________________
____
____

24. About how large is your data processing organization?

People	Equipment	Overall Budget
42.9% Less than 100	28.8% Under $50,000/month	31.9% Under $1-2 million
27.0% 100-300	19.6% $500,000-100,000/month	18.4% $2-4 million
12.3% 300-500	14.1% $100,000-200,000/month	11.7% $4-6 million
8.6% 500-1,000	8.6% $200,000-300,000/month	6.7% $6-8 million
4.9% More than 1,000	5.5% $300,000-400,000/month	4.9% $8-10 million
	3.7% $400,000-500,000/month	3.7% $10-12 million
	12.9% More than $500,000/month	3.7% $12-16 million
		12.9% More than $16 million

25. What is your company's main area of business?

67.9% Industrial
6.2% Banking
7.4% Insurance
3.7% Financial
3.7% Retailing
4.9% Transportation
6.2% Utility
_____Other (please specify)

APPENDIX B—Business Management and Data Processing Courses Ranked by Executives According to Importance

BUSINESS MANAGEMENT COURSES
Ranked in Order of Importance

for Future Corporate Presidents	for Future Data Processing Manager
Economics	Decision-Making
Financial Accounting	Problem Solving
Decision-Making	Business Information Systems
Finance	Writing
Writing	Financial Accounting
Management Psychology	Management Psychology
Problem Solving	Public Speaking
Marketing	Business Policies
Public Speaking	Managerial Accounting
Managerial Accounting	Management Theory
Management Theory	Finance
Business Information Systems	Statistics
Business Policies	Economics
Business Cycles & Forecasting	Business Cycles & Forecasting
Statistics	Marketing
Operations Research Techniques	Principles of Auditing
Theory of the Firm	Operations Research Techniques
Principles of Auditing	Theory of the Firm

DATA PROCESSING COURSES
Ranked in Order of Importance
for Senior-Level Executives

by Corporate Executives	by Data Processing Executives
Computer Concepts	Computer Concepts
Using Models	Data Base Concepts
Data Base Concepts	Using Models
Project Planning	Project Planning
Estimating Techniques	Systems Analysis
Introduction to Teleprocessing	Introduction to Teleprocessing
Systems Analysis	Estimating Techniques
Systems Design	Systems Design
Programming Concepts	Computer Organization
Data Communications	Data Communications
Real Time Systems	Real Time Systems
Computer Organization	Programming Concepts
Operating Systems	Hardware Configurations
Hardware Configurations	System Simulation
System Simulation	Operating Systems
Algorithmic Processes	Algorithmic Processes
Computer Systems Architecture	Computer Systems Architecture
FORTRAN	FORTRAN
COBOL	COBOL
Assembler Language	Assembler Language

A community of individuals—Cooperation and individualization in computer science education

by KENNETH L. MODESITT
Indiana University-Purdue University
Fort Wayne, Indiana

ABSTRACT

A model of education based on cooperation among responsible individuals is presented. Earlier models are discussed which involve traditional competitive-based courses as well as the more recent individualized mode. The latter has been used by the author for several years, utilizing both a Personalized System of Instruction (PSI) and Computer-Based Education (CBE). Now, however, cooperation has been introduced as an explicit and integral part of education. This was done in the belief that most serious problems of our day are amenable to a cooperative problem-solving process. The techniques utilized in an introductory computer science course include: study partners for PSI units, computerized personal data base design by a small group, mutual design and use of interactive programs, use of cooperative exercises (computer and otherwise) and parties. The future of these efforts remains unknown, but there is a strong belief that cooperation is a preferable model to competition in the world to come.

INTRODUCTION

In this paper, the concepts of community and the individual are compared and contrasted. The realization of these concepts has been implemented in the classroom by maintaining a delicate balance between cooperative efforts and individualized instruction. Preserving this balance is advocated as a preferable alternative to either the traditional lecture mode or, more recently, the primarily individualized mode of instruction. Considerable experience has been gained with the individualized mode, utilizing both Personalized Systems of Instruction (PSI) and Computer-Based Education (CBE) over a period of several years. The explicit introduction of cooperation into the computer science classroom is of more recent vintage. It comes in response to a belief that cooperative efforts are most likely to solve the difficult problems of today, and to fears of interpersonal isolation which may result from exclusive or primary reliance on individualization.

DEFINITIONS[17]

Community—	a social group or class having common interests
Common—	belonging equally to two or more; shared by all alike; joint
Cooperate—	work together toward a common end or purpose
Cooperation—	an association of persons for mutual benefit
Individual—	of or relating to a single human being; by or for one person
Individualize—	to modify or suit a particular entity

WHY IS A COMMUNITY OF INDIVIDUALS DESIRABLE?

Whenever I meet a class for the first time, one of the first questions I ask them is: "Why?" "Why are you in this course?" "Why are computers becoming such an integral part of our lives?" "Why is understanding their uses and potential so important?" To me, asking "Why" is an essential first step in looking at a new idea. No particular solution is guaranteed to result. Rather, many times, an increased awe and reverence for the phenomenon results. Nevertheless, when the phrase "community of individuals" first began to have some significance for me as an instructor, *I* needed to ask, and perhaps answer, the "Why" question.

I believe that a community of individuals, working together in cooperation toward mutual goals represents the most hopeful and realistic structure whereby the majority of the important problems confronting us today can be solved. The difficulties and enigmas which surround us today are legion and increasingly severe. For cogent and striking presentations of the present and possible futures, recent issues in The Portable Stanford Series are outstanding.[6,18,20] Of particular import is Harman's *An Incomplete Guide to the Future*. Wars, unemployment, famine, inflation, civil rights, crime, welfare, corruption—the list appears endless.

Few of these, if any, are solvable by one person. What are the alternatives: to continue to accept such problems as "natural" or treat them as "inconveniences"? *I*, at least am not willing so to do. Rather, it is my firm belief that a dedicated community of strong individuals offers a viable and desirable alternative.

ALTERNATIVE MODELS IN EDUCATION

The belief in a dedicated community of strong individuals is manifested within the framework of introductory computer science sources at Indiana University-Purdue University at Fort Wayne. I strongly believe in the impact that models have on our later lives. Hence, I try to include in classroom situations those values which have positive significance for me, i.e., community of and cooperation among mature persons. It is this model I wish students to consider seriously now and in later life.

Students will definitely have other models from which to choose. One of the most common and powerful ones is based on competition, and is usually found in a traditional lecture-oriented course. In earlier papers, my thoughts were shared about where and when competition may be appropriate, e.g., sports, and where it is *not* appropriate, e.g., education.[13,14] In a recent address, Howard Casmey, Minnesota commissioner of education, noted that education today is "predicated on failure."[1] However, a grade of A or B in a course must indicate that a student has demonstrated mastery of the subject. It must *not* indicate that she/he is in the top 10 percent who demonstrate mastery in a predefined, usually short, time frame, e.g., an hour exam, a two hour final, a one semester course, or a four year degree, etc.

So we are led naturally to another, and rapidly growing, model in education: that of individualization. To again quote Casmey,

> "To me 'individualized instruction' connotes 'mastery learning.' It's neither a concept nor a process in and of itself, but rather a concept that requires a process. It is the great humanizing factor in education, because failure is no longer possible. *All students now succeed; some at a faster or slower rate than others, but all learn and all succeed.*"
>
> "If one of our children has more difficulty in learning to tie his shoe, and does not accomplish this task until he is six years old, we do not label the child as a failure. We know that, barring any specific physical problems, he will be able to accomplish this task as his manipulative skills develop."
>
> "This is what the educational process should be and *we've known this for 50 years.*"

Casmey goes on to espouse the use of computer-based education. Personalized systems of instruction are also vitally concerned with individualized instruction. This model is leading to persons increasingly able to become adults, i.e., persons who assume responsibility for their own actions. It is a model I view with great respect and admiration. I have used it for the last four years in several courses. It continues as a vital and integral part today. However, and here I quote the frustrated and honestly concerned housewife who, when asked about her housework, replied, "It's good, but *it's just not enough*!" As I recall the story, she also went on to say she needed someone to share in those responsibilities.

My feelings about individualized instruction are similar: it's good; I like it; I encourage my students to use it; but it is simply not enough! The process *does* promote a more adult, independent attitude. Students *do* gain an acceptable *and* accurate self-image: "I *can* learn this subject pretty much on my own." To many people, young and old alike, this is a dramatic change from "I'm dumb. I never could do math (or English or physics, etc.)." In fact, it was the latter type statement that first prompted me to consider seriously a major shift in my courses from lecture to individualized.

However, it has been several years since that time, and I've come to realize that strong individuals, *by themselves*, do not constitute a complete picture. You and I really do live in a community, several of them in fact. I do not wish to live a life apart from others. I wish to participate, as a strong person, in mutual efforts with the other strong people to solve significant problems *and* to share with them at increasingly open and intense levels. The latter quality is not a "teachable" one in my opinion, but rather flows naturally from significant mutual content-based concerns. Therefore, it is this model of a dedicated community of responsible individuals that I value highly, and consequently emphasize in many of my courses.

COOPERATIVE COURSE EFFORTS

The course in which major efforts have been expended to incorporate cooperation is CS 220, Introduction to Algorithmic Processes. Section enrollment is usually between 20 and 25, and consists primarily of science majors ranging in age from 18 to 30. It represents the first direct exposure to computers for all but those with some experience in high school. A course which several take after CS 220 is CS 461, Algorithmic Languages, which relies extensively on individualized instruction.

The primary cooperative efforts in CS 220 include the following: study partners for PSI units, data base design by small groups, mutual design and use of interactive programs, use of cooperative exercises, and parties.

Study partners for PSI units

As is well known, a personalized system of instruction, also known as the Keller plan, has become one of the most noteworthy and exciting instructional methods in recent years. Earlier referenced papers of mine discuss it at some length, as well as in References 15 and 16. It is appearing in more and more university settings, both as documented from within and without. From within, the Center for Personalized Instruction at Georgetown University holds annual conferences.[21] They also publish a newsletter[19] and a journal.[11] All these publications relate the increased use and very positive results in final exam scores, retention, transfer, attitudes, facilitation, and self-image. From without, recent issues of *Change* magazine contain several articles on PSI as one of the most promising innovative educational concepts.[4,12]

An integral part of PSI is breaking material down into digestable pieces or units. For CS 220, there are 15 such units (two of them review ones), plus a final. Within the normal PSI framework, each student works through the objectives, suggested procedure, and unit test by herself/himself. Within the cooperative PSI framework, each student is assigned a partner on the first day of the course. The partner is usually of the opposite sex, roughly the same year in school, but not necessarily in a related discipline. During the next couple of class days, the partners introduce each other to the class informally.

The PSI partners are encouraged to work together on the objectives and suggested procedure of each unit. The evaluation, of course, is administered separately. Since most of the units involve designing and implementing computer programs, a variant of "egoless" programming is suggested.[2]

The partnership is particularly beneficial in the first several weeks. The intricacies of working with strange machines and procedures are not as intimidating to two people as to one. Therefore, when the units become more concerned with concepts instead of interactive terminals, batch jobs, remote job entry, keypunches, etc., partnerships may deepen, dissolve, or reform. By this time, the partners are often working at different rates. So, they may feel free to work with others, or in some cases, by themselves. In any case, all the students now have first-hand experience in cooperating with another for a common goal.

As those familiar with PSI know, the peer tutors or proctors perform many of the roles just mentioned. The three peer tutors for CS 220 are excellent, in my opinion. The students view them as "willing-to-help" experts, and have no hesitation about asking them questions. However, the tutor is *not* a true partner—she/he has been through the course earlier, whereas the student is still working on it. Hence, I believe both the tutor and partner have valuable roles to play in the learning process.

Automated personal data base design by small groups

After nearly one-half of the semester has passed, most students have a working knowledge of the primary programming language concepts. It then becomes feasible to introduce them to a design project. Since such projects outside the academic world are usually cooperative in nature, this is a good opportunity for small groups of three to four to work together. I wanted the design to be for a realistic area and one of some importance. Automated (computerized) personal data banks immediately come to mind. A group in the College of Education at the University of Illinois under the direction of Bruce Hicks has also investigated such issues.[9] In CS 220, however, the students are not only users of such data banks, they will also be designers thereof.

The sample list of such data banks included in the appendix is almost solely student-generated. Here is an application in which interest is quite high. Outside speakers come in to discuss the issues of privacy and society's need for organization. Students have the necessary skills to build such a data base. The task lends itself readily to sharing ideas and feelings, especially about one's right to privacy. The complete assignment is given in the appendix.

Mutual design and use of interactive programs

After some introductory simple programs which all students are to complete, students are given a choice. They may either work on problems assigned in the unit, or they may choose one of their own, so long as it is of comparable complexity. Several people have opted for the latter. Here they can use the computer directly for a problem in their discipline, or they can design and implement a game, simulation, drill and practice, etc.

In any case, whether the students work on a preassigned problem or one of their own, they are encouraged to seek out an "intelligent but naive" classmate to be a typical user. This user, who is often their partner, takes the program. She/he writes comments relating to directions, accuracy, style, etc., and then signs and dates the listing. She/he is not expected to examine the coding itself. If necessary, the author then modifies the program before turning it and the signed listing in as part of the unit test.

Students realize this as a cooperative venture—one day they may be the designer/coder, the next they may be the "naive" user. They expend considerable effort making certain their program executes properly *and* is designed for human use. Not entirely by accident, they are learning the components of good computer-based education lessons, both as a designer and a user.

Use of cooperative exercises, computer and otherwise

Until relatively recently, this topic virtually did not exist. However, there appears to be an increasing awareness that a model of cooperation is *at least* as important for education as that of competition. Star Trek, Moonwar, Battleship, etc., all have a certain fascination, but how about *cooperative* games also? In a recent issue of *Computers and Society*, Bruce Hicks discussed several such games.[10] "SOS" is available on PLATO and "Lost and Forgotten Island" is available in Basic. In these games, cooperation is espoused for the mutual benefit of all players. After playing these, students (*and* instructors) now have some models for community-building lessons. There have also been several programs in issues of *Creative Computing* (a *great* magazine!) which promote cooperation rather than shooting down Klingons.[3]

I do not restrict cooperation to be viable only when interacting with the computer. This is, however, an increasingly important type of CBE lesson, in my opinion. Students also share together in discussion about films. Controversial ones such as *The Right of Privacy* or *A Matter of Survival* (both available from Indiana University) prompt conversation in which significant and honest feelings are exchanged. People learn to relate to others who may have strong, but different, thoughts. Many of the exercises in *Values Clarification* by Simon, Howe, and Kirschenbaum are also useful in this domain.[22]

Recently, Judy Edwards, new president of the Association for Educational Data Systems (AEDS) brought to my attention the BLUE/GREEN classroom simulation dealing with cooperation within a company framework. This type of activity is growing in popularity as witnessed by the recent formation of the New Games Foundation. Quoting from the director, Pat Farrington, we have:

> "If people center on the joy of playing, cooperating and trusting rather than striving to win, they become part of the process—not spectators. . . . We try to create individual responsibility—helping people create their own games. . . . We're into cooperating rather than competing."[7]

It would appear that more and more people are realizing that cooperation is an increasingly vital element in our lives. That is why I am concerned that students have first-hand experience with this model.

Parties

One of the essential ingredients of community is knowing one another. Parties provide an excellent first step for learning one another on an informal and non-threatening basis. And, because they are so much fun, we try to have several!

The "content excuse" for most parties is the celebration of a student's birthday. Donuts and cider usually provide a common basis for sharing physical sustenance while the class continues. Frequently the student may indicate their special plans for the day.

The date of birth is ascertained from the first program taken by the students. An interview program Query, written in Basic, asks them for their favorite food, and tries to identify them by their preference in later interviews. It then asks them for their birthday, and goes on to ask if *they* have any questions. The latter feature is used as a note/comment/question file throughout the course. The student has the option of refusing to answer any question, e.g., for privacy reasons. She/he also has the option of terminating the interview and thus having no record kept.

These informal in-class celebrations allow us to learn a little more about a person than simply what courses they are taking. They are also supplemented by a get-together every semester, often at the instructor's home. Each person brings their appetizer speciality, and the evening is spent with good food and informal conversation. The latter often arises from a game that each person brings, as well as programs and games available over the interactive terminal I bring to the party. Again, people put out individual efforts to build for common goals and mutual benefit. Cooperation is not an impossible model. It is, I believe, a most possible, desirable, and necessary one for any world in which human beings form a part.

SUMMARY AND FUTURE

In summary, I have tried to indicate why I believe cooperation to be a valuable and integral part of education. It is compared to a traditional mode, usually based on competition, and then to the more recent mode of individualized instruction. The latter has often been implemented, in my courses and others, within a PSI environment and frequently involves computer-based education. These individualized techniques have done much to eliminate the idea that only 10 percent of students are able to master the material presented. However, most serious problems in the world are solved by a community of responsible people. For just one example, Howard Casmey, in the address referred to earlier, dwells at length on the necessity of partnership (cooperation) between companies and education for gaining widespread acceptance of quality CBE.

It is in this belief that cooperation is introduced explicitly in my introductory computer science courses. The primary cooperative efforts are:

(1) study partners of PSI units,
(2) design of a computerized personal data base by a small group,
(3) mutual design and use of interactive programs,
(4) use of cooperative exercises, computers and otherwise, and
(5) parties.

I am under no illusion that these are the only or best ways to introduce cooperation explicitly. Other techniques with which I am familiar include: computer clubs for hobbyists and micro-computer users, computer uses for elders (another project of Hicks),[8] designing programs for the school itself or local community, and, of course, the very widespread efforts in primary and secondary education. Learning exchanges, open classrooms, individually-guided education (IGE) programs, etc., are all oriented toward subject mastery. They also usually occur in a self-paced environment and often involve interpersonal exchanges.[5] I personally would be very grateful if the reader would share additional references for cooperative efforts, games, programs, etc.

What of the future? Only the future itself knows. But I would hope that my understanding of people and the learning process would grow and mature. It may well be that cooperative efforts are "not all they're cracked up to be." The values of one's own self and of one's God are probably pre-eminent. Yet at the same time, we live, love, work, and learn in a community of others. I am grateful for that opportunity, enjoy helping others, and enjoy being helped by a community of caring and responsible individuals.

REFERENCES

1. Casmey, Howard B., "Computer-Based Education: An Approach Toward Adaptive Learning Procedures," *ADCIS Summer Conference*, 1976, pp. 1-22.
2. Cheney, Paul. "Egoless Programming: As an Instructional Technique," *AEDS 14th Annual Convention*, 1976, pp. 222-224.
3. Ahl, David (editor), *Creative Computing*, Morristown, New Jersey.
4. Egerton, John, "Teaching Learning While Learning to Teach," *Change*, Vol. 8, No. 2, March 1976, pp. 58-61.
5. Gehret, Kenneth G., "Children Make the Best Tutors," *Christian Science Monitor*, January 14, 1974, p. F1.
6. Harman, Willis W., *An Incomplete Guide to the Future*, Stanford Alumni Association, Stanford, California, 1976.
7. Harrington, Pat, " 'Earth Ball' Rolls into America," *Fort Wayne Journal-Gazette*, Fort Wayne, Indiana, February 15, 1976, p. 2D.
8. Hicks, Bruce and Kathy Jaycox, "Elders, Students and Computers: A New Team," ISEAC #7, College of Education, University of Illinois, Urbana, Illinois, 1976.
9. Hicks, Bruce and Zielinski, "SEC-A Simulation of Computer Privacy and Security Problems," ISEAC #15, College of Education, University of Illinois, Urbana, Illinois, 1976.
10. Hicks, Bruce, "Computer Outreach," *Computers and Society*, ACM Special Interest Group, Vol. 7, No. 3, Fall 1976, pp. 10-14.
11. Sherman, J. Gilmour (Editor-in-Chief), *Journal of Personalized Instruction*, Center for Personalized Instruction, Georgetown University, Washington, D.C.
12. Lincoln, C. Eric, "Equalizing the Opportunity to Learn," *Change*, Vol. 8, No. 6, July 1976, pp. 60-63.
13. Modesitt, Ken, "The Tangled Triangle: Cooperation, Computer-Based Education and Personalized Systems of Instruction," *ADCIS Summer Conference*, 1976, pp. 320-324.
14. Modesitt, Ken, "Personalized Systems of Instruction in Computer Science: 'Adult' Education," *National Conference on Personalized Instruction in Higher Education*, 1975
15. Modesitt, Ken, "An Excellent Mixture for PSI: Computer Science, PLATO, and Knowledge Levels," *ACM National Conference*, 1974, pp. 89-94.
16. Modesitt, Ken, "PSI: A Valuable Addition to the Alphabet Soup for Computer Science Education," *ACM Special Interest Group on Computer Science Education Bulletin*, Vol. 6, June, 1974, pp. 37-44.
17. Morris, William (editor), *The American Heritage Dictionary of the English Language*, 1969.
18. North, Robert C., *The World that Could Be*, Stanford Alumni Press, Stanford, California, 1976.
19. *PSI Newsletter*, Center for Personalized Instruction, Georgetown University, Washington, D.C.
20. Rhinelander, Philip H., *Is Man Incomprehensible to Man?*, Stanford Alumni Press, Stanford, California, 1973.
21. Ruskin, Robert S. and Steven F. Bono (editors), *Proceedings of the National Conference on Personalized Instruction in Higher Education*, Georgetown University, Washington, D.C., 1974 to present.
22. Simon, S. B., L. W. Howe, and H. Kirschenbaum, *Values Clarification: A Handbook of Practical Strategies for Teachers and Students*. Hart Publishing Co., New York, 1972.

APPENDIX—DATA BASE DESIGN FOR PERSONAL INFORMATION

Group Project
CS 220
Ken Modesitt

I know everybody's income and what everybody earns; and I carefully compare it with the income-tax returns;

To everybody's prejudice I know a thing or two; I can tell a woman's age in half a minute—and I do!

Yet everybody says I am a disagreeable man! And I can't think why!

—King Gama in Gilbert and Sullivan's
Princess Ida

There is today a vast array of data about you and me stored in computerized data bases. The following list, although very incomplete, contains types of organizations and institutions which can input and access data about you from thousands of computers throughout the nation.

Federal Government
Census Bureau
FBI
IRS
Selective Service
Armed Forces
Social Security Administration
State Department (Passports)

State Government
Police
Department of Motor Vehicles
Department of Revenue

County Government
Tax Assessor
Board of Elections

City Government
Banks and Savings and Loans
Educational
Insurance
Utilities
Employer
Credit Cards
Professional Organizations

Special Interest Groups
Alumni
NRA
John Birch Society
Book Clubs

Record Clubs
League of Women Voters
Audubon Society
etc.

Medical Records (Physicians, hospitals)
Mail-order Houses
Political Parties
Dating and Marriage Services
Retail Stores

The potential good from wise and careful use of such automated personal data systems is considerable. *And* so is the potential harm! I am particularly concerned about the latter. Incidents, some humorous and some serious, abound which relate to computerized personal information systems. Can you think of several?

In response to a growing concern about the role of the individual in such computerized systems, the United States Department of Health, Education, and Welfare published a special report. "Records, Computers, and the Rights of Citizens" (1973) is on reserve in the library. The report recommended the enactment of a Federal "Code of Fair Information Practice." The Code rests on five basic principles that would be given legal effect as "safeguard requirements" for automated personal data systems. The following is taken verbatim from the report (pp. 40-41).

> *"Here then is the nub of the matter. Personal privacy, as it relates to personal-data record keeping must be understood in terms of a concept of mutuality. Accordingly, we offer the following formulation:*
>
> > *An individual's personal privacy is directly affected by the kind of disclosure and use made of identifiable information about him in a record. A record containing information about an individual in identifiable form must, therefore, be governed by procedures that afford the individual a right to participate in deciding what the content of the record will be, and what disclosure and use will be made of the identifiable information in it. Any recording, disclosure, and use of identifiable personal information not governed by such procedures must be prescribed as an unfair information practice unless such recording, disclosure or use is specifically authorized by law.*
>
> *This formulation does not provide the basis for determining a priori which data should or may be recorded and used, or why, and when. It does, however, provide a basis for establishing procedures that assure the individual a right to participate in a meaningful way in decisions about what goes into records about him and how that information shall be used.*
>
> *Safeguards for personal privacy based on our concept of mutuality in record-keeping would require adherence by record-keeping organizations to certain fundamental principles of fair information practice.*
>
> > *There must be no personal-data record-keeping systems whose very existence is secret.*
> >
> > *There must be a way for an individual to find out what information about him is in a record and how it is used.*
> >
> > *There must be a way for an individual to prevent information about him obtained for one purpose from being used or made available for other purposes without his consent.*
> >
> > *There must be a way for an individual to correct or amend a record of identifiable information about him.*
> >
> > *Any organization creating, maintaining, using, or disseminating records of identifiable personal data must assure the reliability of the data for their intended use and must take reasonable precautions to prevent misuse of the data."*

Your cooperative group assignment is to design an automated personal data system for a content area of your choice. Feel free to use those listed earlier. Your design should include the principles just articulated *and* any others you would like.

What features would *you* want in such a system were you a user and/or entry thereof?

Would you want to see your record?

What information can others input about you?

Can you challenge and change your record? How?

Who has looked at your record? When?

Who can erase information?

Can you prevent someone from looking?

How do you know incorrect or outdated information was *really* erased (corrected)?

Is the information used only for that purpose for which it was collected?

How would you find out if a record of you is being kept?

Are there levels of information access?

Can you inspect the records of your mate, friend, children, employer, etc.?

Is certain information purged automatically after a certain period of time?

The above questions represent only a *very few* which could be asked. You will undoubtedly uncover others as your project unfolds. You might ask others what questions they have.

The project is a design one only; it does *not* involve coding (unless you wish to). However, the design should proceed in a top-down manner, expanding various steps where necessary. The final written result of your effort should be a set of structured programs given in increasing detail. The most detailed one should be one able to be coded directly into some programming language.

A typed copy of your design will be presented to class during the last two weeks of class. (I will supply the mimeograph service *if* you give me the typed copy two days before handing it out.) In addition, your presentation for that day of class may take any form you wish, as long as each group member has a role. Your group might role-play a "typical day in the life of a computerized dating service," for example. Or someone might indicate how a user might try to use/access your system, honestly or otherwise. Perhaps you could incorporate members of the class in your presentation. In any case, you are to hand in two questions (and their answers) which could be answered only by a person attending your presentation. Some of these will appear on the final exam.

Good luck and hope you enjoy this "realistic" assignment involving the world and computers and other people!

Ken

P.S.—Another excellent reference is *Privacy Journal*. I have several copies in my office.

New perspectives for information systems education

by THOMAS I. M. HO
Purdue University
West Lafayette, Indiana

ABSTRACT

Systems analysis is the examination of a problem situation in order to define the requirements of a solution, often computerized, to that problem. The diversity of problems and the constraints of computing technology require that a problem be thoroughly analyzed in order to insure that the problem is clearly understood. Then, and only then it can be determined how computing technology can be applied to solve the identified problem. Therefore, systems analysis education teaches the discipline of clear and complete problem definition.

Several recent developments offer new perspectives for systems analysis education. These developments provide a conceptual framework for understanding information system and data base characteristics. This framework supports an improved methodology for systems analysis and thereby contributes to higher quality systems analysis education.

INTRODUCTION

A pair of technological developments in computer science are currently making a significant impact on the development of computerized information systems. Data Base Management Systems (DBMS) support sophisticated information systems with flexible facilities for the maintenance and manipulation of large complex data bases. In particular, a DBMS enables the sharing of data resources in order to co-ordinate the diverse activities of an environment supported by an information system. Furthermore, a DBMS enables the representation of complex logical structures in a data base that models the environment supported by that data base.

Computer aids to information systems development manage the activities of personnel engaged in the various phases spanning a development project from the conception of a need to the installation of the solution fulfilling that need. For example, a computer supported data dictionary manages the definition of data resources in order to provide a complete and consistent view of data for use in a DBMS application.

The impact of these technological developments stems from their contribution to relieving the difficulty of applying computing technology to information systems. The size and complexity of information systems are the major obstacles to their successful development. An information system for a large organization consists of several functional subsystems that represent the various functions performed by the organization. Co-ordination of the subsystems is necessary in order to achieve organizational objectives. Therefore, a DBMS enables the sharing of data resources among subsystems in order to co-ordinate these subsystems. Furthermore, each subsystem is a large problem by itself and so therefore, responsibility for a complete information system must be shared among many individuals. Therefore, computer aids to information systems development enable communication among systems development personnel to aid co-ordination of their activities.

The impact of these technological developments is further magnified by the wide spectrum of applications spanned by computing technology. The breadth of the span of applications justifies both the creation and the further improvement of educational programs that provide instruction in the application of computing technology to information systems.

The computer science curriculum at Purdue University includes an information systems program that provides instruction in the application of computing technology to information systems. This program includes instruction in both systems analysis and systems design techniques. Systems analysis is the problem definition activity that provides a complete and clear perception of the problem for which a computing solution must be determined during systems design. The dissimilarity between the varied problems in the problem domain and the varied technologies in the computing domain presents a unique situation that challenges the capabilities of information systems education.

The dissimilarity between the problem domain and the computing domain suggests the need for a common interface at which the system analysis and systems design activities can meet. Such an interface is a model of an information system which provides a conceptual framework for the statement of requirements of an information system. The statement of requirements represents the problem requirements determined during systems analysis that must be satisfied by the technological solution selected during systems design. The conceptual framework must be consistent with a wide class of problems in the problem domain and with a wide class of technological tools in the computing domain.

The current approach to information systems education suffers from the absence of a conceptual framework for information systems. A narrow view of systems analysis education restricts instruction to a survey of traditional implementations of common application problems. As a result, the student encounters a motley assortment of computerized solutions that are applicable to only specific problems and dependent on specific implementation solutions of those problems. Therefore, *no* general problem-solving approach is apparent to a student whose education is limited by this narrow perspective.

Furthermore, a narrow view of systems design education restricts instruction to a survey of hardware and software technologies. In the absence of a common conceptual framework for information systems, such a technological survey leaves the student with no apparent approach to the selection of technological tools that are isolated from the problem domain. The virtually unlimited variety of the problem domain makes it practically impossible to indicate the technological solution to each problem. In addition, the absence of a conceptual framework makes it difficult to motivate the need for the capabilities of the technological solutions in the computing domain. For example, the need for representation of complex data base structure that is fulfilled by DBMS is not necessarily apparent to anyone who is unaware of the size and complexity of contemporary information systems problems. However, characterization of the role of the data base in the context of both the information system model and the problem domain motivates the need for complex logical structure.

CONCEPTUAL FRAMEWORKS FOR INFORMATION SYSTEMS

Information system model

The study of information systems is complicated by the dissimilarity of the organization system and the computer system. The organization system performs the activities that must be supported by the information system. The computer system performs computational and data management functions. There is *no* correspondence between the organization's activities and the computer's functions. The organization system contains the persons, objects, and events that are the subjects of organizational activities. The computer system contains the hardware and software facilities that perform computerized functions. There is *no* correspondence between the organization's subjects and the computer's facilities.

The gap between the organization and computer systems suggests the need for a conceptual bridge between these two systems. Such a bridge would guide and structure a systems analyst's activities while he formulates his approach to an organizational requirement. Such a bridge would *not* remove the necessity for the analyst to be familiar with the application domain with which he is dealing. Instead, the conceptual link identifies the concepts common to all applications of management information systems in order to supplement specific knowledge of organization and computer systems.

The conceptual link is an information system model[1] that provides a standard that enables organization system concepts to be expressed in a conceptual framework that is also compatible with computer system concepts. The information system model is itself a system composed of interacting subsystems:

1. Input subsystem
2. Output subsystem
3. Data base subsystem
4. Process subsystem.

The role of the information system model is illustrated in Figure 1.

The correspondence to the various subsystems of the computer system is clear and this is no surprise. Correspondence to the elements of the organizational subsystem can be established. The elements of the output subsystem correspond to the actions and decisions performed by each functional subsystem. The elements of the process subsystem correspond to the procedures and models used to perform each action or decision. The elements of the input subsystem correspond to the data received from the environment by the elements of the process subsystem to generate the elements of the output subsystem.

Data base subsystem

The data base subsystem serves as a decoupling mechanism between the input and output subsystems. The input subsystem gathers the data from the environment to be used to generate information to the environment through the output subsystem. However, the output subsystem does not necessarily generate information at the same time nor at the same rate as the input subsystem receives data. Therefore, the data base subsystem is an inventory of data

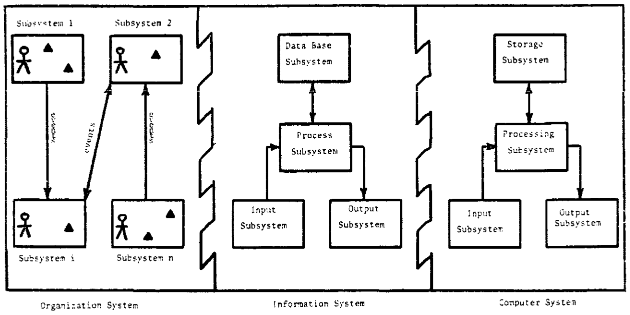


Figure 1—Management information system

resources. Furthermore, the output subsystem does not necessarily request information in a format that is identical with that of the data used to generate the desired information. Hence, the data base subsystem maintains a standard specification for data resources in order to decouple the incompatibilities between the input and output subsystems. The decoupling role of the data base subsystem in these respects motivates the residence of the data base subsystem in the storage subsystem of a computer system.

With respect to the organization system, the data base subsystem also functions as a decoupling mechanism. The various functional subsystems of an organization system are interacting subsystems that must communicate with one another to achieve the desired synergistic effect. Again, the data base subsystem serves as both an inventory and as a standard for the data resources that are generated by any functional subsystem and can be used by any other functional subsystem in pursuit of that subsystem's objectives. Similarly, the data base subsystem also decouples separate procedures and models within a single subsystem. However, it is the data base subsystem's role as a decoupling mechanism between functional subsystems that elevates it to its central role in an integrated information system.

Entity-relationship model of data

In its role as a decoupling mechanism, the data base subsystem should therefore contain representations of the persons, objects, and events of interest to organizational activities. The elements of the data base subsystem that represent these persons, objects, and events are called *entities*. Furthermore, the data base subsystem should also contain representations of the relevant associations among the organizational persons, objects, and events. The elements of the data base subsystem that represent these associations are called *relationships* among the corresponding entities. The concepts of entity and relationship for data base definition have been proposed by both Teichroew,[2] Chen,[3] and ANSI/X3/SPARC Study Group on DBMS.[4]

An *entity* type is a model of a person, object, or event of interest to the organization system. An entity *occurrence* is the representation of an instance of the person, object, or event represented by the corresponding entity type. Therefore, EMPLOYEE may be an entity type while JOHN DOE is an occurrence of EMPLOYEE. An entity type consists of *attributes* that describe the entity. An entity occurrence consists of *facts* that describe the instance being represented. Therefore, if EMPLOYEE consists of the attributes NAME and ADDRESS, JOHN DOE might consist of the facts NAME is JOHN DOE and ADDRESS is 123 MAIN STREET. One or more of the attributes must serve as an *identifier* whose value distinguishes one occurrence of an entity from another occurrence of the same entity.

The scope of an entity is arbitrary. Part of one entity can be separately defined as another entity. For example, an object entity called Product can also be defined in terms of another object entity called Subassembly. Conversely, a *collection* of entities can be separately defined as another entity. For example, the collection of object entities Part and Product can be defined instead as the single object entity Material. Hence, in any organization system, any number of entity types is possible. Some guidelines for the selection of attributes of an entity have been presented by Brown.[5]

An entity type may be associated with some other entity type, *not* necessarily different from the first, by a *relationship* type. For each occurrence of one entity type, a relationship type defined between that first entity type and some other entity type defines a set of occurrences of the second entity type that have a common property (implied by the relationship) with respect to the occurrence of the first entity type. For example, a relationship type defined between the entity types CUSTOMER and ORDER defines the set of ORDER occurrences that were placed by each CUSTOMER occurrence. An important property of a relationship is its *connectivity*. For each occurrence of one entity type, connectivity indicates the maximum size of the set of occurrences of the second entity type that have the common property with respect to the occurrence of the first entity type. For example, the relationship CUSTOMER and ORDER has connectivity 1 to N (>1) because each CUSTOMER may place *more* than one ORDER, but each ORDER is placed by *only* one CUSTOMER.

The relationship concept is essential to the fulfillment of the decoupling role of the data base subsystem. The integration of the various functional subsystems of the organization system is promoted by relationships among the entities that represent the various persons, objects, and events of interest. For example, integration of the activities of the Order Entry, Inventory, and Shipping subsystems motivates the relationships indicated in Figure 2. A rectangle is used to represent an entity and a diamond is used to represent a relationship. The relationships indicate the creation of either SHIPMENT or BACKORDER occurrences to fulfill each ORDER occurrence. In addition, the relationship between BACKORDER and SHIPMENT indicates creation of a SHIPMENT occurrence when each BACKORDER occurrence is fulfilled. With respect to a customer's inquiry concerning any ORDER, the contribution to integration is apparent in the ability to respond with relevant information of either SHIPMENTs or BACKORDERs that fulfill that ORDER. Feedback is also apparent in the ability to inform a customer of the imminent receipt of his unfulfilled ORDER by virtue of the fulfillment of the responsible BACKORDER.

Effective organizational control is promoted by the relationship concept. Control is possible only if there exists a sensor mechanism to detect a system state that is at variance with some designated system standard. The sensor mechanism is enabled by the data base representation of a relationship that enables ready detection of the variance condition. As illustrated in Figure 2, a relationship type between BACKORDER and PRODUCT enables easy detection of the variance condition exhibited by excessive backorders for any particular product.

The relationship concept also restores the loss of structure that is apparent when the scope of an entity is

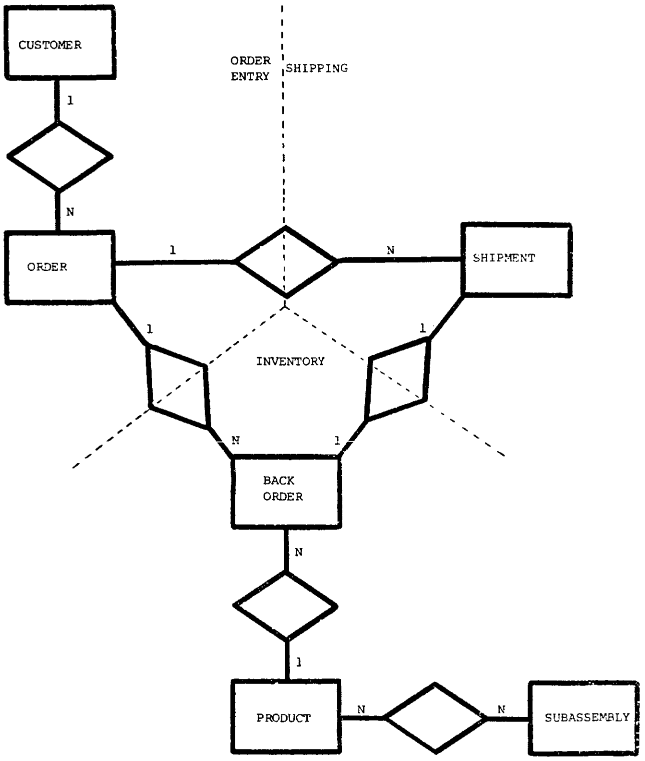


Figure 2

narrowed. When part of one entity is separately defined as another entity, the original data structure can be preserved by defining a relationship between the two entities. As illustrated in Figure 2, when a PRODUCT entity is defined in terms of a SUBASSEMBLY entity, structure can be preserved by a relationship type that defines the set of SUBASSEMBLY occurrences that compose each PRODUCT occurrence.

Finally, the relationship concept promotes data nonredundancy and its recognized contribution to data consistency and storage savings. Figure 2 includes a relationship type between CUSTOMER and ORDER that avoids redundant representation of CUSTOMER data in multiple ORDER occurrences placed by the same CUSTOMER.

CONCEPTS FOR INFORMATION SYSTEMS EDUCATION

Requirements statement language

An information system model suitable as a conceptual framework for information systems education is provided by a Requirements Statement Language (RSL). An RSL[1] is a high-level language for describing information system requirements that are determined during systems analysis and fulfilled during systems design. An RSL is *not* a programming language since an RSL statement expresses what requirements must be fulfilled rather than how those requirements are implemented in a hardware and software solution.

The most advanced RSL is the Problem Statement Language (PSL) developed by the Information Systems Design and Optimization System (ISDOS) Project.[6] PSL facilities for statement of data requirements conform to the entity-relationship model of data.[3] As described by Ho,[7] the entity-relationship model instills a data management perspective that exemplifies the role of the data base in the information system. To complement this systems analysis perspective on data management, a systems design perspective on data management is provided by techniques for the inference of a data base schema from a PSL statement of data requirements.[8]

In the case of statement of data manipulation requirements, PSL does not provide the desired facilities. PSL facilities do not provide the capability for stating detailed requirements for data manipulation and processing of data elements. The Accurately Defined Systems (ADS) technique[9] provides a practical method for describing system flow at the data element level. ADS describes the composition of the output, input, process, and data base subsystems of the information system model. Then, ADS describes system flow by specifying the source of each data element occurrence in the output, process, and data base subsystems. The source of a data element is an input, process, or data base occurrence of the same data element. However, ADS does not provide any facility for the statement of data manipulation requirements that might be fulfilled by a DBMS.

Requirements statement analyzer

The effective use of an RSL is supported by a software package known as a Requirements Statement Analyzer (RSA). An RSA performs logical checks on an RSL statement for compliance with completeness and consistency conditions defined in terms of the information system model.[10] The relevance to information systems education is evident in the RSA contribution to improving the quality of systems analysis and design. System design is entirely dependent on the completeness and consistency of the requirements determined during systems analysis. In addition, an RSA also displays the system requirements in various tabular and graphical formats that provide a system perspective of inestimable value to systems analysis and design.

PSL is supported by the Problem Statement Analyzer (PSA), an RSA that provides extensive capabilities for maintaining and displaying system requirements expressed in a PSL statement. The use of ADS as an RSL is vitally supported by PSA/ADS, an RSA for ADS reported by Nunamaker, Ho, Konsynski, and Singer.[11]

Requirements statement tools in information systems education

Currently at Purdue, undergraduate instructional activity in systems analysis is supported by the use of ADS as an RSL. Students learn the systems analysis task by participating in a term project that produces a requirements statement for an information system described in a case study. The case study problem includes a diverse collection of interacting organizational activities in order to create a problem situation of sufficient size and complexity. These characteristics of the problem situation create the need for data sharing and project management that motivates the use of DBMS and computer aids for information systems development.

However, ADS has proven to be inadequate for statement of the data definition and manipulation requirements for data management that can be fulfilled by a DBMS. Furthermore, use of PSA/ADS for requirements statement analysis is less than satisfactory due to the limitations of its primitive implementation. All these shortcomings do not exist in PSA/PSL with one major exception. PSL does not provide facilities for stating detailed requirements for data manipulation as might be performed by a DBMS. In addition, PSL does not allow specification of the source of each data element occurrence as provided by ADS. Our experience in systems analysis education has demonstrated that the concept of data sources is pedagogically essential to understanding the flow of data in an information system. In all other respects, PSA/PSL has proven to be an excellent tool for systems analysis education. Our experience with use of PSA/PSL in graduate systems analysis education here at Purdue has been extremely favorable.

In addition to the more obvious contribution to a conceptual framework for information systems education, computer aids also contribute to the productivity gain that results from the detection of systems analysis errors and the automated maintenance of requirements statements. Productivity is especially vital in the educational environment which is expected to provide relevant student experiences in courses of relatively short duration. We therefore confront students with problems of realistic size and complexity that require high productivity for their successful completion in the short span of a course.

CONTRIBUTION TO INFORMATION SYSTEMS EDUCATION

The greatest impact of the products described herein will be higher quality systems analysis education. The capability for completeness and consistency checking afforded by the RSA provides feedback on student performance and better systems analysis technique as a result. The structure inherent in RSL statement re-inforces the concept of structured programming and other progressive software development disciplines. The productivity gain accomplished by the use of automated techniques encourages creativity by relieving the user of the burden of manual systems analysis techniques. The exposure to state-of-the-art techniques expands student perspectives beyond the traditionally confined horizons of traditional data processing. In summary, the impact is most evident in the improved ability to apply computing technology to the wide and varied spectrum of problems that could benefit from data management relevant to their decision-making and other activities.

REFERENCES

1. Ho, T. I. M., "Systems Analysis Perspectives," *Proc. 14th Annual Conference on Computer Personnel Research,* July 1976.
2. Teichroew, D., *et al., An Introduction to PSL/PSA,* University of Michigan, Department of Industrial and Operations Engineering, ISDOS Working Paper No. 86, March 1974.
3. Chen, P. P. S., "The Entity-Relationship Model: Toward a Unified View of Data," *Trans. Database Systems,* Vol. 1, No. 1, March 1976, pp. 9-36.
4. ANSI/X3/SPARC Study Group on DBMS, *Interim Report,* Doc. No. 7514TS01, CBEMA, Washington, DC, February 1975.
5. Brown, A. P. G., "Modelling a Real World System and Designing a Schema to Represent It," *Data Base Description,* Douque, B. C. M. and G. M. Nijssen (eds.), North-Holland Publishing, 1975, pp. 339-347.
6. Teichroew, D. and H. Sayani, "Automation of System Building," *Datamation* Vol. 17, No. 16, August 15, 1971, pp. 25-30.
7. Ho, T. I. M., *Data Base Concepts for Systems Analysis,* Purdue University, Computer Sciences Department Technical Report, November 1976.
8. Blosser, P. A., *An Automatic System for Application Software Generation and Portability,* Ph.D. dissertation, Purdue University, May 1976.
9. Lynch, H. J., "ADS: A Technique in System Documentation," *Data Base* Vol. 1, No. 1, Spring 1969, pp. 6-18.
10. Ho, T. I. M. and J. F. Nunamaker, Jr., "Requirements Statement Language Principles for Automatic Programming," *Proc. 1974 ACM National Conference,* November 1974, pp. 279-288.
11. Nunamaker, J. F., Jr., T. I. M. Ho, B. Konsynski, and C. Singer, "Computer-Aided Analysis of Information Systems," *Comm. ACM,* to appear during Winter 1977.

Petroleum data system—A network of energy information

by PATRICIA A. TRACY
University of Oklahoma
Norman, Oklahoma

ABSTRACT

The Petroleum Data System is a comprehensive computer base of petroleum information. PDS is available worldwide through the General Electric time-sharing network. It has received widespread usage since its release in April, 1976.

The software supporting PDS is a group of general purpose software packages. A driver program links the packages together and provides the interface between the packages and the time-sharing network. These packages handle the problems of information storage and retrieval, applications, and graphics. They have proved to be adaptable to a wide variety of applications. In addition to PDS, the University of Oklahoma plans to make available data bases on coal, geothermal, and minerals. These data bases will be accessed via the same operating procedure as PDS, creating a complete energy-information network.

The Petroleum Data System consists of information on over 75,000 oil and gas fields in the United States and Canada. The Petroleum Data System or PDS, as the group of data bases is called, is a collection of publicly available information on all oil and gas fields and reservoirs. These data bases were developed over the past seven years by the University of Oklahoma under contract to the United States Geological Survey.

Information from hundreds of published reports have been incorporated into PDS, creating a library of computerized energy data. State annual reports are the primary source of data. Supplementary sources include the International Oil Scouts Association and state geologists. Federal agencies such as the Federal Energy Administration, Federal Power Commission, Energy Research and Development Administration, and Bureau of Mines are major contributors of data.

These files provide locations, geologic, engineering and production information. They are being used to analyze trends and risks, and perform economic evaluations and enhanced recovery studies.

The University has chosen to use several existing software systems to help it accomplish its goal of a complete energy network. These systems are generalized systems, which have proved themselves adaptable to a wide variety of applications. The use of existing software has freed the University of many problems in program design and implementation. It has also resulted in lower implementation costs and lower operating costs.

PDS uses a highly sophisticated storage and retrieval system called GIPSY. GIPSY was developed at the University of Oklahoma and has been operational since July, 1968. GIPSY is an acronym for General Information Processing System. The computer environment used by GIPSY is a mainframe of IBM 360 or 370 series operating under IBM OS. It is an excellent system for handling the diverse types of data in PDS, i.e., numbers, codes, and alphanumeric character strings, both fixed length and variable length.

GIPSY has certain characteristics which are prerequisite to the management of any large data base system. As I mentioned before, the Petroleum Data System has over 75,000 records. A single record may have as many as 600 different items of information. Records vary considerably in the degree of completeness. A typical record may have as few as 25 items of information or up to approximately 300 items of information. GIPSY uses economical storage techniques, compressing all empty spaces from the file. GIPSY takes advantage of binary searching techniques to locate data and to decrease costs and computer time.

GIPSY also has the flexibility necessary to handle the constantly changing facade of the PDS data. A set of utility programs handle standard file maintenance procedures, such as adding records to files, deleting records, creating backups, restoring files, and updating files. The update utility allows for the addition or replacement of data to PDS. It also allows for the deletion and for the extension of existing information such as a comments fields or a source document.

PDS is continually being updated. New discoveries are added to the file, and new sources of information are continually researched to fill in missing data. Frequently it is necessary to "restructure" the file by defining new data items, or by deleting items which were defined and have since been determined unnecessary. For example, we are currently adding the latitude and longitude of the centerpoints of the fields to the records in PDS. These items were not defined when the files were initially created. GIPSY handles these new data items with ease. All that is necessary is to add a new data definition to the file dictionary. No special programming is required.

GIPSY is a complete retrieval system. It allows the user to isolate from the mass of information that part which satisfies his immediate requirements. GIPSY has its own retrieval language. It is a simple command language which can be learned with a minimal degree of effort.

Two commands are available for searching data bases and retrieving selected subsets of records.

SELECT will peruse an entire record file or the index of the record file checking records for conditions which the user specifies. As many conditions as necessary may be specified. Conditions can be specified which simply check records for the existence of a particular data item. Numeric items can be checked to determine whether they are less than, greater than, or equal to a given number, or to determine if they fall within a given numeric range. An alphanumeric item can be checked for a specific character string, or for strings which fall within a given character range. Boolean logic (and, or, not) is used to describe how the conditions should be met in the records.

ITERATE will narrow a subset of records down to a smaller, more specific set of records.

A command called SORT will order records in whatever sequence is desired.

Using a command called PRINT, records in the system can be printed in their entirety in a predefined format. The command LIST can be used to print only selected information from the subset of records.

Ths SUM command will generate sums, averages, maximums, and minimums of specific items in a subset. The COUNT command generates one way frequency distributions.

GIPSY also provides a method of interfacing the data with other processing systems. The COPY command will output data in a fixed format defined by the user to tape or disk. The data can then be manipulated in any manner, either with special user-written programs, or interfaced with statistical packages, plotting routines or report-writers.

GIPSY gives the Petroleum Data System an economical, flexible, user-oriented retrieval mechanism.

The Petroleum Data System has been available to the public since August, 1975. Initially, the University attempted to make the file available through its own in-house computer time-sharing system. However, the response from the industry was so overwhelming that the University was unable to fulfill the needs of its customers.

In an effort to make the data more readily available to users of the Petroleum Data System, the University of Oklahoma has put the system on the General Electric time-sharing network. A validated customer may access the data via a local telephone call from any major city in the United States and cities in 20 foreign countries. The Petroleum Data System is now accessible to any group, large or small, government or private industry, on any scale they wish to operate, from a country-wide study to a small area such as a county or field.

The Petroleum Data System resides on the General Electric remote batch system. This provides a significant cost savings as opposed to the cost of trying to keep such a large volume of data available in an on-line interactive file.

Among the PDS users there is a large variation in user expertise. Typical users include geologists, consultants, programmers, systems analysts, and geological secretaries and librarians. To simplify the use of the remote batch computer, an interface was developed between the foreground interactive time-sharing service and the remote batch service. This interface, called a foreground driver, sets up the IBM Job Control Language and the GIPSY batch retrieval commands and automatically submits them to the remote batch computer. The driver program also monitors the job status, retrieves reports, and gives job statistics.

In a typical terminal session, the user submits a job to select a subset of records and to output certain information from those records. Generally, the user may log back on the terminal within 30 minutes to an hour after the job has been submitted to retrieve the results of the job. Printed reports may be listed totally or in part at the terminal. Large reports can be printed on the customer's in-house high-speed printer, or printed at General Electric and couriered to the customer's office. General Electric also provides many statistical packages and report generating programs which can be used for data manipulation.

PDS has been linked to an applications package developed by Amoco Production Company. The Applications Management System provides data processing, file interface, and display capabilities.

AMS is composed of many separate modules. These modules perform specific functions, which are data independent and often have general application. This allows the AMS user to develop custom solutions to a variety of problems, depending on the manner in which the modules are blended. Models have been developed for PDS which will generate scatterplots, histograms, regression analyses, trend analyses, and write reports.

Scatterplot, for example, will plot any two numeric items from any data base in PDS. The user responds to a series of questions asking for such information as the x,y coordinates and size of plots.

Sophisticated PDS/AMS users have the ability to develop models tailored to more specific needs. AMS provides a uniform syntax to aid the user in communicating with the data structure and the computer, thus simplifying and streamlining the computer process.

The next stage of development will be to incorporate a graphics package into the Petroleum Data System. This will give users the ability to plot locations and assign symbolic values to the graphics data.

Many other energy resource related data bases are being developed including coal, geothermal, and minerals. It is the plan of the University to make all of these data bases available on the General Electric time-sharing network. While the data bases differ substantially in file content and structure, the customer will be able to use the same operating procedure to access the data.

The result of all this has been a highly successful energy information network. Using developed software has been a distinct advantage. It has given us a wide range of flexibility by allowing us to develop applications as the demand arises. By molding these packages to meet the requirements of the Petroleum Data System and the many other energy related data bases, we have been able to provide our users with a large variety of output options.

Much of the success of the system can be attributed to the energy crisis and to an awakened realization by the industry for the need to immediate access to a variety of energy related data. It is our goal to encourage industry and government agencies to cooperate in the exchange of information which might be used in the conservation and development of our resources.

Applications of SPARCOM data base concepts to a crime combating environment*

by RON ASHANY

Thomas J. Watson Research Center
Yorktown Heights, New York

ABSTRACT

In this paper only certain aspects of a powerful Data Base System called SPARCOM, as reflected by some applications in a crime-combating environment are presented. SPARCOM stands for Sparse Associative Relational Connection Matrix. It is a method that was developed for the analysis, interpretation, organization, classification, update, and structure of stored data as well as for the search and retrieval of information from large data base (LDB) systems. The unique approach of this system is the conversion of data into large sparse binary matrices that enables one to apply sophisticated sparse matrix techniques to perform data base operations. The operations are performed on the matrices as though the entire matrix were present, but great amounts of storage space are saved, and execution time is significantly reduced by the storage and manipulation of the nonzero values only. Additional reduction in storage requirements and in execution time is achieved by SPARCOM's intrinsic normalization process that alleviates the grave problem of data redundancy, caused by multi-value attributes.

INTRODUCTION

At the current state-of-the-art, Data Base systems are still largely the result of ingenious casual and often belated attention to human factors, users' needs, system configurations and programming problems. The methods used in the development and design of such systems are essentially trial-and-error. No evidence is to be found in the technical literature, describing existing systems, to indicate that the development was based on a sound scientific foundation or on well established engineering disciplines. Generally speaking, no unifying concept, general theory, or common systematic approach is as yet evident in work in the field of data base technology. One of the major contributing factors to the haphazard approach is the lack of distinction, during the development phases, between physical and logical considerations.

Before contemplating the development and design of any system, an appropriate conceptual framework must be established. To understand and communicate the concepts of such a framework certain definitions must be postulated. Some of the definitions are precise and have been established on rigorous mathematical foundations, others are of a generic nature and still others have been established to provide a common base of reference. In the new field of data base technology, there is no evidence of an existing commonly agreed nomenclature. Many terms, introduced in this paper are new, others were adopted from different sources.

BASIC CONCEPTS AND DEFINITIONS

The digital computer cannot deal with physical objects, optical images, events or abstract concepts directly. A data model of the physical world or of the abstract concepts which can be easily manipulated must be constructed first. The *Data Model* (DM) is the information content of the data base as it is viewed by the users. For example, a Regular Data Organization (RDO) in an array form as illustrated by the criminal file in Figure 1, is an accepted form of representing data; therefore, it is a data model.

A collection of processable data from which an enterprise can derive information is called a *Data Base*. A police department, a hospital, a court of justice, a university, a bank or any large-scale administrative, scientific, technical, commercial or other type of operation is called an *Enterprise*. An item of interest to an enterprise about which data are recorded or computed is called an *Entity*.

In any given enterprise, there may exist different types of entities, a criminal, a detective, a bullet's signature, a car, a firearm, a crime, a portrait, a report, etc. Entities are identified via their attributes. An *Attribute* is a specific class of information about an entity; each entity has at least one attribute. Because of their wide and frequent usage, a number of attributes have been given names such as: Age,

* This work represents applications of some concepts, investigated as part of the research, reported in the dissertation by R. Ashany, *SPARCOM: A Sparse Matrix Associative Relational Approach to Dynamic Data Structuring and Data Retrieval*, submitted to the Faculty of the Polytechnic Institute of New York in partial fulfillment of the requirements for the degree of Doctor of Philosophy (Electrical Engineering) June 1976.

ID	NAME	CRIME	ARM	LOCATION*	SEX	EYES	LANGUAGE
1	John	Murder	A	Detroit	M	Blue	English
2	Claude	Rape Arm. Rob.	B	N.Y.C.	M	Green	English French German
3	Robert	Hijack	D	Houston	M	Hazel	English German
4	Nancy	Homicide	A	Chicago	F	Blue	English German
5	Ingmar	Rape. Murder	D	Detroit	M	Blue	English Russian Swedish
6	Roberto	Murder	A	Chicago	M	Black	English Italian Spanish
7	Marcel	Kidnap. Hijack	C	N.Y.C.	M	Green	French German Spanish
8	Johanna	Drug Push Arm. Rob.	E	L.A.	F	Green	English French Spanish
9	Jurgen	Kidnap. Homicide	B	Detroit	M	Blue	English German Russian
10	Tom	Drug Push Arm. Rob.	C	L.A.	M	Hazel	English French Swedish

*(Crime Location)

Figure 1—Sample of regular data organization from a criminal file

Salary, Color, Sex, Weight, etc. There is a clear distinction between the *Attribute Name* and *Attribute Value*. An entity named *car* has an attribute named *color*, the value of the attribute is *blue*. The ordered pair Attribute-Value is called a *Property*. A collection of entities described by similar attributes is called an *Entity-Set*. If an attribute assumes a unique value for each entity in an entity set, it is called an *Identifying-Attribute*. An entity may have several identifying attributes; e.g., Full Name, Social Security Number, Employee Number, Prisoner Number, etc. The values of identifying attributes are called *Identifiers*. Attributes whose values can be calculated are called *Virtual Attributes*, such as Age; subtracting from the current year the year of birth the age value is determined.

In the regular data organization of the criminal file, illustrated in Figure 1, each row in the array belongs to a specific entity, $E_1,E_2,...,E_m$. Each column belongs to a specific attribute; $A_1,A_2,...,A_n$. Each element of the array is associated with a specific entity E_i and a specific attribute A_j and represents a specific value V_{ij} selected from a finite set of values pertaining to attribute A_j. The set of values pertaining to attribute A_j is called the Domain D_j. The domain of attribute sex contains only two elements (M,F); the domain of attribute eyes (meaning eye color) contains five elements (black, blue, brown, green, hazel); the domain of attribute language is much larger; it contains all languages spoken in the world. A distinction should be made between single-value attributes and multi-value attributes. An entity can possess only one element from the domain of a *single-value* attribute, but it can possess one or more elements from the domain of a *multi-value* attribute. Sex and eyes are single-value attributes; language is a multi-value attribute. The multi-value attributes cause serious problems in some existing approaches for searching and retrieving data[1-3] but, they cause no problem whatsoever in SPARCOM.

The ordered triple (E_i,A_j,V_{ij}) represents an *elementary data item* which is the minimum amount of data required for meaningful information. The ordered pair (A_j,V_{ij}) represents the property P_{ij}. Consequently, the ordered pair (E_i,P_{ij}) represents an elementary data item. Let's illustrate the meaning of the minimum amount of data required for meaningful information; the entity John by itself does not represent meaningful information, the property Eyes Blue does not represent by itself meaningful information, but the triple John Eyes Blue is meaningful and precise. It is an elementary data item. The elementary data item (E_i,A_j,V_{ij}) is also known as the *Atom* of information because it cannot be further decomposed without losing its message.

MULTI-DIMENSIONAL ATTRIBUTE-SPACE MODEL

An entity can be represented as a point in a multi-dimensional Euclidean space by a vector. The six-tuple E_1(John,23,5.10,180,M,Blue) is a vector representing a point in a six-dimensional space and it can be interpreted as a person possessing the properties Name-John, Age-23, Height-5′10″, Weight-180, Sex-M, Eyes-Color-Blue.

The notions of point, line, and plane from the familiar three-dimensional space may be generalized to higher dimensional spaces, and extended geometric interpretations of algebraic relationships may be given. Starting with a three-dimensional space, an entity set consisting of "m" entities can be envisioned as a set of points with corresponding coordinates described by a set of ordered triples $E_i(V_{i1},V_{i2},V_{i3})$ where $i=1,2,\ldots,m$. The values V_{i1},V_{i2},V_{i3}, are selected from the respective domains of attributes A_1,A_2,A_3. For example, the attributes Age, Height, Weight can represent the respective orthogonal axes A, H, W. The first octant of the space is bounded by the three orthogonal planes; Age-Height, Age-Weight; and Height-Weight or AH, AW, HW, respectively. The triples

$$E_O(0,0,0);\ E_A(1,0,0);\ E_H(0,1,0);\ E_W(0,0,1)$$

represent the *Origin* and *Unit* points, respectively. To represent the aforementioned properties of the first four entities from a specific set, the following four triples are needed:

$$E_1(23,5.10,180);\ E_2(35,5.11,170);$$
$$E_3(27,6.02,210);\ E_4(19,5.04,130)$$

Each tuple represents a vector from the origin to a specific point in space. Each point can be projected on any of the three orthogonal planes. The 3-tuples

$$E_1(23,5.10,0);\ E_1(23,0,180);\ E_1(0,5.10,180)$$

represent the *projection* of the first point on the AH, AW, and HW planes, respectively. They also represent the *projection* of any points that may be found along the respective perpendicular lines through the points of projection. Clearly, any entity with properties Age 23 and Height 5′10″ regardless of its Weight will be projected on the same point on the AH plane, the discussion is similar for the other planes. Much in the same manner the tuples

$$E_1(23,0,0);\ E_1(0,5.10,0);\ E_1(0,0,180)$$

represent the projection of the first point on the A, H, and W axes, respectively. They also represent the projection of any points that may be found on the respective planes that are perpendicular to the respective axes and intersect them at the points of projection. Actually, the triple $E_1(0,0,180)$ represents a plane parallel to the AH plane (perpendicular to the W axis) and intersecting the W axis at 180. The triple $E_1(23,0,180)$ represents a line parallel to the H axis (perpendicular to the AW plane) and intersecting the AW plane at the point (23,180).

QUERIES AND SEARCHING HYPERPLANES

Following the same line of reasoning, it may be said that in a three-dimensional space, a 3-tuple $Q(V_1,V_2,V_3)$ can be interpreted as a *Query* (an interrogation, a question) addressed to the data model. The request of the query is to search all entities represented by the points that coincide with the point specified by the coordinates of the query Vector Q. This is defined as the *Searching Point Query*. Each of the 3-tuples $Q(0,V_2,V_3)$; $Q(V_1,0,V_3)$; $Q(V_1,V_2,0)$ represents a query vector that requests the search of all entities represented by the points that their *projections* have the coordinates (V_2,V_3); (V_1,V_3); (V_1,V_2) on the orthogonal planes.

Obviously, in the multi-dimensional Euclidean space, the entities and queries are represented by *ordered* N-tuples. The five tuple $Q(V_1,V_2,V_3,V_4,V_5)$ represents a searching point query in a five dimensional attribute space. The tuples $Q(V_1,V_2,0,V_4,V_5)$ and $Q(V_1,0,V_3,0,V_4)$ represent searching lines and searching plane queries, respectively, while the tuples $Q(V_1,0,V_2,0,0)$ and $Q(0,V_2,0,0,0)$ represent 3-flat and 4-flat *Searching Hyperplane* queries respectively.

From this discussion, it should be obvious that the search is based on the content of the entities, rather than on their addresses, and that the entities that qualify are determined by the association that exists between the properties contained by the entities and those specified by the queries.

The query vectors must have the same dimensionality as the entity vectors searched by them. A collection of ℓ query vectors is called a *Query Set* and is represented by an $\ell \times n$ *Query Matrix. Query Complexity Degree* (QCD) is defined as the number of non-zero elements specified in the query vector. The QCD should not be confused with the *Request Complexity Degree* (RCD) which is determined by the type and number of properties, operators and operations required to get a complete response. The RCD will be defined in another section when the meaning of type of operators and type of operations will be better understood. To illustrate the meaning of QCD; the entity set defined by the attributes, Age; Height; Weight is interrogated by the following three requests:

Q_1(21,0,0)—Identify all criminals of age 21.

Q_2(35,5.11,0)—Identify all criminals of age 35 *AND* Height 5.11.

Q_3(27,6.02,210)—Identify all criminals of age 27 *AND* Height 6.02 *AND* Weight 210.

Q_1 is a *simple query* QCD=1 and it requires the search of only one domain.

Q_2 is a complex query QCD=2 and it requires the search of two domains, retrieving two subsets and retrieving the subset that results from the *intersection* of the previously retrieved two subsets.

Q_3 is a complex query QCD=3, the same as for Q_2, but with three domains.

If the *request* is to identify all criminals of age 21, *and* retrieve the names of all criminals who live in the same cities as those of the identified criminals, *and* speak Spanish, the request indeed starts with a simple query over one domain, but, imbedded in the request are complex queries that require the search of additional entity sets over different domains; obviously, the RCD should reflect the necessary interactions.

Another type of query known by the name of *Range Query* specifies a range of values over a domain rather than one specific value. For example, the request to identify all criminals between the age of 21 and 24, when the age is a discrete type attribute with a domain of positive integer numbers in increments of one year, the query has an QCD=4, because in this DM, the request has to be formulated as "identify all criminals of age 21 *OR* 22 *OR* 23 *OR* 24". In this data model, if there is a number r of *OR*s in the request, what appears to be one query has actually to be split into $q=(r+1)$ queries. The four 3-tuples representing the above mentioned request are

$$Q_1(21,0,0);\ Q_2(22,0,0);\ Q_3(23,0,0);\ Q_4(24,0,0)$$

and four subsets representing the points found on the four parallel planes will be retrieved. The subset obtained by the *union* of the retrieved subsets will represent the requested answer.

A similar situation will occur when the request deals with a *multi-value* attribute. For example, assuming that the entity set belongs to a 4-D (four-dimensional) attribute-space, and the request is to identify all criminals who speak English *AND* French, the query will have to be split into two 4-tuples

$$Q_1(0,0,0,\text{English}) \text{ and } Q_2(0,0,0,\text{French})$$

but the subset representing the result will be obtained by the *intersection* of the two subsets.

It is to be emphasized that the *AND* operators as opposed to the *OR* operators do not require the splitting of the query except when a multi-value attribute is involved. The request to identify all criminals who are males *AND* have blue eyes *AND* speak English *AND* German addressed to a 3-D entity set will be satisfied by splitting the query into two 3-tuples; i.e., Q_1(M,Blue,English) and Q_2(M,Blue,German). The split is caused by the *AND* of the multi-value attribute and the answer is obtained by the intersection of the two respective subsets.

SEARCH DIFFICULTIES IN THE RDO MODEL

In the RDO model where numeric and alphanumeric values appear in the same array it is very difficult to perform a search simply by comparing the corresponding elements of the entity vectors and query vectors. The search process can be greatly facilitated by encoding all alphanumeric values into pure numeric values. For example, the values of the crime attribute can be encoded so that Murder = 1, Rape = 2, Homicide = 3, and so on; the values of the eyes-color attribute can be encoded so that Black=1, Blue=2, Brown=3, Green=4, and in the same manner for all attributes with alphanumeric values.

Assume that each entity and each query, represented by numeric vectors only, are structured into an $m \times n$ matrix called E and into an $\ell \times n$ matrix called Q, respectively, and that the n attributes are *single-value* attributes *only*. To perform a *search* becomes a simple task. It requires determining whether the number of identical elements, when comparing corresponding elements of an entity vector and a query vector, is equal to the number of non-zero elements specified in the query vector. *If the answer is positive,* it indicates that the respective entity is represented by a point found on the searching hyperplane. Algebraically, if $E=[e_{ij}]$ and $Q=[e_{kj}]$ are the two matrices, and denoting the comparison of identical elements $(e_{ij}=e_{kj})=1$ and non-identical elements $(e_{ij}\neq e_{kj})=0$, the above condition is met if and only if (iff)

$$\sum_{j=1}^{n} (e_{ij}=q_{kj})=t_k \quad \text{where } t_k=n-z_k \text{ and } t_k\leq n$$

z_k being the total number of zero elements specified in the k query vector.

The total search process can therefore be performed by a *Generalized Inner Product* (GIP) between matrices E and Q. The GIP permits the use of any *dyadic* operator between corresponding elements of two matrices where the only necessary condition is, as usual, that the two matrices must be *conformable* (the number of columns of the left matrix must be equal to the number of rows of the right matrix). Two among many acceptable GIP forms (in the APL language) are:

$$A+.\times B;\ \text{and}\ A+.=B$$

The first form is the known matrix product, the last form is the one used to perform the search described above.

By "post-multiplying" (using the GIP+.=) the entity matrix E by the transpose of the query matrix Q a response matrix R of $m\times\ell$ dimensions is obtained. Denoting by Q^T the transpose of Q

$$R=E+.=Q^T \quad r_{ik}=\sum_{j=1}^{n}(e_{ij}=q_{jk}) \ \text{ and } r_{ik}\leq t_k$$

where $i=1,2,\ldots,m$ $j=1,2,\ldots,n$ $k=1,2,\ldots,\ell$. The point representing entity i is found on the searching hyperplane defined by query vector k iff $r_{ik}=t_k$. Clearly, each column of the R matrix contains the answer to the respective ℓ queries. If $r_{ik}<t_k$ the indication is that entity i has only r_{ik} identical properties with those specified in the query vector k. According to the user's needs entities can be retrieved for any r_{ik} value.

MULTI-DIMENSIONAL PROPERTY-SPACE MODEL

To illustrate the strength of the model developed and described in this section, a specific case must be analyzed first. In the section that describes queries and searching hyperplanes in the RDO model, the Range Query was discussed. This is one type of query that is most frequently used in searching criminal files. A witness may describe a criminal in the following terms: Age between 18 to 24, Height between 5.07 to 5.11, Weight between 170 to 190, Male, Caucasian. The ranges specified for the Age, Height and Weight attributes indicate that there is a certain *Degree of Uncertainty* (DOU). Assume that age is stored in increments of one year, the height in increments of one inch and the weight in increments of 10 lbs., and the DOU for each attribute is defined as the number of nonzero values within the respective specified ranges *minus* one. (The minus one comes to indicate that at least one value has to be always specified.)

In the given example, the DOUs are 6, 4 and 2, respectively. As explained previously, each range causes the query to be split. In this particular case, the query has a QCD=7+5+3=15; theoretically, at least 15 queries are needed to retrieve the set of criminals that qualify. The following process has to be invoked: (1) Structuring a 5-D subschema of attributes Age, Height, Weight, Sex, Race; (2) Formulating three 5-tuples (0,0,170,M,Cauc.) (0,0, 180,M,Cauc.) (0,0,190,M,Cauc.); (3) Retrieving a subset of entities obtained from the union of the three previously retrieved subsets; (4) Searching the retrieved subset by five 5-tuples of the form $(0,V_2,0,0,0)$ where $V_2=5.07, 5.08,\ldots,5.11$; (5) Retrieving a subset of entities obtained from the union of the five previously retrieved subsets; (6) Searching the last retrieved subset by seven 5-tuples of the form $(V_1,0,0,0,0)$ where $V_1=18,19,\ldots,24$; (7) Obtaining the subset of entities that represent the answer, from the union of the seven previously retrieved subsets.

The same result can be obtained by formulating 105 5-tuples queries ($g=7\times5\times3=105$) of the form (V_1,V_2,V_3,M, Cauc.) obtained from the Cartesian product $V_1\times V_2\times V_3$ where $V_1\{18,19,\ldots,24\}$; $V_2=\{5.07,5.08,\ldots,5.11\}$ and $V_3=\{170,180,190\}$. The final subset that represents the answer is obtained from the union of the 105 previously retrieved subsets.

Despite the use of computers where the union, intersection and all other operations can be easily performed on large number of sets, the approach just described for handling one request appears to be too complicated, more so when many similar requests have to be satisfied over a short period of time. The solution to this problem, to problems previously described and to additional stringent requirements is found in the multi-dimensional property-space model, represented by the extended binary vectors and the binary connection matrices.

THE BINARY CONNECTION MATRIX

An *n-D* attribute-space containing $A_1,A_2,\ldots,A_n$ attributes, with their domains of distinct elements $D_1,D_2,\ldots,D_n$ of cardinality $d_1,d_2,\ldots,d_n$, respectively, can be transformed into an N-dimensional property space where $N=\Sigma_{i=1}^{n} d_i$ represents the number of distinct properties $P_1,P_2,\ldots,P_N$ necessary to map any point from the *n-D* attribute-space into the *N-D* property-space. Obviously N is larger than n, and to represent a point in a multidimensional Euclidean space requires many more coordinate axes, thus larger vectors. In the property space, however, only two distinct points exist on each axis; *zero* and *one*, and each coordinate axis represents a specific property.

An entity that is described in the attribute-space by n single-value attributes; i.e., by an n-tuple, is described in the property space by a binary N-tuple with *n one* values and *N-n* zero values. The *one* values are inserted in the positions representing the applicable properties. Since the cardinal number of attribute Sex is two (M,F) and the cardinal number of attribute Eyes-color is five (Black,Blue,Brown,Green,Hazel) the entity with properties (M,Blue); i.e., Sex-M and Eyes-Blue is represented by the 7-tuple $E(1,0,0,1,0,0,0)$. The 2-tuple is transformed into a 7-tuple with *two* nonzero elements, and if the single value attributes would have respective cardinal numbers, say, $d_1=10$ and $d_2=12$ the 2-tuple from the attribute-space would be transformed into a 22-tuple in the property-space, again a binary vector with *two* nonzero values but 20 zero values. These vectors of the property-space are called *Extended Binary Vectors* (EBV) and they are usually very sparse.

A set of m entities will be described by a $m\times N$ binary matrix called the *Binary Connection Matrix* (BCM), because its nonzero elements indicate the connection that exists between each entity and its respective applicable properties, it is more specifically called the *Binary Property Matrix* (BPM). One important feature of EBV is that single-value attributes and multi-value attributes are represented by one and the same vector solving the serious redundancy problem described previously. The 6-*D* attribute-space (the name-attribute is considered a unique identifier) of the criminal file from Figure 1, is transformed into 32-*D* property-space as illustrated in Figure 2. The corresponding BPM is illustrated in Figure 3. Obviously, in this matrix numeric and alphanumeric values are represented in the same manner, there is no need for special encoding as in the RDO model since the nonzero bit position implies the value.

The matrix is very sparse and this is another advantage because the application of sophisticated sparse matrix techniques not only provides the capability to save in storage requirements (only the nonzero elements are stored) but it provides the means to perform very efficient operations on the matrices (only the nonzero elements are operated on) as indicated in References 4 through 7.

Among other advantages the binary matrices enable one to use *Boolean Algebra* algorithms to solve data base problems (see References 8 and 9), to eliminate the difficulties of the range-query, to gather statistical data, to determine the relationships among all entity pairs and all property pairs, to assure privacy and security and many more.

Just looking at the binary matrix of Figure 2, one can scan the rows and see the applicable properties of each and every entity, but scanning the columns enables one to determine all entities that have a specific property. This is the most important feature of the *fully inverted files,* where for each and every property, a list of entities possessing the property is maintained. Therefore, the BCM in the property-space represents both a direct file and a fully inverted file at the same time, which is a very significant feature, to be fully taken advantage of in this investigation.

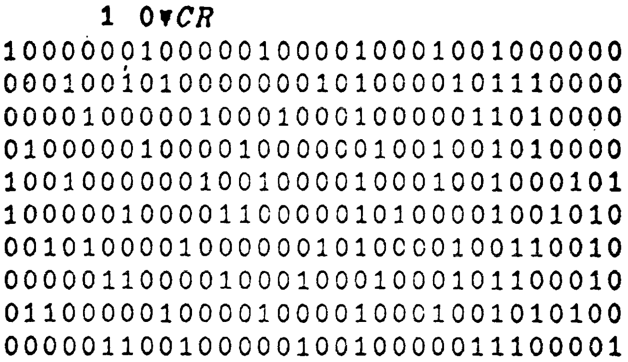
SEARCHING THE PROPERTY-SPACE

The search in this model is performed much in the same manner as the search in the attribute-space model. The query vector must have the same dimensionality as the entity vector, it indicates the coordinates of the projection

		CRIMES							ARM TYPE					LOCATIONS						SEX		EYE COLOR					LANGUAGES						
		Murder	Homicide	Kidnapping	Rape	Hijacking	Drug Push.	Armed Rob.	A	B	C	D	E	Chicago	Detroit	Houston	Los Angeles	New York	Washington	M	F	Black	Blue	Brown	Green	Hazel	English	French	German	Italian	Russian	Spanish	Swedish
ID	NAME	1	2	3	4	5	6	7	1	2	3	4	5	1	2	3	4	5	6	1	2	1	2	3	4	5	1	2	3	4	5	6	7
1	John	1							1						1					1			1				1						
2	Claude				1			1		1								1		1					1		1	1	1				
3	Robert					1						1				1				1						1	1		1				
4	Nancy		1						1					1							1		1				1		1				
5	Ingmar	1			1							1			1					1			1				1				1		1
6	Roberto	1						1						1						1		1					1			1		1	
7	Marcel			1		1					1							1		1					1			1	1			1	
8	Johanna						1	1					1				1				1				1		1	1				1	
9	Jurgen		1	1						1					1								1				1		1		1		
10	Tom						1	1			1						1			1						1	1	1					1
		1	2	3	4	5	6	7	8	9	0 10	1	2	3	4	5	6	7	8	9	0 20	1	2	3	4	5	6	7	8	9	0 30	1	2

* (Crime Location)

Figure 2—The binary property matrix of the RDO sample



```
      1  0⍕CR
100000010000001000010001001000000
000100101000000001010000101110000
000010000010001000100000011010000
010000001000010000001001001010000
100100000001001000010001001000101
100000100001100000101000001001010
001010000010000000101000100110010
000001100001000100010001011000 10
011000000100001000010001001010100
000001100100000010010000011100001
```

Figure 3—The binary property matrix (compact form)

point, on the hyperplane defined by its nonzero values, that all entity points must have in order to qualify. Denoting the $m \times N$ entity matrix by E and the $\ell \times N$ query matrix by Q, the search process is performed again by a generalized inner product (GIP) between the $E=[e_{ij}]$ matrix and the transpose of the $Q=[q_{jk}]$ matrix named Q^T. By post multiplying E by Q^T the $m \times \ell$ response matrix $R=[r_{ik}]$ is obtained. This time, however, a regular matrix multiplication is performed, because the search criterion requires determining the number of corresponding nonzero elements between the entity vector and the query vector (i.e., the number of identical properties specified in the two vectors) and, for binary vectors

$$e_{ij} \times q_{kj} = 1 \text{ iff } e_{ij}=1 \text{ and } q_{kj}=1;$$

$$\text{for other cases } e_{ij}=q_{kj}=0$$

$$R=E+.\times Q^T$$

$$r_{ik}=\sum_{j=1}^{N} e_{ij} \times q_{jk}$$

$$\text{where } r_{ik} \leq t_k$$

$$t_k=\sum_{j=1}^{N} q_{kj}$$

The number of nonzero values specified in a vector is called the *weight of the vector,* the weight of vector Q_k is t_k. The matrices in Figure 4 illustrate the searching process.

The *Negation Query* is another efficient form of file searching. Rather than specifying in the query the properties that an entity must have, the *one* values in the query indicate the properties that the entities must *not* have. For example, the hijacker does not speak English, French and German. By post "multiplying" E by Q^T using a GIP form

$$R=E+.<Q^T \text{ which is}$$

$$\text{r}_{ik}=\sum_{j=1}^{N} (\text{e}_{ij}<\text{q}_{jk}) \quad \text{r}_{ik} \leq \text{t}_k \quad \text{t}_k=\sum_{j=1}^{N} \text{q}_{kj}$$

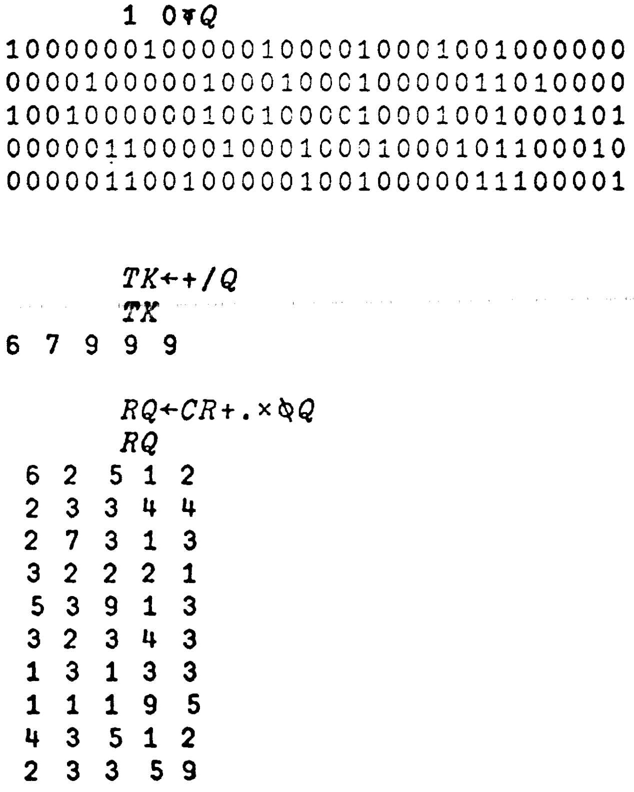
The Range Query. Let attribute Age have six distinct values; 20,21,. . . ,25 and attribute Weight, four distinct values; 160,170,180,190. In the property-space the Age,Weight attributes for m entities are represented by m 10-*D* binary vectors representing the respective property positions

$$\begin{array}{l} P\ (20,21,22,23,24,25,160,170,180,190) \\ E_1(\ 0,\ 0,\ 1,\ 0,\ 0,\ 0,\ \ 0,\ \ 1,\ \ 0,\ \ 0) \\ Q_1(\ 0,\ 1,\ 1,\ 1,\ 1,\ 0,\ \ 0,\ \ 1,\ \ 1,\ \ 1) \end{array}$$

The E_1 entity vector represents a criminal of Age-22 and Weight-170, all other $(m-1)$ entity vectors have *two* nonzero values at different positions. The Q_1 query vector represents the range query for the identification of all criminals of age between 21 to 24 and weight 170 to 190. By post multiplying the $m \times 10$ entity matrix by the transpose of the Q_1 query vector, an $m \times 1$ R_1 response vector is obtained. The elements R_{i1}; $i=1,2,. . . ,m$ of the response vector have values of 0, 1, or 2. Clearly, only entities associated with $R_{i1}=2$ qualify, that $R_{11}=2$ is obvious. As seen in this particular case, the weight of the query vector is $t_k=7$ and $r_{ik} \leq 2$, the cause is selfexplanatory.

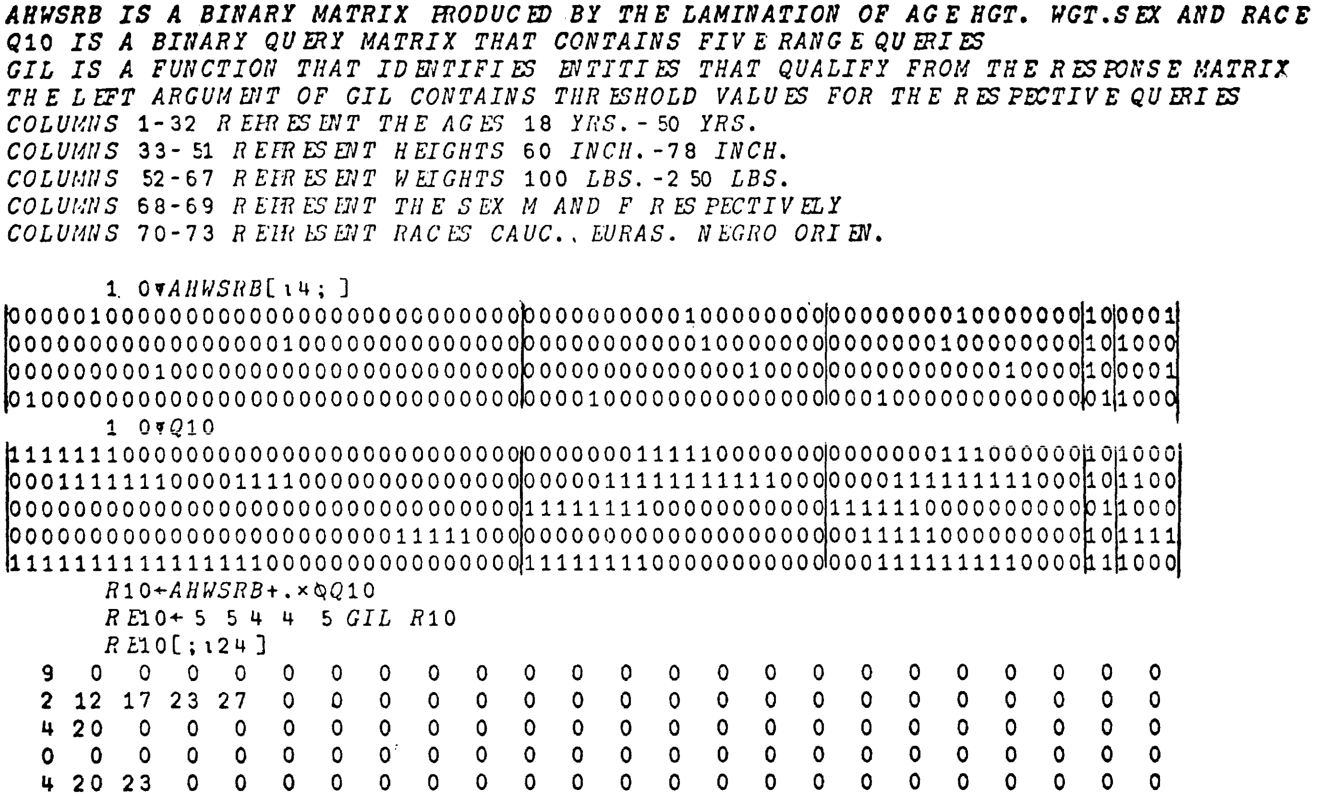
Thus, this is the simple and elegant solution to the complicated range-query problem postulated at the beginning of a former section. Instead of 15 queries and the necessary set-operations, or instead of the 105 queries with their set-operation, one and only one query vector with a weight $t_k=17$ will produce a response vector with elements $r_{ik} \leq 5$, and all the entities with $r_{ik}=5$ will qualify. This is illustrated in Figure 5, where a number of range-queries searched simultaneously the entity set.

The OR-Query. In the RDO model the OR operator always causes a split of the query, in the BPM model it



```
      1  0⍕Q
100000010000001000010001001000000
000010000010001000100000011010000
100100000001001000010001001000101
000001100001000100010001011000010
000001100100000010010000011100001


      TK←+/Q
      TK
6  7  9  9  9

      RQ←CR+.×⍉Q
      RQ
 6  2  5  1  2
 2  3  3  4  4
 2  7  3  1  3
 3  2  2  2  1
 5  3  9  1  3
 3  2  3  4  3
 1  3  1  3  3
 1  1  1  9  5
 4  3  5  1  2
 2  3  3  5  9
```

Figure 4—The searching process, query and response matrices



```
AHWSRB IS A BINARY MATRIX PRODUCED BY THE LAMINATION OF AGE HGT. WGT.SEX AND RACE
Q10 IS A BINARY QUERY MATRIX THAT CONTAINS FIVE RANGE QUERIES
GIL IS A FUNCTION THAT IDENTIFIES ENTITIES THAT QUALIFY FROM THE RESPONSE MATRIX
THE LEFT ARGUMENT OF GIL CONTAINS THRESHOLD VALUES FOR THE RESPECTIVE QUERIES
COLUMNS 1-32 REPRESENT THE AGES 18 YRS. - 50 YRS.
COLUMNS 33- 51 REPRESENT HEIGHTS 60 INCH. -78 INCH.
COLUMNS 52-67 REPRESENT WEIGHTS 100 LBS. -2 50 LBS.
COLUMNS 68-69 REPRESENT THE SEX M AND F RESPECTIVELY
COLUMNS 70-73 REPRESENT RACES CAUC., EURAS. NEGRO ORIEN.

      1 0⍕AHWSRB[⍳4; ]
|00000100000000000000000000000000|0000000000100000000|0000000010000000|10|0001|
|00000000000000001000000000000000|0000000000010000000|0000000100000000|10|1000|
|00000000010000000000000000000000|0000000000000010000|0000000000001000|10|0001|
|01000000000000000000000000000000|0000100000000000000|0001000000000000|01|1000|
      1 0⍕Q10
|11111100000000000000000000000000|0000000111110000000|0000000111000000|10|1000|
|00011111110000111100000000000000|0000011111111111000|0000111111111000|10|1100|
|00000000000000000000000000000000|1111111000000000000|1111110000000000|01|1000|
|00000000000000000000000011111000|0000000000000000000|0011111000000000|10|1111|
|11111111111111110000000000000000|1111111000000000000|0001111111110000|11|1000|
      R10←AHWSRB+.×⍉Q10
      RE10← 5 5 4 4 5 GIL R10
      RE10[;⍳24]
 9  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 2 12 17 23 27  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 4 20  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  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 4 20 23  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Figure 5A—Search by range queries

depends on the form of the request. To identify all criminals of age 27, height 5′10″ and blue or green or hazel eyes requires the formulation of one query vector, of weight $t_k=5$; the entities that qualify have an $r_{ik}=3$. The same request, only this time for criminals of age 27 or 28 or 29, will require again one query of the same dimensionality with a $t_k=7$ and $r_{ik}=3$. Actually, these *OR* queries are range queries. But, if the request is age 27, height 5′10″ and blue or green eyes, or age 28, height 5′10″ and green or hazel eyes, or age 29, height 5′10″ and blue or hazel eyes, three separate queries are required.

Denoting attributes Age, Height and Eyes Color by *A*, *H* and *C*, respectively, and using switching theory notation, the three requests can be formulated as

(1) $A_1H_1(C_1+C_2+C_3)$
$=A_1H_1C_1+A_1H_1C_2+A_1H_1C_3 \rightarrow 3$ triples
(2) $(A_1+A_2+A_3)H_1(C_1+C_2+C_3)$
$=A_1H_1C_1+A_1H_1C_2+\ldots+A_3H_1C_3 \rightarrow 9$ triples
(3) $A_1H_1(C_1+C_2)+A_2H_1(C_2+C_3)+A_3H_1(C_1+C_3)$
$=A_1H_1C_1+A_1H_1C_2+\ldots+A_3H_1C_3 \rightarrow 6$ triples

Obviously, all entities that can be projected in any of the three points specified by the triples of the first request will qualify for the first, second and third request, all the entities that can be projected in any of the six points specified by the third request will qualify for the second and third requests but not all entities that qualify for the second request will qualify for the first and third requests. In the property-space, *m* entities can be represented by a subschema that contains all the properties specified in the three requests by *m*, 7-*D* binary vectors representing the respective property positions associated with each entity. Each and every entity can have a maximum of three nonzero values, because Age, Height and Eyes Color are single-value attributes.

$$\begin{array}{l}
P(A_1,A_2,A_3,H_1,C_1,C_2,C_3)\\
E_1(1,\ 0,\ 0,\ 1,\ 0,\ 0,\ 1)\\
Q_1(1,\ 0,\ 0,\ 1,\ 1,\ 1,\ 1)\\
Q_2(1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1)\\
Q_3'(1,\ 0,\ 0,\ 1,\ 1,\ 1,\ 0)\\
Q_3''(0,\ 1,\ 0,\ 1,\ 0,\ 1,\ 1)\\
Q_3'''(0,\ 0,\ 1,\ 1,\ 1,\ 0,\ 1)
\end{array}$$

It is apparent, however, that while the Q_1 query vector defines the three necessary triples and the Q_2 query vector defines the nine necessary triples, to define the six necessary triples for the third request three separate queries Q_3',Q_3'' and Q_3''' have to be formulated. This is required in order to exclude the three underscored triples from the nine

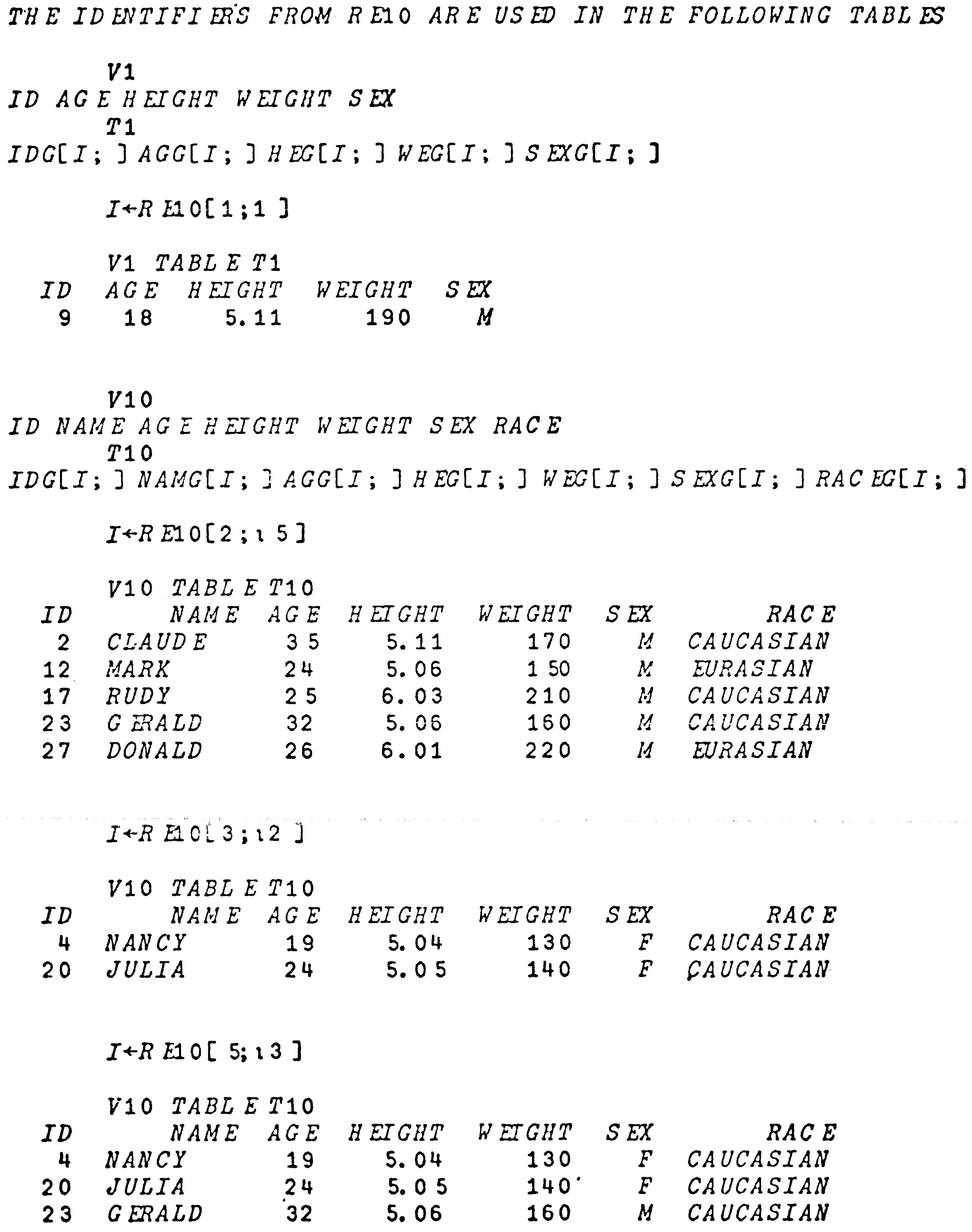
$$\begin{array}{c}
A_1H_1C_1;A_1H_1C_2;\underline{A_1H_1C_3}\\
\underline{A_2H_1C_1};A_2H_1C_2;\underline{A_2H_1C_3}\\
\underline{A_3H_1C_1};\underline{A_3H_1C_2};A_3H_1C_3
\end{array}$$

possible triples. The response to the third request is obtained by taking the *union* of the three subsets retrieved by

Q_3', Q_3'' and Q_3'''. Observe that entity E_1 that possesses the projection $\underline{A_1H_1C_3}$, qualifies for the first and second requests but does not qualify for the third. Informally, it may be stated that the number of separate queries to be formulated is equal to the number of members in the left-hand side of the equation.

HUMAN-FACE IDENTIFICATION

Human-face features of a large population can be stored in a computer. The problem is how to structure the data so that an efficient search can be performed to identify an individual when a number of features is specified, or how to



```
THE IDENTIFIER'S FROM RE10 ARE USED IN THE FOLLOWING TABLES

       V1
ID AGE HEIGHT WEIGHT SEX
       T1
IDG[I; ]AGG[I; ]HEG[I; ]WEG[I; ]SEXG[I; ]

       I←RE10[1;1 ]

       V1 TABLE T1
  ID   AGE  HEIGHT  WEIGHT  SEX
   9    18    5.11     190    M


       V10
ID NAME AGE HEIGHT WEIGHT SEX RACE
       T10
IDG[I; ]NAMG[I; ]AGG[I; ]HEG[I; ]WEG[I; ]SEXG[I; ]RACEG[I; ]

       I←RE10[2;ι 5]

       V10 TABLE T10
  ID        NAME  AGE  HEIGHT   WEIGHT  SEX        RACE
   2  CLAUDE       35    5.11      170    M   CAUCASIAN
  12  MARK         24    5.06      1 50   M   EURASIAN
  17  RUDY         25    6.03      210    M   CAUCASIAN
  23  GERALD       32    5.06      160    M   CAUCASIAN
  27  DONALD       26    6.01      220    M   EURASIAN


       I←RE10[3;ι2 ]

       V10 TABLE T10
  ID        NAME  AGE  HEIGHT   WEIGHT  SEX        RACE
   4  NANCY        19    5.04      130    F   CAUCASIAN
  20  JULIA        24    5.05      140    F   CAUCASIAN


       I←RE10[ 5;ι3 ]

       V10 TABLE T10
  ID        NAME  AGE  HEIGHT   WEIGHT  SEX        RACE
   4  NANCY        19    5.04      130    F   CAUCASIAN
  20  JULIA        24    5.05      140'   F   CAUCASIAN
  23  GERALD       32    5.06      160    M   CAUCASIAN
```

Figure 5B—Retrieval after range query search

classify individuals in distinct classes as a function of their features. A few researchers have investigated the problem.[10,11] It appears, however, that more efficient methods are needed to obtain a quick answer, especially when many different descriptions are to be considered simultaneously.

Structuring the data in a binary property-matrix form, as illustrated in Figure 6, and Figure 7, provides the means for fast identification and classification. The tables are self-explanatory, the attributes can be permuted in any order so that many different sub-schemas can be derived. If a witness describes a criminal that had his face covered from the nose down, only the hair, eyebrows, eyes, forehead and ears are needed in the sub-schema. A witness may describe a criminal without being able to assign a specific value to each and every feature; in other words, features may be described with different degrees of uncertainty (DOU). For example, describing the nose, the witness may say that he is not sure if the nose-length should be considered as short or medium and if the profile is concave, straight, or somewhere inbetween. In this particular case, these attributes will be covered by a *range-query* of the form (1,1,1,-0,0,1,1,1,0,0).

The strength of the method is not restricted to the capability of applying range-queries to each and every attribute at once, so as to overcome the uncertainty problem. Assume that several witnesses describe the same criminal, or a number of criminals, with different DOUs, the process of interrogating the entity set with many queries concurrently, called a parallel search, provides the capability to check a number of alternatives, as illustrated in Figure 8.

Distinctive features are easily discovered in this model by calculating a figure of merit called the *Degree of Distinction* (DOD). Assume that an $m\times N$ matrix $E=[e_{ij}]$ contains the data of m human faces, some of the N features are associated with fewer faces than others, these are the more distinctive features. The DOD for each and every property can be calculated by dividing the column vector weights of the E matrix by m,

$$\mathrm{DOD}(j)=\frac{1}{m}\sum_{i=1}^{m} e_{ij}$$

The lower DOD(j) values indicate the most distinctive properties P_j, which are very important indicators in determining the features to be described in a query to get the most pertinent answer.

The human face file contains n single-value attributes (eyes-opening, eyes-separation, eyes-color are three distinct attributes) and considering that the row vector weights of an n-attribute entity set is $\sum_{j=1}^{N} e_{ij}=n$, the degree of uncertainty reflected by each query vector of a $Q=[q_{kj}]$ query matrix that interrogates the $E=[e_{ij}]$ entity matrix can

	FOREHEAD					CHEEKS					CHIN					MOUTH								NOSE													EARS									
											Profile					Width					Lip*			Length					Profile					Tip			Length					Protusion				
	Receding		Vertical		Bulging	Sunken		Average		Full	Receding		Straight		Jutting	Small		Medium		Large	Upper	Neither	Lower	Short		Medium		Long	Concave		Straight		Hooked	Upward	Horizon	Downward	Short		Medium		Long	Slight		Medium		Large
ID	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5	1	2	3	1	2	3	4	5	1	2	3	4	5	1	2	3	1	2	3	4	5	1	2	3	4	5
1	1									1		1								1	1				1						1				1					1				1		
2		1					1								1		1						1					1					1			1		1								1
3				1					1					1					1		1					1					1				1						1	1				
4			1					1					1			1						1		1					1					1			1		1						1	
5					1	1						1						1					1				1					1			1					1			1			
6				1					1						1					1		1					1			1				1			1						1			
7		1					1				1					1						1			1							1				1		1				1				
8			1			1							1					1			1					1					1				1				1					1		
9	1							1			1								1			1						1	1					1						1					1	
10					1					1					1		1						1					1					1			1					1					1
	4	5	6	7	8	9	0 40	1	2	3	4	5	6	7	8	9	0 50	1	2	3	4	5	6	7	8	9	0 60	1	2	3	4	5	6	7	8	9	0 70	1	2	3	4	5	6	7	8	9

*(Lip Overlap)

Figure 6—Additional features of human-faces

```
     MN E
    MOUTH                        NOSE                              EARS

0 0 0 0 1 |1 0 0 |0 1 0 0 0 |0 0 1 0 0 |0 1 0 |0 0 0 1 0 |0 0 1 0 0
0 1 0 0 0 |0 0 1 |0 0 0 0 1 |0 0 0 0 1 |0 0 1 |0 1 0 0 0 |0 0 0 0 1
0 0 0 1 0 |1 0 0 |0 0 1 0 0 |0 0 1 0 0 |0 1 0 |0 0 0 0 1 |1 0 0 0 0
1 0 0 0 0 |0 1 0 |1 0 0 0 0 |1 0 0 0 0 |1 0 0 |1 0 0 0 0 |0 0 0 1 0
0 0 1 0 0 |0 0 1 |0 0 0 1 0 |0 0 0 1 0 |0 1 0 |0 0 0 1 0 |0 1 0 0 0
0 0 0 0 1 |0 1 0 |0 0 0 1 0 |0 1 0 0 0 |1 0 0 |1 0 0 0 0 |0 1 0 0 0
1 0 0 0 0 |0 1 0 |0 1 0 0 0 |0 0 0 1 0 |0 0 1 |0 1 0 0 0 |1 0 0 0 0
0 0 1 0 0 |1 0 0 |0 0 1 0 0 |0 0 1 0 0 |0 1 0 |0 0 1 0 0 |0 0 1 0 0
0 0 0 1 0 |0 1 0 |0 0 0 0 1 |1 0 0 0 0 |1 0 0 |0 0 0 1 0 |0 0 0 1 0
0 1 0 0 0 |0 0 1 |0 0 0 0 1 |0 0 0 0 1 |0 0 1 |0 0 0 0 1 |0 0 0 0 1
```

```
    FCC←FOR,CH-E,CHI                          NOS E←NOL,NO P,NOT
1 0 0 0 0 |0 0 0 0 1 |0 1 0 0 0         0 1 0 0 0 |0 0 1 0 0 |0 1 0
0 1 0 0 0 |0 1 0 0 0 |0 0 0 0 1         0 0 0 0 1 |0 0 0 0 1 |0 0 1
0 0 0 1 0 |0 0 0 1 0 |0 0 0 1 0         0 0 1 0 0 |0 0 1 0 0 |0 1 0
0 0 1 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
0 0 0 0 1 |1 0 0 0 0 |0 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 0 0 0 1         0 0 0 1 0 |0 1 0 0 0 |1 0 0
0 1 0 0 0 |0 1 0 0 0 |1 0 0 0 0         0 1 0 0 0 |0 0 0 1 0 |0 0 1
0 0 1 0 0 |1 0 0 0 0 |0 0 1 0 0         0 0 1 0 0 |0 0 1 0 0 |0 1 0
1 0 0 0 0 |0 0 1 0 0 |1 0 0 0 0         0 0 0 0 1 |1 0 0 0 0 |1 0 0
0 0 0 0 1 |0 0 0 0 1 |0 0 0 0 1         0 0 0 0 1 |0 0 0 0 1 |0 0 1
```

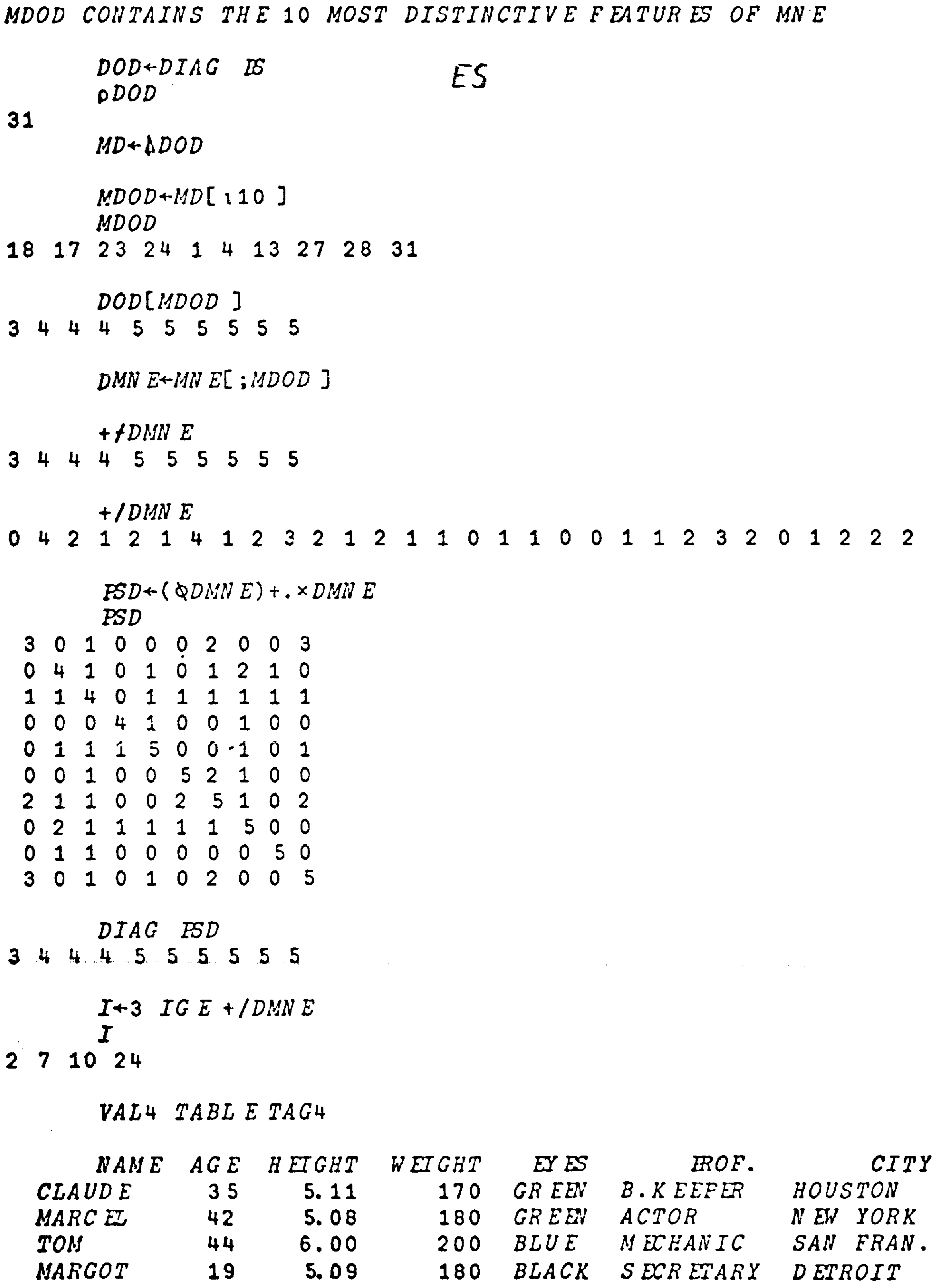
```
    MOUTH←MOW,MOL                EARS← EAL, EA P
0 0 0 0 1 |1 0 0           0 0 0 1 0 |0 0 1 0 0
0 1 0 0 0 |0 0 1           0 1 0 0 0 |0 0 0 0 1
0 0 0 1 0 |1 0 0           0 0 0 0 1 |1 0 0 0 0
1 0 0 0 0 |0 1 0           1 0 0 0 0 |0 0 0 1 0
0 0 1 0 0 |0 0 1           0 0 0 1 0 |0 1 0 0 0
0 0 0 0 1 |0 1 0           1 0 0 0 0 |0 1 0 0 0
1 0 0 0 0 |0 1 0           0 1 0 0 0 |1 0 0 0 0
0 0 1 0 0 |1 0 0           0 0 1 0 0 |0 0 1 0 0
0 0 0 1 0 |0 1 0           0 0 0 1 0 |0 0 0 1 0
0 1 0 0 0 |0 0 1           0 0 0 0 1 |0 0 0 0 1
```

```
   FOR           CH E          CHI           MOW           NOL
1 0 0 0 0     0 0 0 0 1     0 1 0 0 0     0 0 0 0 1     0 1 0 0 0
0 1 0 0 0     0 1 0 0 0     0 0 0 0 1     0 1 0 0 0     0 0 0 0 1
0 0 0 1 0     0 0 0 1 0     0 0 0 1 0     0 0 0 1 0     0 0 1 0 0
0 0 1 0 0     0 0 1 0 0     0 0 1 0 0     1 0 0 0 0     1 0 0 0 0
0 0 0 0 1     1 0 0 0 0     0 1 0 0 0     0 0 1 0 0     0 0 0 1 0
0 0 0 1 0     0 0 0 1 0     0 0 0 0 1     0 0 0 0 1     0 0 0 1 0
0 1 0 0 0     0 1 0 0 0     1 0 0 0 0     1 0 0 0 0     0 1 0 0 0
0 0 1 0 0     1 0 0 0 0     0 0 1 0 0     0 0 1 0 0     0 0 1 0 0
1 0 0 0 0     0 0 1 0 0     1 0 0 0 0     0 0 0 1 0     0 0 0 0 1
0 0 0 0 1     0 0 0 0 1     0 0 0 0 1     0 1 0 0 0     0 0 0 0 1
```

Figure 7—Binary matrices of human-face features

```
      QFCC
0 0 0 0 0 1 1 0 0 0 0 0 1 1 1
0 1 1 1 0 0 0 1 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 1 0
0 0 0 1 1 0 1 1 0 0 0 1 1 1 0
1 1 0 0 0 1 1 1 0 0 0 0 0 1 1
0 0 1 1 1 0 0 1 0 0 1 1 1 0 0
1 0 0 0 0 0 0 1 0 0 1 1 0 0 0
1 1 0 0 0 1 0 0 0 0 0 0 0 1 0
      TK
5 6 2 7 7 7 4 4
      VK
2 2 1 3 3 3 3 3

      DOUK←TK-VK
      DOUK
3 4 1 4 4 4 1 1

      RFCC←FCC+.×⍉QFCC
      TRFCC←V GIL RFCC
      TRF←TRFCC[;⍳12]

      TRF
 2  8 11 14 28 29  0  0  0  0  0  0
 3  4  6 15 19 22 23 25 26 27 30  0
 3  4  8 12 15 23 25 26 28  0  0  0
23 30  0  0  0  0  0  0  0  0  0  0
 2 27  0  0  0  0  0  0  0  0  0  0
 4 23 30  0  0  0  0  0  0  0  0  0
 9 24  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0

      VAL3
ID NAME CRIME LOCATION DATE DETECTIVE
      TAG3
IDG[I2;] NAMG[I2;] CRG[I2;] LOCG[I2;] DATEG[I2;] DETG[I2;]
      I2←TRF[6;⍳3]

      VAL3 TABLE TAG3

 ID       NAME        CRIME      LOCATION         DATE    DETECTIVE
  4   NANCY       HOMICIDE     DETROIT      03/15/73    GONZALEZ
 23   GERALD      ARMED ROB.   DETROIT      12/25/75    BURNSIDE
 30   JEAN        ARMED ROB.   LOS ANG.     12/25/72    EXPOSITO

      DODJ←(+/FCC)÷1↑⍴FCC
      4 2⍕DODJ
 .13 .17 .23 .27 .20 .20 .23 .23 .17 .17 .23 .20 .27 .03 .27
```

Figure 8—Search and retrieval by queries with different DOUs



```
MDOD CONTAINS THE 10 MOST DISTINCTIVE F EATUR ES OF MN E

      DOD←DIAG ES                ES
      ⍴DOD
31
      MD←⍒DOD

      MDOD←MD[⍳10 ]
      MDOD
18 17 23 24 1 4 13 27 28 31

      DOD[MDOD ]
3 4 4 4 5 5 5 5 5 5

      DMN E←MN E[ ;MDOD ]

      +⌿DMN E
3 4 4 4 5 5 5 5 5 5

      +/DMN E
0 4 2 1 2 1 4 1 2 3 2 1 2 1 1 0 1 1 0 0 1 1 2 3 2 0 1 2 2 2

      PSD←(⍉DMN E)+.×DMN E
      PSD
 3 0 1 0 0 0 2 0 0 3
 0 4 1 0 1 0 1 2 1 0
 1 1 4 0 1 1 1 1 1 1
 0 0 0 4 1 0 0 1 0 0
 0 1 1 1 5 0 0 -1 0 1
 0 0 1 0 0 5 2 1 0 0
 2 1 1 0 0 2  5 1 0 2
 0 2 1 1 1 1 1  5 0 0
 0 1 1 0 0 0 0 0  5 0
 3 0 1 0 1 0 2 0 0  5

      DIAG PSD
3 4 4 4 5 5 5 5 5 5

      I←3 IG E +/DMN E
      I
2 7 10 24

      VAL4 TABL E TAG4

      NAME   AGE  H EIGHT   W EIGHT   EY ES    PROF.        CITY
    CLAUD E   35    5.11      170   GR EEN  B.K EEPER   HOUSTON
    MARC EL   42    5.08      180   GR EEN  ACTOR       N EW YORK
    TOM       44    6.00      200   BLU E   M ECHANIC   SAN FRAN.
    MARGOT    19    5.09      180   BLACK   S ECR ETARY D ETROIT
```

Figure 9—Similarity matrix of ten distinctive features

be determined as follows

$$\mathrm{DOU}(k)=\sum_{j=1}^{N} q_{kk}-V(k) \quad \text{where } V(k)\leq n$$

The $V(k)$ value is determined by the number of attributes covered by query Q_k. The two defined coefficients DOD and DOU can be calculated in many ways as illustrated in Figure 8, but they are readily obtained from Similarity Matrices.

The Entity Similarity Matrix (ES) is a symmetric $m\times m$ matrix obtained by post-multiplying the entity matrix E by its transpose

$$\mathrm{ES}=E+.\times E^T=[e_{ij}]\times[e_{jk}] \quad es_{ik}=\sum_{j=1}^{N} e_{ij}e_{jk}$$

where $i=1,2,.\ .\ .\ ,m;\ k=1,2,.\ .\ .\ ,m.$,

The (i,j)th element of ES represents the number of common properties possessed by the pair of entities E_i and E_j. When $i=j$ the elements (i,i) of ES are the elements of the main diagonal, each element representing the number of properties specified in entity E_i, which is the weight of vector E_i. The (i,j) elements indicate the degree of similarity between entities E_i and E_j.

The Property Similarity Matrix (PS) is obtained by pre-multiplying the E matrix by its transpose E^T, it is an $N\times N$ symmetric matrix $\mathrm{PS}=E^T+.\times E$. The (i,j)th element of PS represents the number of entities that possess the properties P_i *and* P_j. The (i,i)th element of the main diagonal represents the number of entities associated with property P_i. Dividing the (i,i) elements by m the DOD(i) is obtained.

The Query Similarity Matrix (QS) is obtained by postmultiplying the query matrix Q by its transpose Q^T, if Q is an $\ell \times N$ matrix, QS is a symmetric $\ell\times\ell$ matrix. The (i,j)th element of QS represents the number of common properties specified in queries Q_i and Q_j. The (i,i)th element of the main diagonal represents the number of properties specified in query Q_i, this number was defined, previously, as the query complexity degree QCD. Subtracting from the element of the main diagonal the $V(i)$ value, which represents the number of attributes covered by query Q_i, the *DOU*(i) is obtained.

The $V(i)$ value is called the *Covering Coefficient* of query Q_i and as stated previously $V(i)\leq n$. If there are n attributes it is known that $N=\Sigma_{i=1}^{n} d_i$ is the dimensionality of the query vector in the property-space. One can, however, specify in a Q_i query vector of dimensionality N, properties pertaining to k domains $(k\leq n)$; i.e., the query *covers* only k attributes, therefore $V(i)=k$.

The Query-Property Similarity Matrix (QPS) is an $N\times N$ symmetric matrix obtained by pre-multiplying the Q matrix by its transpose Q^T. The (i,j) element of QPS indicates the number of queries in matrix Q in which properties P_i *and* P_j are specified, this is a *Pair-Usage-Frequency Indicator* (PUFI) because it indicates how frequently pairs of properties are specified in queries. The (i,i) element of QPS indicates the number of queries in matrix Q in which property P_i is specified, it is called the Usage Frequency Indicator (UFI). The UFI is a very important factor in determining the best strategy for storage allocation.

One of the most important aspects of this associative approach is its great flexibility in *Logical Reconfiguration*; i.e., the capability to "cut and paste" attributes or properties as a function of the time-variable queries. As illustrated in Figure 7 by the matrices FOR; CHE; CHI; MOW; MOL; NOL; NOP, etc., one can concentrate on the forehead, cheeks, chin, mouth's width, mouth's lip overlap, nose's length, nose's profile, respectively, or any combination thereof, to create new subschemas or *Aggregates*. If a witness recalls that a criminal has a long nose and curly hair, the search can be performed over attributes hair-texture and nose-length, or over any specific properties. The flexibility of this approach enables one to create aggregates of properties that belong to different attributes. One can also create aggregates of the most distinctive features stored in a file, for example, the ten properties with the lowest DOD values as illustrated in Figure 9.

If the intersection of the set of criminals that possess the most distinctive properties with the set of criminals retrieved by the search over the properties specified by the witness is not empty, the interrogator may ask the witness to try and recall if the criminal did not have any of the most distinctive features.

CONCLUSIONS

Only a few basic concepts of SPARCOM have been presented in this paper. Many other concepts and algorithms have also been implemented and tested. The APL is used here merely as a tool to explain the concepts. The strength of the method and the efficiency of the system have been clearly demonstrated in an Assembler Language implementation when a Data Base of 100,000 entities vs. 65,000 potential properties was searched concurrently by 50 complex queries. (Each query contained over 30 different properties from a set of 65,000 potential properties.) On a system 360 Mod 85 all entities that qualified were retrieved within 60 seconds. A detailed description of SPARCOM's concepts can be found in References 7 and 12.

REFERENCES

1. Codd, E. F., "A Relational Model for Data for Large Shared Data Banks," *Comm. ACM* 13, 6, June 1970, pp. 377-387.
2. Date, C. J., *An Introduction to Data Base Systems,* Addison-Wesley Pub. Co., Reading, Massachusetts, 1975.
3. Martin, J., *Computer Data-Base Organization,* Prentice-Hall, Inc., Englewood Cliffs, N.J., 1975.
4. Ashany, R., "Sparse Matrix Techniques for Information Storage and Retrieval," *Proc. of Fifth Annual Princeton Conf. on Inf. Sciences and Systems,* Princeton, N.J., March, 1971.
5. Ashany, R., "Automatic Classification and Relationship Analysis for Information Storage and Retrieval," *Proc. of Third Southeastern Conf. on System Theory,* Atlanta, Georgia, April, 1971.
6. Ashany, R., "Concepts of Data Manipulation—The Connection Matrix Method," IBM Technical Report T. R. 00.2200, Poughkeepsie, N.Y., June, 1971.

7. Ashany, R., "SPARCOM: A Sparse Matrix Associative Relational Approach to Dynamic Data Structuring and Data Retrieval," IBM Technical Report T. R. 00.2774 Poughkeepsie, N.Y., July 1976.
8. Delobel, C. and R. G. Casey, "Decomposition of Data Base and the Theory of Boolean Switching Functions," *IBM J. R&D* 17, 5, September 1973, pp. 374-386.
9. Delobel, C. and J. Rissanen, "Decomposition of Files, A Basis for Data Storage and Retrieval," IBM Research Report RJ1220, San Jose, Calif., May 10, 1973.
10. Goldstein, L. D., et al., "Man-Machine Interaction in Human-Face Identification," *The Bell System Technical Journal,* Vol. 51, No. 2, February 1972, pp. 399-427.
11. Goldstein, L. D. et al., "Identification of Human Faces," *Proc. IEEE,* 59, No. 5, 1971, pp. 748-760.
12. Ashany, R., "SPARCOM: A Sparse Matrix Associative Relational Approach to Dynamic Data Structuring and Data Retrieval," Ph.D. Dissertation, Dept. of Electrical Engineering, Polytechnic Institute of New York, June 1976.

Integrated data base concepts and structures for combat models

by WILLIAM A. BAYSE and DEAN P. RISSEEUW
US Army Concepts Analysis Agency
Bethesda, Maryland

and

CHARLES S. MATHENY
Centec Consultants, Inc.
Reston, Virginia

ABSTRACT

Application of computer-based combat simulation models in support of major Army force design and weapons systems analyses frequently involves the use of large and complex sets of data from various organizational sources, reporting systems and data acquisition activities. Emerging data base technology has resulted in powerful techniques and operational tools which provide a range of new and versatile capabilities to structure and handle efficiently the large data sets which often pervade the analytic process. Of particular importance in the area of data base management systems (DBMS) are automated features which facilitate definition, integration and audit trail of data elements and structures into a cohesive, useful body of accessible user-oriented information. As a principal activity of a command-wide model/data quality assurance program, the US Army Concepts Analysis Agency (CAA) is employing the technology and software of data base management systems to construct an integrated data base to support simulation model operation and analytic efforts. This paper presents a description of design concepts, mathematical and systems analysis techniques, quality assurance methodologies, and resulting integrated automated data base capabilities which are in use or development at CAA to support major Army analyses and simulation model operations.

INTRODUCTION

Major mission activities of the US Army Concepts Analysis Agency (CAA) involve quantitative studies and analyses directed principally toward mid-range and long-range force concepts. Additionally, substantial mission effort is allocated to special analyses concerning materiel systems mix and resource management issues important to Army decision-making in PPBS and acquisition processes. To support an increasingly broad spectrum of study areas, a large, active inventory of automated models, information systems and utility programs are maintained as primary tools of analysis. Computer-based combat simulation models addressing theater, intermediate and small unit levels comprise a major portion of the CAA inventory. Model methodology development and improvement work is continuous under an agency-wide program. Application of computer-based combat simulation models in support of major Army force design and material systems analyses usually involves the use of large-volume, complex sets of data from various organizational sources, reporting systems and data from various organizational sources, reporting systems and data acquisition activities. Representations of combat activity at various levels require a wide variety of data. Significantly, models used to simulate combat at different force or organizational levels often require varying data elements for a given weapon system. For example, a high-resolution model such as CARMONETTE* uses probability-of-hit data for a weapon system such as the tank; whereas, a low-resolution model such as the Concepts Evaluation Model (CEM)** uses an aggregate measure of tank firepower capability. With about 20 combat models and an equal number of other types of automated models and utilities (e.g., mathematical routines) in the CAA inventory at any time, the collection of technical, operational and management functions required for models and data necessarily encompasses an extensive, complex and resource-consuming enterprise. Activities such as model logic and user interface improvements and data validation are integral to the conduct of studies and are essential to analytic productivity and success.

* High-resolution stochastic model for simulation of ground combat up to reinforced company level. (Used in Army Cost and Operational Effectiveness Analyses (COEA's); e.g., XM-1, AAH.)

** Low-resolution theater combat model. (Used in major Army force design and capability studies; e.g., Total Force, OMNIBUS, CONAF.)

GENERAL REQUIREMENTS FOR MODEL/DATA QUALITY ASSURANCE

The importance of the role of models and data in Army studies conducted at CAA prompted the formulation of a command-wide program for model/data quality assurance (QA). One of the four foundation constituents of the CAA QA program is development of a data base† for model applications. The QA program data base component establishes and maintains a controlled, evolutionary path to an agency-wide model-oriented data base. Ultimate objectives of data base development are long-range in nature; however, the project is subdivided and staged in a manner which yields near-term products useful in studies and analyses. Under any developmental scheme—short- or long-range—data base management for models is a challenging and complex task. To demonstrate the wide variety of data types, Figure 1 contains a sample list of candidates (now in individual files) for inclusion in a composite, model-oriented data base. Associated quality assurance requirements are commensurately complex. For example, a vital and extremely intricate facet of data base management and quality assurance involves data screening, editing, analysis and verification. Additionally, driven by the model-orientation of CAA's data base effort is the requirement for automated user/model/data base interfaces. In this latter area, user-oriented, conversational (man-machine) linkages are under development for selected models—a particularly demanding and advanced technical ADP and human engineering task (in this task, a serious complication arises from the number and variety of autonomous data file schemes developed in conjunction with models over two or more generations of digital computers and related software). Other key ingredients of data base management include definition of relationships or logical connections among individual elements (items) of data‡ and audit trail or journal log of all data changes. The CAA data base program evolved from a critical need to manage and assure the quality of model inputs. The complexity of this endeavor led to a full investigation of data base technology and data base management tools to underpin and facilitate the effort. The importance of this investigative work led to the inclusion of several ensuing sections of this paper which discuss the analytical considerations, data base technology and related operational capabilities which are directly involved in the program and are fundamental to its progress and success.

FORCE COMPOSITION
WEAPONS' CHARACTERISTICS
TERRAIN
MOVEMENT RATES
CASUALTY RATES
FFP/WEI/ICE
SENSORS' CHARACTERISTICS
VULNERABILITY
LETHAL AREA

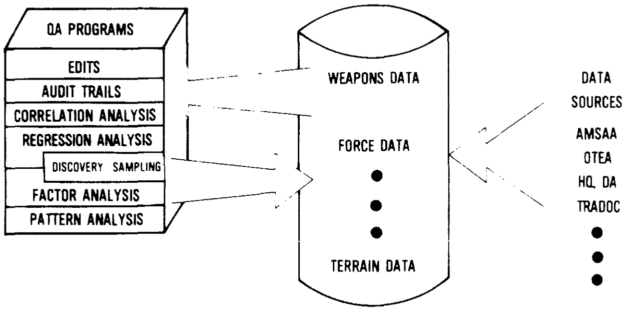
WEAPON/AMMUNITION
P_k & P_d TABLES AND FACTORS
BATTLEFIELD PARTIONING
SUB-UNIT STATUS (SUSF)
UNIT CHARACTERISTICS
DEPLOYMENT SCHEDULES
AMMUNITION CHARACTERISTICS
RESOURCE REQUIREMENTS/UNIT OR WEAPON

Figure 1—Candidate data files

DATA BASE CHARACTERISTICS

The model-oriented data base considerations outlined in the preceding sections lead to a number of characteristics related to command-wide data base requirements. The diagram in Figure 2 highlights a number of general characteristics which are objectives of the CAA data base.

(1) A keystone feature is application of quality assurance procedures to the model-oriented data.
(2) Basic data validity targets such as accuracy, consistency and currency are tractible where data inputs flow from identifiable sources (preferably, those designated and recognized as sole competent sources) and are placed in a visible, central repository.
(3) Management and technical control procedures contribute to proper selection and application of data for use in studies.
(4) As indicated previously, analysis of quantitative data base entries by means of a variety of quality assurance utility programs is a primary area of payoff.



- CENTRAL REPOSITORY
- MANAGEMENT & TECHNICAL CONTROL-QUALITY ASSURANCE
- DATA ITEM RELATIONSHIPS
- AUDIT, ANALYSIS, CORRECTION
- AUTOMATED UTILITIES
- MODEL INTERFACES-STUDY SUPPORT

Figure 2—Data base characteristics

† "A data base may be defined as a collection of interrelated data stored together without harmful or unnecessary redundancy to serve one or more applications in an optimal fashion; the data are stored so that they are independent of programs which use the data; a common and controlled approach is used in adding new data and in modifying and retrieving existing data within the data base. One system is said to contain a collection of data bases if they are entirely separate in structure." James Martin: *Computer Data-Base Organization*, Englewood Cliffs, NJ, Prentice-Hall, Inc., 1975.

‡ Considerable work in the field of data base technology has been devoted to terminology; e.g., "An information system deals with objects and events in the real world that are of interest. These real objects or events, called 'entities,' are represented in the system by data. Information about a particular entity is in the form of 'values' which describe quantitatively and/or qualitatively a set of attributes that have significance in the system." CODASYL Development Committee, "An Information Algebra, Phase I Report of the Language Structure Group," *Communications of the ACM*, 5, 4 (Apr 62).

Beyond the use of conventional edits, multivariate statistical analysis can be applied to screen the data base and identify questionable values or "out-liers" for closer scrutiny and verification.

(5) Selection of random samples for detailed analytic verification is another technique—sometimes termed discovery sampling—for assuring data integrity.

(6) Audit trails are instrumental in tracing modifications to data quantities. For example, a quality audit[7] is an attempt to determine through examination of original and modified data, who or what procedure changed the value(s). Data may be of poor quality for several reasons: not good originally; altered by human error; altered by a program (model) with a bug; altered by a machine error; or destroyed by a major problem such as general mechanical failure of an automated storage device. To enhance data validity, data base facilities must exist which provide precautionary, diagnostic, prescriptive and/or corrective measures in dealing with the challenges of data quality in a modeling environment. Data audits support such facilities by making transaction changes and their detailed effects visible to and traceable by the analyst.

Finally, a characteristic central to data base development concepts is the capability to remove a portion of time-consuming data preparation and handling work from the study team and to free them for additional diagnosis and analysis of inputs, model processes and outputs.

DBMS—THE FACILITATING MECHANISM

Emerging data base technology has resulted in powerful techniques and operational automated tools which offer a range of new, versatile capabilities to structure and manage efficiently the large bodies of data which are often required to carry out the analytic process. Compatible with concepts outlined in the foregoing discussion of data base characteristics are the underlying objectives[7] of data base management technology:

(1) To make an integrated collection of data available to a wide variety of users—(in our case, analysts and/or models);
(2) To provide for quality and integrity of the data—(quality assurance);
(3) To insure retention of privacy through security within the system—(technical control);
(4) To allow centralized control of the data base, which is necessary for efficient data administration—(central repository; management/technical control).

Since the late 1960's, the idea of organizing data bases into a single collection of files in which (roughly speaking) each data element appears only once has gained increasing acceptance. The idea grew out of numerous situations in which ever larger efforts were required to write new programs to operate on data used and maintained by other programs, and to devise systems for insuring that a data element used in several programs was current in each. A manager's request to look at data in a new way might entail several weeks of programmer effort to extract the required data from several files, perhaps having incompatible time frames. Changes to record formats in a file to accommodate new data elements could result in the need to modify many programs that operate on the file.

To implement the idea of a single integrated data base for an organization required software for organizing, maintaining and accessing the data base. A software package designed to perform these functions is called a data base management system (DBMS). A considerable number of such systems exist today. They have been implemented in several thousand installations. They have generally achieved the improvements in data management and reductions in programming that motivated their development.

Current DBMS's may be thought of as primarily second generation systems, having been developed and implemented since publication of the CODASYL recommendations in 1971. The CODASYL report culminated about two years of intensive study, by users and computer equipment companies, of the requirements of an effective DBMS. Experiences with a few first generation DBMS's were examined in depth. While not all DBMS's have adopted the CODASYL viewpoint, and none has implemented all of the CODASYL features, the CODASYL report stands as a milestone in DBMS development and as a framework against which to measure the capabilities of a DBMS.

No standards on DBMS design have been adopted to date. However, the American National Standards Institute has established a group to study the desirability of defining standards.

The objectives of DBMS derive naturally and consequentially from those expressed for data base technology. To provide insights into the potential role of DBMS in the CAA analytic/modeling environment, key objectives and terminology are summarized as follows:

- *Minimize redundancy*. Efficiency is often not best served by *complete* elimination of redundancy, but the idea of holding redundancy of data elements to a minimum is a major objective of DBMS's. It reduces storage space and reduces the effort necessary to insure currency and consistency among multiple uses of the data.
- *Data independence*. The single most important aim of a DBMS is often to relieve data users of concern for the logical and physical structures§ in which data are stored. This is possible to the extent that data elements can be retrieved and assembled into records and files tailored to particular uses as needed. When data inde-

§ "... information (data) structuring (the selection of entities and specification of relationships between them) is a modeling process.... In order to use a DBMS, the information structure must be mapped to the logical structure of the system (application). The mapping is expressed in a DDL (Data Description Language). The instances of data (the data base) are stored by the DBMS to conform to this logical structure." J. P. Fry and E. H. Sibley, already cited.

pendence is achieved, programs do not have to be revised to reflect shifts or expansions in the data base structure. Also, new programs need not be concerned with how other programs have chosen to organize the data.

- *Data currency.* Data values change. Ease of change and documentation of change histories are major objectives of DBMS's.
- *Data security.* Since the computerized data base of an organization is usually vital to its ongoing operations, DBMS's incorporate facilities that provide for reconstruction of the data base in the event of loss due to failures of hardware or power supply.
- *Data privacy.* It is generally accepted that a comprehensive DBMS will provide "locks" to protect proprietary or otherwise private information from unauthorized access.
- *Data relations.* DBMS's are powerful to the degree that they provide for efficient retrieval of data through complex relations among data elements. When data can be stored in a manner that reflects their interrelations, it is easier for a user to construct a new report, model interface or a new analysis.

Most DBMS's achieve the objectives of minimizing data redundancy and providing a substantial degree of data independence. Further, most provide efficient facilities for updating and for reconstructing a data base. Also, most current DBMS's contain limited features for data protection. DBMS's differ primarily in the kind of logical data structures they can accommodate, the details of the software by which they allocate data to physical storage, and the efficiency with which they access and retrieve data. Generally, DBMS offers sufficient capabilities to support a model-oriented data base. For this purpose, especially important features of DBMS design are those which support and facilitate definition* of data elements and their structures and those which aid data integration** and audit trail.

The prospect of using a DBMS to support model application at CAA led to the formulation of the integrated system framework for operation portrayed in Figure 3. Although not discussed to this point, the data dictionary represents an essential ingredient in an organizational data base system. Generally, the dictionary provides names, lengths, representations and descriptions of all data items. (This feature will be covered in more detail in the next section.)

Diagrammatically, the chart in Figure 3 coalesces the principal managerial and technical components of the CAA Model Data Base System.

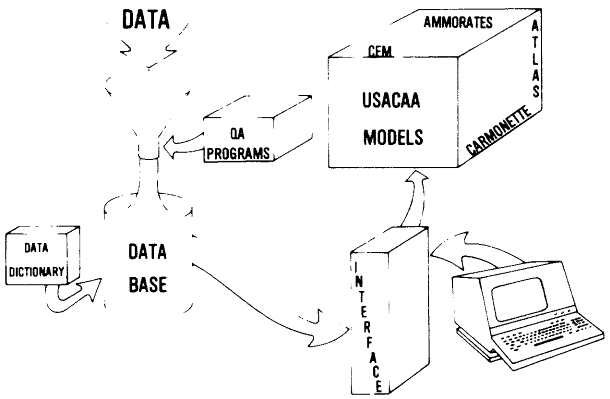


Figure 3—Data base operation

- The Data Dictionary is an instrument for central storage and maintenance of descriptive, user-oriented information concerning the format and contents of the data base.
- The Data Base is a central repository containing input data for selected models. All data transactions are managed by DBMS.
- QA programs consist of editing and mathematical/statistical routines to screen and analyze source data.
- The data base interface module, including a Cathode Bay Tube (CRT) user terminal device, supports a man-machine dialogue between the model user/analyst and the tools at his disposal—data base (through DBMS) and combat simulation models.

This operational scheme has existed at CAA in prototype form and in limited production mode for several months using weapon data for the Tank Antitank Simulation (TATS) Model of the AMMORATES family. The prototype system is operational on CAA's UNIVAC Model 1108 computing system. The DBMS is UNIVAC's Data Management System (DMS-1100). The design features of DMS-1100 adhere strongly to CODASYL conventions. This DBMS readily accommodates intricate network structures required for efficient handling of complex model data inputs. The man-machine-system link is effected by means of an interactive conversational interface.

The following section of this paper describes technical and operational features and characteristics of the data base system in development at CAA.

CAA DATA BASE SYSTEM—TECHNICAL AND OPERATIONAL CHARACTERISTICS

The CAA Model Data Base System consists of three major portions: the Data Base (repository) under DMS-1100, level 6; Data Base Utility and Service Routines; and Model Interfaces. Figure 4 provides a general structural view of the system.

The physical storage medium for the system consists of removable magnetic disk packs managed by CAA's com-

* ". . . generally consists of a statement of the names of elements, their properties (such as character and numerical type), and their relationship to other elements (including complex groupings) which make up the data base. The data definition of a specific data base is often called a schema." J. P. Fry and E. H. Sibley, already cited.

** Explicit expression (in the data base) of relationships among data. *Ian Palmer, Database Systems: A Practical Reference*, (London, England, CACI) 1975. (Data are interrelated or linked together at lowest (element) level.)

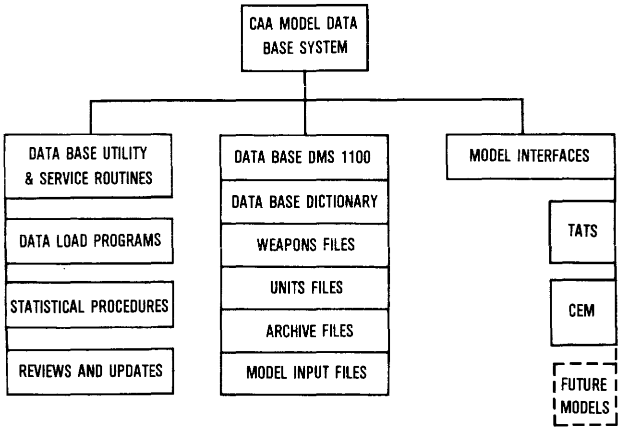


Figure 4—An overview of the CAA model data base system

puter center. Principal ADP support at CAA is provided by a UNIVAC 1108 computing system with four banks of magnetic core storage (provides approximately 170K words of usable main memory).† On-line secondary storage is achieved through ten removable disk pack drives and four fixed packs. Users may interface with the system through batch (via main card reader or one of three remote job entry devices) or demand (five remote alphanumeric CRT consoles and three graphic display consoles) modes.

Various utility and service routines allow loading, modification, retrieval, inspection and display of the data base. Also, they give the capability of performing various mathematical/statistical procedures on the contents of the data base. The model interfaces can be invoked by the user to access the data base in order to create an input file for execution of a particular model.

The CAA Model Data Base may be broadly, and informally, subdivided into five interrelated segments: the Data Base Dictionary, the Weapons Files, the Units Files, the Archives Files and the Model Input Files. All files are integrated (logically interrelated); their structure is defined under one schema and controlled by the DMS-1100 system.

The Data Base Dictionary. As mentioned previously in this paper, a data base dictionary‡ is a user tool which contains essential descriptive information for data base management and application. The CAA Data Base Dictionary also includes the data item's source, frequency of update, acceptable range of values and other items of importance. The Data Base Dictionary can be used by the analyst to ascertain what data is available to him in the system. Further, the dictionary indicates to programmer/analyst the format and the variables. Currently, there are over 150 data items described in the dictionary, defining each model input variable contained in the weapons and units files.

The Weapons Files. For each weapon in the system, there exists a Basic Weapon Record. The Basic Weapon Record is common to models for which weapon data is to be stored. The data items in this record contain (1) data for identification of a given weapon (e.g., generic nomenclature, Line Item Number, Federal Stock Number); (2) numeric values for variables which are common to two or more models (e.g., crew size, maximum effective range, rate of fire); and (3) identification data and values of variables (for certain models) which are so few in number that it would be inappropriate to form a separate record. If data for a specific model is to be maintained in the data base, a unique model record is established for a given weapon for that model (e.g., CEM record, CARMONETTE record, TATS record). Data in a model-specific record pertain only to the identified model. Thus, to support the simulated play of a given weapon in a given model, only two records on the data base need be accessed: the Basic Weapon Record and the Model Record. Figure 5 illustrates the Basic Weapon Record and the Model Record. Figure 5 illustrates the Basic Weapon Record-Model Record arrangement for data base application. Currently, data on 78 weapons are contained in the weapons files; data for many of these weapons are described for two or more models.

The Units Files. Six different types of records exist in the Units Files: Force, Army, Corps, Division, Brigade and Battalion Records. These records are hierarchical in nature, with the Battalion Record being the basic building block. In addition to basic unit description data (e.g., Unit name, TOE strength), each record holds information required by applicable models to simulate combat activity of the units (e.g., indices of firepower potential, fuel consumption rate). Also included are the type and number of subordinate

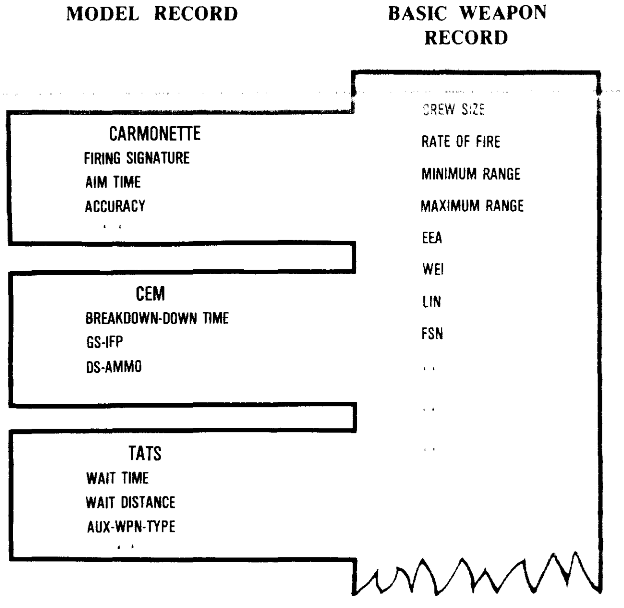


Figure 5—Model record

† DMS-1100 currently requires about 32K (36-bit) words of main (core) memory. System enhancements such as a Query Language Processor are expected to increase memory needs substantially.

‡ "A documentation of the types of data units in the data base or available for inclusion in the data base in terms of their definition, purpose, controls, formats, relationships with other data units, and other properties relevant to data base design and application development." Ian Palmer, *Data Base Systems: A Practical Reference*, (London, England, CACI) 1975.

elements. Thus, inspection of a particular Division Record will reveal various subordinate elements (brigades and battalions), each of which is described separately in appropriate unit records. The Battalion Record is the only unit record which does not contain pointers, or connecting logical linkages, to other records in the Units Files. The Battalion Record, in addition to containing unit descriptors, carries a table of component weapons and quantities. These weapon identifiers are logically connected to data records for weapons described in the Weapons Files. On occasion, a Battalion Record may be used to describe a unit other than a battalion. For example, a cavalry troop supporting a division would be described using a battalion record. In general, all units are composed of aggregations of other unit types with the exception of battalions which are composed of aggregations of weapons.

Archives Files. The Archives Files segment of the CAA Model Data Base is designed to contain a full set of information related either to basic changes in data base contents or to past model operations. When a user/analyst accesses the data base (through special service or utility routines) and changes the stored value of a specific variable, a transaction record is automatically generated for audit trail purposes. This operation is highly disciplined by means of security access procedures. Contents of the transaction audit record—including analyst identifier, date, variable identifier, old value, new value, reason for change—are placed in the Archives Files. For model operations, the Archives Files contain changes to input data used for execution of a given model relative to the data values in the data base (e.g., firepower data changes made only for model sensitivity testing).

The Model Input Files. Consider the case of a user/analyst constructing input data for model operation. This work proceeds by means of the model interface module. Once basic data modifications are completed by the user, two separate operational files are automatically created by the Model Data Base System—one external and one internal. The external file is in a form acceptable to the model and can be immediately used by the model to execute a simulation. The internal file contains a copy of all the model data, but is retained internally within the system for future reference. This internal copy of model inputs may be used as a base for constructing data for simulation excursions and as a historical record of the inputs used for the simulation.

CEM Interface. As indicated in the preceding section, an interface has been implemented to support the Tank Antitank System (TATS) Model. The success of this effort furnished "proof-of-principle" of an interactive data base system to prepare model inputs for execution of a selected model. With the acceptance of the TATS interface by CAA model users, work commenced on development of a broader, more complex interface to create the full range of inputs required for the Concepts Evaluation Model (CEM), the prime theater level combat model currently used at CAA. CEM has the capability of simulating a theater level conflict with 70 Blue divisions fighting 125 Red divisions. The original input scheme for CEM involved a file of approximately 3000 card images. In one typical game, 1600 cards were used to describe the Blue force structure, weapons systems, unit arrival schedule and logistical support activities.

An implementation plan was developed to build in-house a CEM interface to give the user the capability interactively to construct from the CAA Data Base the inputs required to drive the model. The first part of the man-system interface addresses Blue force units; with minor changes, the design package can be adapted for the Red unit portion. When a user at a terminal accesses the system, he will be presented a list of forces whose data is already available for simulation purposes. Subsequently, he may select any of these as a starting point or he may elect to construct his force structure from individual unit data. Alternatively, a combination of these paths is available. The primary restriction placed on the user is that any data for a weapon system or maneuver battalion he chooses must be in the data base. However, this is not prohibitive since either of these sets of information may be loaded into the data base in about fifteen minutes using special utility routines in demand or batch mode. At any level of force definition, if he selects a unit whose data is already in the data base, that unit will be retrieved and all subordinates automatically incorporated into his structure. For example, an entire corps force could be immediately assembled from already defined unit structures. As he makes modifications from a previous force, appropriate audit trail entries will be made automatically. When he is satisfied with the structure of his forces, and when the package has insured the force's internal consistency and correctness to meet CEM requirements/constraints, the weapons associated with the force are retrieved from the data base and the entire units portion is produced for simulation activity.

For computer-based implementation, the interface system is being written in FORTRAN V and accesses the data base through CAA subroutines. System design is being conducted with a top-down, structured approach. The computer program code is being written for maximum syntactical correctness for either Field Data of ASCII FORTRAN. Ease of maintenance is an explicit design goal. Currently, the package has approximately 3000 lines of operational code in 74 subroutines (structured modules). The interface now uses approximately 28K words of main (core) memory and is expected to require about 30K when complete—without overlays. About 8000 words of this total are taken up by the various tables which are constructed to define the force and drive the card image production. The project was begun in early September with three OR analysts working full time. Expected completion of the units portion is in December. Since this part of the interface represents about 80 percent of the volume of data required for CEM execution and contains the data elements most susceptible to change for successive simulation excursions or production runs, its completion constitutes a major achievement in the CAA QA program.

Statistical Procedures for Data Base Analysis. A composite, integrated data base offers a fertile area for work in data verification using statistical techniques to support analysis of model input data values. As an initial step in this type of effort at CAA, a set of computer programs was written to select parametrically specified combinations of variables related to selected types of weapons. Subsequently, the selected data is placed in a form for input to automated statistical routines currently in use at the agency. A particularly powerful set of data analysis capabilities is embodied in a combination of computer graphics routines and mathematical/statistical utilities. Figure 6 shows illustrative results of the use of a two-dimensional graphical display generated automatically with the CAA Multi-optioned Interactive Data Analysis System (MIDAS) in conjunction with regression analysis routines. The weapons max-range-vs-caliber plot and regression parameters provide a visible analytic base for examination of data values individually and as a set based upon common characteristics. Figure 7 depicts sample results using 3-D graphical plots and numerical taxonomy-clustering routines for weapons analysis based upon max-range, caliber and rate-of-fire relationships.

Other current or planned data analysis capabilities include use of regression equations to predict update values prior to receipt and the use of confidence intervals to assist in detection of questionable values for scrutiny and validation.

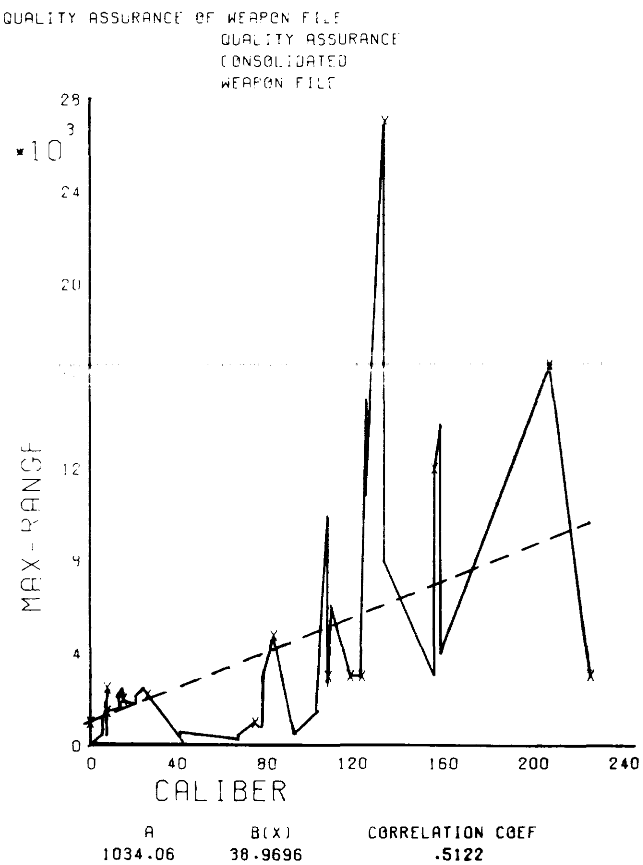


Figure 6—Sample two dimensional graphical display

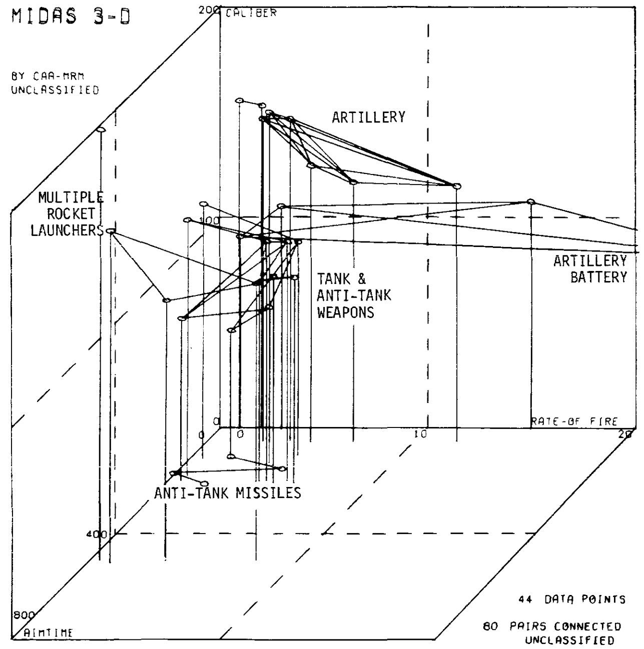


Figure 7—Sample three demensional graphical display

CONCLUSION

The use of a command-wide QA Model Data Base Management System offers many advantages and opportunities in providing improved user information for major Army studies and analyses. Data validity, analytic quality and operational efficiencies are objectives whose achievement can be facilitated by such a system. Figure 8 highlights a range of opportunities potentially provided by an operational data base of the nature described in this paper. By blending the principles, methodologies and procedures of OR/SA, com-

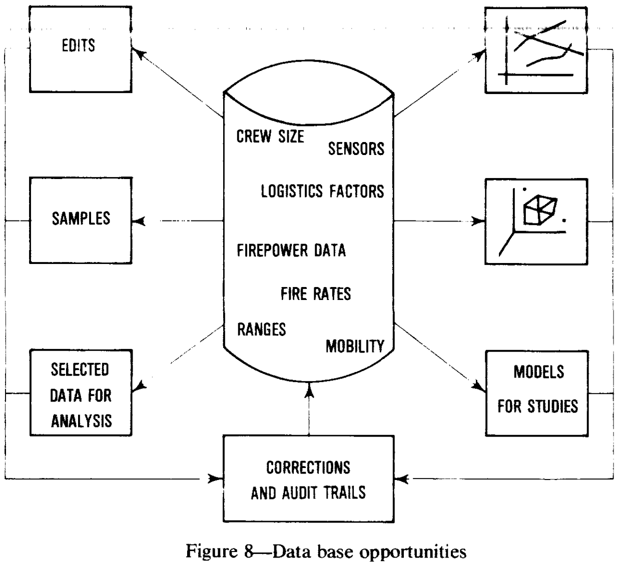


Figure 8—Data base opportunities

puter science, emerging data base technology, mathematics/statistics and ADP management, the data base component of the organizational QA program will contribute a necessary foundation element integral to the CAA model-supported study environment.

REFERENCES

1. Bayse, William A. and J. F. Henry, "Quality Assurance," *Proceedings: AORS XIV, Vol I,* US Army Materiel Development and Readiness Command, Alexandria, VA, 1975.
2. Canning, R. G., "The Cautious Path to a Data Base," *EDP Analyzer,* Vol 11, No 6, June 1973.
3. Canning, R. G., "A Structure for EDP Projects," *EDP Analyzer,* Vol 11, No 5, May 1973.
4. Chamberlain, R. G., "Conventions for Interactive Computer Programs," *Interfaces,* Vol VI, No 1, The Institute of Management Sciences, November 1975.
5. Centec Consultants, Inc., "CAA Data Base System Requirements—Intermediate and Long-range," Centec Consultants, Inc., Reston, VA, June 1976.
6. Davidoff, A. E., *Concept of Operations for a CAA Data Base,* CAA-SP-76-1, US Army Concepts Analysis Agency (CAA), Bethesda, MD, January 1976.
7. Fry, J. P. and E. H. Sibley, "Evolution of Data-Base Management Systems," *ACM Computing Surveys,* Vol 8, No 1, March 1976.
8. General Research Corporation, *CARMONETTE* (Model Documentation), CAA-D-74-11, US Army Concepts Analysis Agency (CAA), Bethesda, MD, November 1974.
9. General Research Corporation, *CONAF Evaluation Model* IV (Model Documentation), GRC, McLean, VA, December 1974.
10. Martin, James, *Computer Data-Base Organization,* Prentice-Hall, Inc. Englewood Cliffs, NJ, 1975.
11. McGowan, C. L. and J. R. Kelly, *Top-Down Structured Programming Techniques,* Petrocelli/Charter, New York, NY, 1975.
12. Nolan, R. L., "Computer Data Bases: the Future is Now," *Harvard Business Review,* September-October 1973.
13. Nussbaum, D. A., *Multivariate Analysis System,* CAA-D-76-4, US Army Concepts Analysis Agency (CAA), Bethesda, MD, June 1976.
14. Palmer, Ian, *Data Base Technology: A Practical Reference,* CACI, London, England, 1975.
15. *Report of CODASYL Data Base Task Group, April 1971,* Association for Computing Machinery, New York, NY, 1971.
16. Sperry UNIVAC, *Data Management System (DMS-1100), Schema Definition* (Data Administrator Reference) (UP-7907 Rev 2), Sperry Rand Corp, 1974.
17. Sperry UNIVAC, *Data Management System (DMS-1100), American National Standard COBOL (Fieldata) Data Manipulation Language* (Programmer Reference) (UP-7908 Rev 1), Sperry Rand Corp, 1974.
18. US Army Concepts Analysis Agency, *Tabulation of Models of Interest to CAA,* CAA, Bethesda, MD, July 1976.

Routing and control in a centrally directed network

by JOSEPH RINDE

Tymshare Inc.
Cupertino, California

ABSTRACT

TYMNET I is a centrally directed network with over 200 nodes interconnected in a topology that allows alternate paths between nodes in the network. Routing within the network is done by a central supervisor program, with full knowledge of network topology and network load. Within network nodes routing is table driven.

The supervisor communicates with nodes through a command tree that is built at network takeover time. The supervisor has no a priori knowledge of the network topology when it starts network takeover, and the topology may change while the supervisor is in control. The control tree is dynamically modified by the supervisor to accommodate the new topology.

INTRODUCTION

In 1969 Tymshare Inc. started the development of TYMNET I* to supply the communications needs of its growing time-sharing market. The design objectives were to replace hardware multiplexing gear with a more reliable and more versatile communications facility. Beyond these functional requirements, since it was developed to meet the demands of a commercial environment, the network also had to be inexpensive to implement and operate.

A centrally directed network of Varian 620 mini-computers was developed. When the network became fully operational in 1971 it consisted of 30 nodes and a central supervisor (with three backup supervisors) that ran on an SDS 940 computer system. The network has since undergone a prolonged evolution and has grown to 200 nodes (see Figure 1).

ROUTING

TYMNET I has been described as a virtual circuit switched network[3] which includes ARPANET type packets as a subset of its data grouping capabilities.[2] A virtual circuit is bidirectional and ties up some memory in the nodes that comprise the circuit, but line bandwidth is utilized only when user data is flowing through the network.

Routing within the nodes is done implicitly through routing tables called permuter tables. Figure 2 shows a virtual circuit in TYMNET I with all its associated permuter table entries and buffer assignments. Each permuter table entry points to one buffer of a buffer pair. One buffer of the pair is for incoming characters, one for outgoing characters.

Each physical record (analogous to a packet) traveling between nodes is a collection of logical records, each of which is associated with a virtual circuit. This allows the physical record overhead to be distributed over the data of multiple users. As seen in Figure 3, each logical record has a header specifying its logical record number (used to index the permuter table) and a count of the data bytes contained in the logical record. To describe the routing of data in the logical records we define

link	a connection between two adjacent nodes
P(i,j)	the jth entry in the permuter table for link i
[P(i,j)]	the contents of P(i,j) (a buffer number)
O([P(i,j)])	the other buffer number of the buffer pair

A physical record arriving on link A containing a logical record X would cause the characters in logical record X to be placed in the buffer [P(A,X)]. Note that the physical record processor need have no knowledge of the final destination of the data in a logical record. Even knowledge about the link on which these characters will leave is not explicitly known. P(A,X) fully defines the path the data is to follow, because (1) the buffer [P(A,X)] is a port buffer, causing the characters in it to be processed by a port driver (this could be a host port or a terminal port), or (2) there exists a P(B,Y) for some B different from A (though X and Y may be equal) such that [P(B,Y)]=O([P(A,X)]). When the physical record-making process runs for link B it scans P(B,i), i=0, . . . , n. Each P(B,i) points to a buffer; the

* TYMNET is a registered trademark of TYMSHARE Inc.

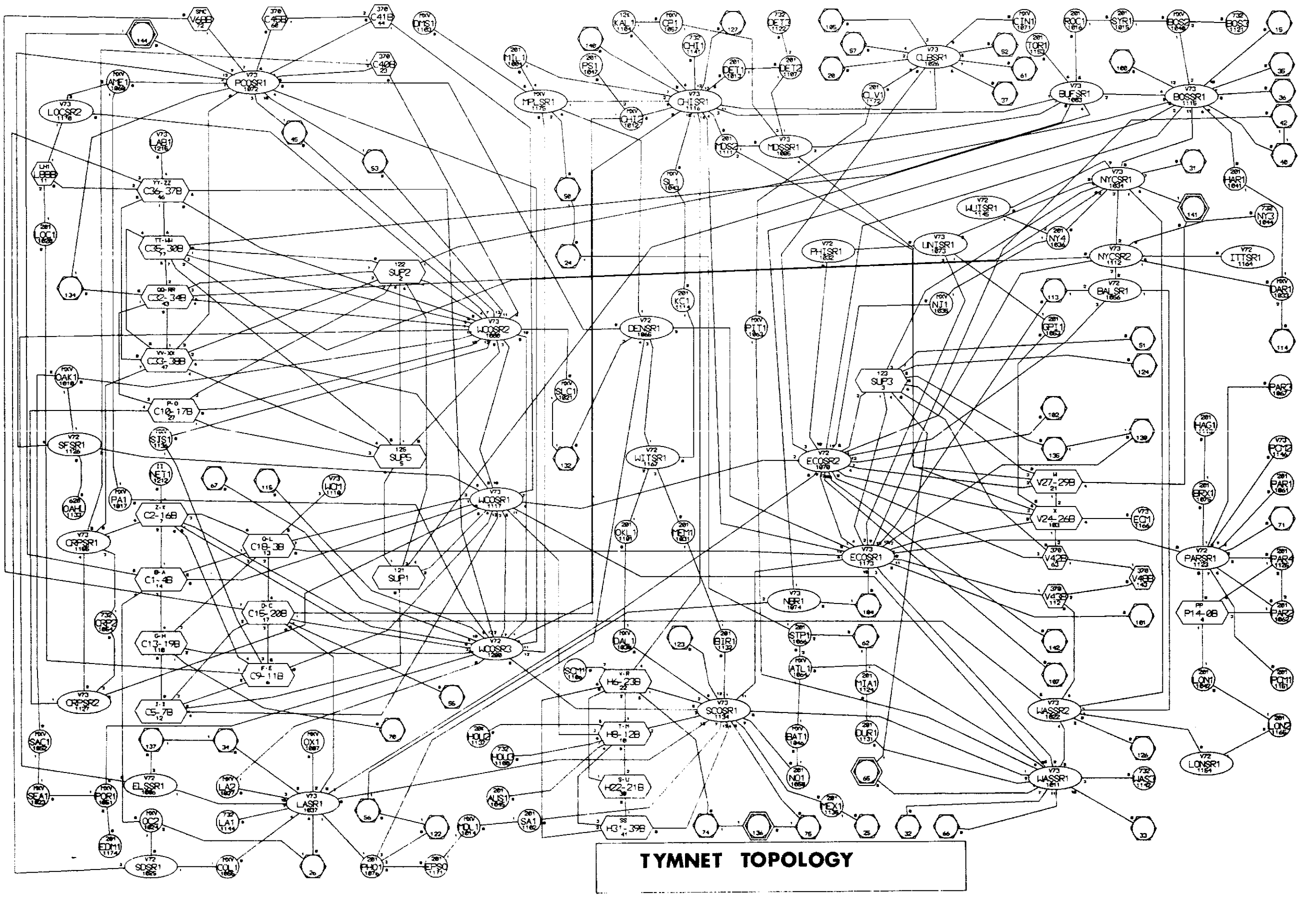


Figure 1—TYMNET topology

other buffer of the pair (i.e. O([P(B,i)])) is checked for a non-empty condition. This condition will be satisfied by P(B,Y) causing logical record Y to be created with the data in the buffer [P(A,X)].

It is possible for data for a given virtual circuit to arrive at a node in two separate physical records, but leave that node in one physical record. This can happen since both arriving physical records will have a logical record X, and the data in both logical records will be placed in buffer [P(A,X)]. It is also possible that data arriving in a single physical record will leave in two physical records. This is why we speak of a flow of characters in TYMNET I, rather than packet switching.

A physical record is a collection of logical records plus a physical record header (Figure 4). A cyclic record numbering scheme is used to facilitate acknowledgment of physical records correctly received and to detect and retransmit records which arrived incorrectly. Sixteen bits of vertical checksum and sixteen bits of diagonal checksum are used to check that records crossing a link are correct.

ESTABLISHING A CONNECTION

A user connects his terminal to a host on the network by dialing a local node. After typing a character to identify his terminal characteristics, the user enters his user name, host number and password. This information is sent, by the local node, to the network supervisor. The supervisor verifies the user's name in the Master User Directory (MUD) and then checks for a correct password. (Passwords are stored in the MUD only in ciphered form. To check a password given at login, it is enciphered and compared to the cipher in the MUD.) If the user did not specify a host number, his standard host number is taken from the MUD.

The supervisor now knows the node which the user called (origination) and the node to which he wishes a connection (destination). The supervisor then computes the minimum cost path for the virtual circuit.[1] Each link in the network has a cost associated with it. This cost is a function of the link bandwidth and the presence of overload conditions (Table I).

TABLE I—Link Costs in TYMNET I

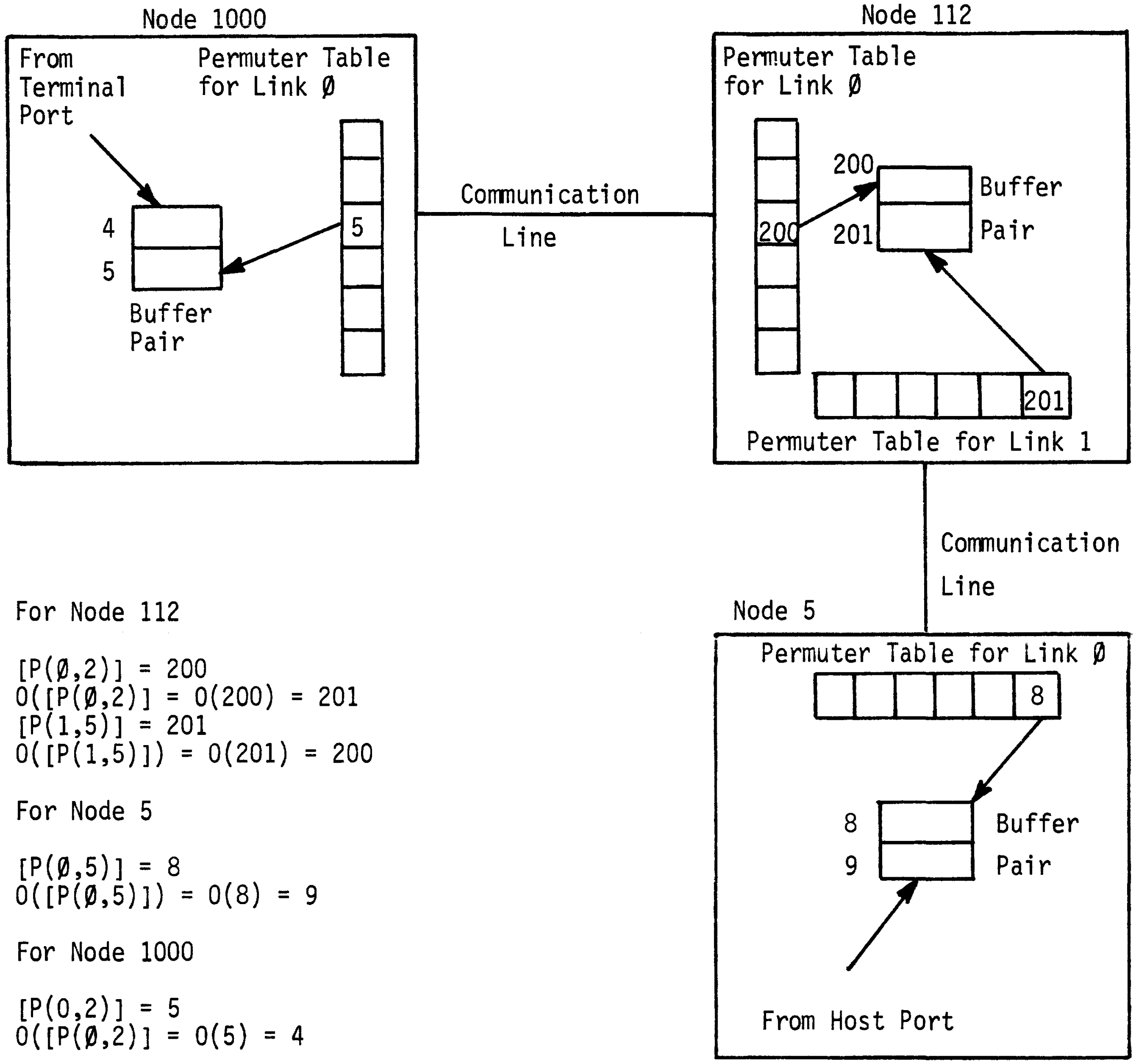
Line speed	cost normal	cost overloaded one way	cost overloaded both ways
9600 bps	10	26	42
7200 bps	11	27	43
4800 bps	12	28	44
2400 bps	16	32	48

Thus the minimum cost path, as computed by the supervisor, is an optimization of network resources, rather than a dollar cost. Once a path has been plotted, the supervisor allocates buffer pairs and permuter table positions in each node on the path to create the virtual circuit. Messages are sent to all nodes along the path to make the appropriate permuter table entries. This implicitly causes buffer assignments in the nodes. Nodes send an acknowledgment to the supervisor after making a permuter table entry. Once all the acknowledgments are in, the supervisor sends the user name, the user's status (from the MUD), the originating node number and port number, and the terminal characteristics to the destination node, plus a message to tell the attached host that there is a new login.

The host may now read the user name to verify that this user name is valid on this host. No further checking is required! All the security checking has been done on a host, with access restricted to network personnel, where not even the passwords are vulnerable to theft. This allows a host with minimal login security to be connected to the network, with confidence that only authorized people may log in through the network. Additional login security may, of course, be imposed by the host computer system.

NETWORK CONTROL

The network supervisor is a program that runs under a special time sharing system on an Interdata 7/32. The



```
For Node 112

[P(Ø,2)] = 200
O([P(Ø,2)] = O(200) = 201
[P(1,5)] = 201
O([P(1,5)]) = O(201) = 200

For Node 5

[P(Ø,5)] = 8
O([P(Ø,5)]) = O(8) = 9

For Node 1000

[P(0,2)] = 5
O([P(Ø,2)] = O(5) = 4
```

Figure 2—A virtual circuit in TYMNET I

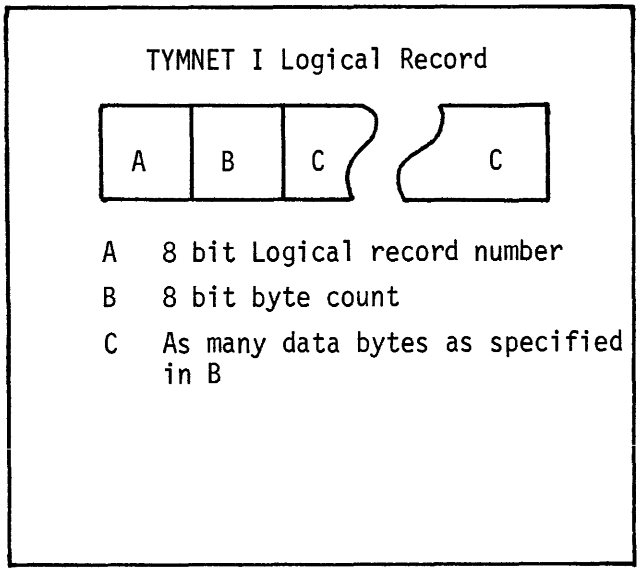

TYMNET I Logical Record
A
B
C
C
A 8 bit Logical record number
B 8 bit byte count
C As many data bytes as specified in B

Figure 3—TYMNET I logical record

supervisor, like any piece of software, and the 7/32, like any piece of hardware, are subject to failure. Although failures (hardware and software) are infrequent (on the order of one every three weeks) the absence of a supervisor to build virtual circuits cannot be tolerated for very long. To deal with this problem, four potential supervisors exist in the network, only one of which is active at any one time. The active supervisor keeps the other supervisors dormant by sending "sleeping pills" to them at regular intervals. If the active supervisor fails, the operators at the network control center can immediately awaken one of the dormant supervisors. Even without human intervention, the dormant supervisors will notice the absence of the active supervisor when they cease to receive "sleeping pills." The various supervisors have staggered sleep times, at the end of which they will awaken if no sleeping pills have arrived. Thus, in case of a supervisor failure one of the dormant supervisors will awaken and take control of the network. It is possible for multiple supervisors to be trying to take over the network simultaneously. This situation is resolved gracefully by the less dominant supervisor going to sleep when it discovers the presence of a more dominant supervisor.

NETWORK TAKEOVER

In order to control nodes and carry out the supervisor's function of building virtual circuits, the supervisor must know the capacity of all nodes, their link capacities, the network topology and the value of every permuter table entry in the network.

A supervisor starting network takeover has no a priori knowledge of the network topology. The supervisor first sends a takeover command to its own node, and learns of that node's capacity, the capacity of its links, every permuter table entry in that node and the neighbors of that node on each link. The latter is the basis on which the supervisor discovers the topology of the network. The supervisor now sends takeover commands to each neighbor of its own node. As each of these nodes comes under complete control of the supervisor, each of its neighbors is checked to discover previously unknown nodes, and these are in turn taken over.

In this way the supervisor learns the topology of the network, all its capacities and (from the permuter tables) which resources are in use. The network takeover duration is approximately three minutes. The time determining factors are the number of nodes, the bandwidth of links in the vicinity of the supervisor, the number of permuter table entries and the connectivity of the network vis-a-vis the supervisor's own node. From a control standpoint the supervisor views the network as a tree. The control tree comprises a subset of the links in the network. The balance and depth of this tree is the measure of connectivity of the network in relation to the supervisor's own node.

If a previously non-operational link becomes operational it may reveal one or more nodes that were previously inaccessible. The supervisor would then extend the control tree by taking over these newly discovered nodes. On the other hand, if a link that is part of the control tree goes out, the supervisor loses control of the nodes in that subtree. The control tree is then rebuilt to regain control of all the lost nodes that are still accessible through the network topology.

In a large network like TYMNET I, good connectivity is important to maintain fast response time and minimize bandwidth overhead. A deep supervisory control tree will cause undesirable delay in delivery of supervisory commands. Even independent of the centralized control proper-

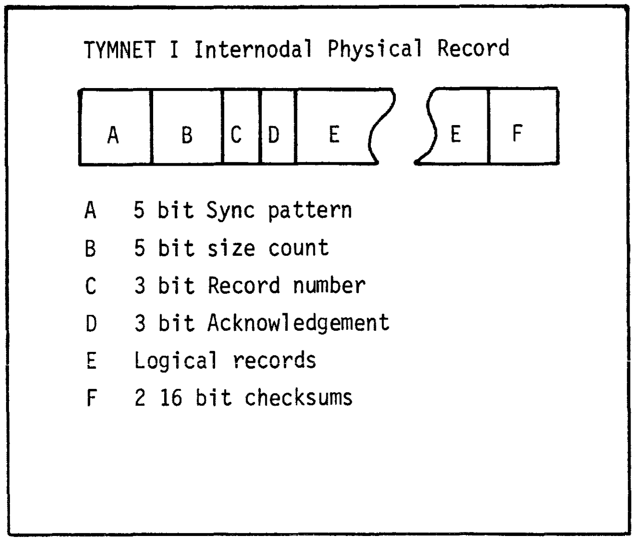


Figure 4—TYMNET I internodal physical record

ties of TYMNET I, a loosely connected network increases user response time by increasing the network transit delay. This added network delay time has been observed in the ARPANET[4] and is due to a deficiency of links in the network as a whole and to the limit on the number of links each node may have. The physical topology of TYMNET I is based on a topology generating program which employs simulations using accounting data to determine virtual circuit lengths and telephone costs. The average TYMNET I virtual circuit is 3.1 links long.

The original TYMNET I nodes allowed only three links per node. Presently TYMNET I nodes allow 16 links per node. The cost of more links is increased CPU and memory requirements for the node. However, even 16 links per node will soon be inadequate for the growth of TYMNET. One solution to this problem is the use of node clusters. A node cluster is a group of two or more nodes in close proximity, interconnected by an inexpensive high speed distributed memory transfer device. We call this device a "memory shuffler," and it is capable of data transfers at memory bandwidth rates. We plan a fully interconnected eight node cluster at TYMNET's Cupertino center, with 72 links to the rest of the network. This approach creates a logically very large node (including two supervisor machines) that does not change the fundamental logical structure of the network. (A connection through the memory shuffler is viewed by the node and the supervisor as a high bandwidth link.) With clusters at all three major TYMNET centers in the United States, the network can be kept well connected (average virtual circuit length of less than two links plus a memory shuffler link, and network takeover time of one to two minutes) for the foreseeable future. A prototype of the memory shuffler is already deployed in the network.

Eventually, network growth will reach a point where further schemes to improve connectivity will yield small returns. At that time a partitioned topology (similar to the telephone area code) will become necessary. Little work has been done on this since no existing computer network has reached a size requiring a partitioned topology. Using TYMNET's centralized control approach, each area can have its own supervisor. There could be a master supervisor to act as the focal point of all interarea communication. Alternatively, each area supervisor could communicate with its neighboring area supervisors, and interarea routing could be done by an adaptive routing scheme (this is not unreasaonable if the number of areas remains small).

REFLECTIONS

The underlying principles of TYMNET have passed the test of time. The orientation of the network to terminal users, and specifically to the support of the full duplex terminal, has given TYMNET a great deal of flexibility. Full duplex is more than simultaneous bidirectional transmission. It implies character by character interaction and full echo control. These features are an integral part of the TYMNET design (e.g., logical records of one or more characters).[3] The addition of host to host communication required only a subset of the facilities available (e.g., echo control is not needed). Other high speed communications requirements have also been accommodated with low overhead.[2]

Since TYMNET I interfaces to a wide variety of terminal types, a host connected to TYMNET I can similarly be accessed by a wide variety of terminals, even though the host may only be able to communicate directly with one type of terminal. In this sense TYMNET I is a more effective means (independent of cost) of connecting terminals to hosts than direct dial telephone service. TYMNET I's ability to translate between character sets, so useful in the case of accommodating a variety of terminals, can be turned off to permit shipment of pure binary data through the network.

The success of TYMNET's centralized approach can be measured, in part, by the routing overhead (worst case 1.25 percent of a 9600bps link[2]) and the accessibility, by terminal users, of all of TYMSHARE's hosts (27 SDS940s, 10 PDP10s and 4 IBM370s) solely through the network. This relieves the hosts of password checking responsibilities, and reduces capital investment by having only one communications interface for terminals. Another feature of centralized control is a network clock, kept by the supervisor. The clock is a radio receiver for station WWVB, a time signal broadcast by the National Bureau of Standards. Each host is informed of the current time when it comes up, keeping all hosts synchronized (a distinct advantage for accounting). The central supervisor is not only a good source of information, but provides a natural collection point for network diagnostics and network accounting data.

The evolution of TYMNET I from a 30 node private network to a 200 node value added carrier is an example of successful adaptation of new technology to expand a general design beyond its original goals. However, the design of TYMNET I reflects the technology available in 1969. Today's lower CPU and memory costs remove the justification of some original design decisions.

As the network grew, the technology of mini-computers advanced, and prices came down, more powerful CPUs were brought into the network and new node types were developed with more capacity, in both CPU and memory. The additional capacity was used to provide more links per node and more ports per node. More terminal types became supported (e.g. 120cps terminals and batch terminals of the 2780 and 3780 type).

In 1975 the supervisor running on the SDS 940 reached its capacity and was replaced by a supervisor running on an Interdata 7/32, reducing network takeover time from 15 minutes to 2.5 minutes. Network takeover time is currently limited by bandwidth and connectivity of the network.

The original TYMNET I nodes had 8K of memory, a limiting factor on the capabilities of the nodes. This alone required the bulk of decision making to be vested in the network supervisor (e.g., allocation of buffer and permuter table entries in the nodes). A direct consequence of this decision was the supervisor's need to have a copy of every permuter table entry in the network. With network growth, this has proven expensive in both supervisor memory requirements and the length of network takeover. All the evolution and growth of the last five years has not changed that basic division of decision making between the supervisor and the nodes. This division is no longer in tune with the available technology, but changing it requires such a fundamental change in the implementation of both the supervisor and the nodes as to necessitate an ongoing redesign of both. The external view of the network need not change with this redesign, however. As a result the deployment of TYMNET II will occur in 1977 without disruption of service to users of the network.

ACKNOWLEDGMENTS

The author owes a great debt to LaRoy Tymes, the designer of TYMNET.

REFERENCES

1. Rajaraman, A., "Routing in TYMNET," submitted for publication.
2. Rinde, J., "TYMNET I—An alternative to packet switching technology," *Third International Conference on Computer Communications,* August, 1976.
3. Tymes, L., "TYMNET—A Terminal Oriented Communications Network," *AFIPS Conference Proceedings,* Vol. 38, spring 1971.
4. Walden, D. C., "Experiences in Building, Operating, and Using the ARPA Network," *Second USA/Japan Computer Conference* August, 1975.

TYMNET as a multiplexed packet network

by JOHN KOPF
Tymshare, Inc.
Cupertino, California

ABSTRACT

TYMNET is a commercially successful computer network that has been in continuous operation since November, 1971. It contains over 200 nodes. Circuit routing is established by a central process, the supervisor.

TYMNET has never been fully documented in the literature. While similar to other packet networks, the mechanisms used by TYMNET differ significantly from those used in other documented networks, such as ARPANET. Packets are used, but only as the carriers for virtual channels between neighboring nodes, and are not themselves "switched." The mechanisms used lead to good line utilization, especially for individual, small-volume transmissions. Flow-control mechanisms permit information sources and sinks to operate at different intrinsic speeds.

INTRODUCTION

TYMNET* is a communications network which was originally designed by Tymshare to handle it's own communication needs, and is now operated by Tymnet, Inc. (a wholly owned subsidiary) as a common carrier service. TYMNET has been in service since 1971, during which time it has expanded from the initial 30-node network[1,2] to the current 200 nodes.[3] On the basis of both experience with TYMNET and new technological developments, TYMNET is currently undergoing a transition to a new, even more flexible network—TYMNET-II. TYMNET-I has never been adequately documented in the literature.

TYMNET has been described as a virtual circuit switching network, without setting forth the techniques used. In order to discuss these techniques, it is first necessary to give a general overview of TYMNET.

THE PARTS OF TYMNET

TYMNET consists of a large number of NODES interconnected by LINES. Only one line can connect two nodes. Each node has a description of the lines and neighbor nodes it has, as well as a local (partial) description of the circuits passing through or terminating at that node. The information in all nodes is the PRIMARY description of the network. TYMNET also has a SUPERVISOR, whose function is described below.

A node consists of a mini-computer, some simple interface circuitry, and the node-code. Nodes are classified by function. The function defines the interface, both in terms of circuitry, and also in terms of the code (in TYMNET, the interface circuitry is deliberately kept primitive, and the functional interface is built into the code).

The node-code can be functionally partitioned into several distinct modules:

a. Buffer management;
b. Supervisor communications;
c. Line interface;
d. One or more process interfaces.

One type of node is the TYMSAT, which contains the interface used for low-speed dial-up (asynchronous). It may also include an interface for high-speed access (bisync). A second type is the BASE, which contains the interfaces to TYMSHARE's XDS-940, PDP-10, and IBM-370 computers. A third type is the TYMCOM, which is a base in function but interfaces to a broader variety of HOST computers.

The supervisor is a program which maintains control over the network and manages the network resources. The supervisor originally ran in a XDS-940, connected to the network through one of the bases, but has recently been recoded to run in an INTERDATA 7/32, as one of the nodes in the network. There are several nodes in the network which are capable of running the supervisor, but only one supervisor controls the network at any given time.

When a supervisor becomes active, it has no knowledge of the topology of the network, or of any of the circuits active within the network. Its first activity is therefore to take over the network. This consists of taking over its node, determining the neighbors of that node, then taking over each of the neighbors, and iterating until there are no nodes left which have not been taken over. During this period, the supervisor also determines and records the

* TYMNET is a registered trademark of TYMSHARE Inc.

resources used by each circuit active in the network at takeover time. The supervisor retains this information as a SECONDARY description of the network circuit topology, and will update this description on the basis of subsequent information volunteered by the nodes, and generated by the supervisor. If a part of the network is subsequently lost, such as by the loss of a line on the takeover path, the supervisor will retake all nodes which are still accessible through alternate paths in the network. The act of taking over a node generates a path from that node to the supervisor, so the node can communicate with the supervisor.

DATA WITHIN TYMNET

The basic unit of data is the CHARACTER or BYTE of 8 bits. A circuit within TYMNET is a construct which transfers a stream of bytes over a fixed path (and in both directions) while maintaining the order of the bytes. The CONTENT of the data bytes is transparent to the network, and is the responsibility of the interface at each end of the circuit. Because the sequence is maintained, the actual data need not really be 8-bit quantities, but may instead be anything from 1 bit to 1,000,000 or more bits—the only constraint is that the total transfer (including padding if necessary) be a multiple of 8 bits.

To contain the characters while in a node, BUFFERS are needed. In TYMNET, Buffers are assigned unique numbers, to distinguish them from each other. "BUFFER" in this context is actually a misnomer, since a buffer does not contain any data, but instead is a descriptor of where the data actually is, and how much data is there. Data bytes are actually stored in a separate area, which is allocated as the need arises, and which is freed when no longer in use. The "buffer" describes where the first byte is, how many bytes are there, and where the last byte is. In addition, a buffer discriptor also contains a pointer—unique to that circuit—which indicates which process is to take data from that buffer. By this mechanism, source and destination processes are uncoupled, and leads to complete flexibility as to circuit routing and termination. A byte is added to the buffer by appending to the last byte, and updating the buffer descriptor. A byte is removed by removing the FIRST (original) byte, and updating the descriptor.

The actual IMPLEMENTATION is as follows: Space is allocated for the buffer descriptors, buffer data area, and a linear bit array at the time the node-code is created. Primitive subroutines are provided to place and remove a byte from a specified buffer (descriptor). Each buffer descriptor includes a unique index into the linear bit array, and the routines are coded to manipulate the bit specified, turning the bit off if no data bytes are in the buffer, and on if there are one or more data bytes present.

The bit array is partitioned into areas, each of which corresponds to a given process. Each process can look at the set of bits assigned to it, to determine if there is anything to do, and if so, can pick one of the bits that is on, and translate the bit back into a buffer number. The bit order need not be in one-to-one correspondance with the buffer number order. When a buffer becomes empty, the bit goes off, and the process will ignore the buffer, until some other process places more data into it.

LINE PROTOCOL

TYMNET uses a line protocol which is essentially a variation on that used by other packet networks, but which has less overhead while providing flexibility. The normal TYMNET data packet (or physical record) has two bytes of header, from three to sixty bytes of contents, and four bytes of checksum at the end of the packet. With packets of variable length, line bandwidth is not wasted in transmitting padding or fill characters.

Each line is assigned a fixed number of channels. On a given line, a circuit is assigned to one channel, and has exclusive use of that channel for the duration of the circuit. At each end of the line, a pair of buffers have been assigned to that circuit, also fixed for the duration of that circuit. These channel and buffer assignments are made by the supervisor at the time the circuit is initially built, and each node along the circuit is informed of its portion. As a result, the line overhead is lowered, since complete routing information need not be transmitted along with each packet.

Each circuit is terminated at both ends by a connection to an interface. The interfaces include an ordered set of buffers, normally referred to as PORTS. The intermediate nodes along a circuit each have a set of buffers, assigned to PASSTHROUGH usage. Passthroughs are in addition to, and distinct from, ports. The port interface has responsibility for the data protocol (e.g., code conversion, echoing of characters to terminals, hand-shaking across the interface, etc.).

The physical packet is subdivided into one or more subpackets (logical records). Each logical record contains two characters of header, which specify the channel number, and the number of characters of data present. This permits the multiplexing of a number of short messages (containing as little as one character each) into a single packet, with consequent savings in overhead, since the checksum at the end of the packet covers all of the logical records within the packet. Individual logical records do not need the overhead of individual checksums.

If single characters are traveling through a circuit, their overhead is high (2 bytes overhead/3 bytes of logical record). However, an interesting situation occurs dynamically. If, because of reduced bandwidth (due to a heavy load or line errors), bytes are delayed at some point in the circuit, the probability of a subsequent byte catching up increases. As soon as this happens, the bytes will start traveling as a pair (2 bytes overhead/4 bytes of logical record).

A round-robin algorithm is used to build logical records. That is, once a logical record is made for a given channel, that channel will not be serviced again until all other channels with data present have in turn been serviced.

When a logical record is made, the record will have available a certain number of bytes of data to fill out the packet. If there are less bytes than this present, they will all be placed in the packet. If there are more, as many as possible will be placed in the packet. This means that each circuit will have equal frequency of attention, independent of the actual data transfer rate. The user typing one character every 10 seconds gets as good service as the second user who is transferring 500 characters/second.

Experience has shown that the terminal user tends to get back 10 characters for each one he types. The overhead on the individual characters he types are high, but the characters appear infrequently. The characters he gets back, since generated by a computer at a high rate, tend to come back in large logical records, thereby enhancing the efficiency.

When the originator of a data stream is not limited by mechanical processes (as in terminals), and instead can introduce bytes into the network as fast as they can flow through the network (as is normally true of the HOST computers connected to the bases), then the tendency is to make a full packet containing the data for only that circuit. In this case, the overhead for the packet is [6 bytes (packet overhead)+2 bytes (logical record overhead)]/[8 bytes (total overhead)+58 bytes of data], or 8/66 (=12.1 percent). By comparison, the WORST case is the transmission of a packet with only 1 byte of data (=90 percent), but this occurs only in the case where the line is so lightly loaded that there is only the one character to transmit, so we can afford the overhead. In a heavily loaded worst case, where we have a single byte to transmit for each possible channel, each channel will require three bytes of space within the packet, and we can thus multiplex 20 separate circuits in one packet. The overhead is now 46/66 (=69.7 percent), and if each logical record averages two bytes of data, we advance to 36/66 (=54.5 percent).

Actual measurements on lines carrying a normal mix of large and small data records indicate that the average overhead for a line is 1/6; that is, for each six bytes transmitted, five bytes are user data.

The "life" of a packet consists of being built, transmitted over exactly one line, and, when validated, being torn down (the contained data being "scattered" to the appropriate buffers). Because there are no variable delays, such as would be encountered in a multiple-link transfer, the packet can be acknowledged as soon as it is received, while the succeeding packet is in transit. This permits the use of a very small span of discrete packet identifiers. In TYMNET, the numbers range from 0 to 7, which may be specified in a 3-bit field. An acknowledgment is thus also a 3-bit field, and is piggybacked on the packet header of packets going the other way. The transmitter is constrained to never advance more than four identifiers beyond the last packet acknowledged. If the condition does occur, the oldest packet not yet acknowledged will be re-transmitted, until acknowledgment does occur. The acknowledgment specifies the most recent, sequential packet successfully received, and thus may acknowledge from 0 to 4 packets. Because there are never more than four packets in circulation, the amount of buffering required for packet storage in TYMNET is greatly reduced over conventional packet networks.

CIRCUIT CONSTRUCTION

Circuits are built at the request of a user. A circuit may only be built to a base, but may originate at either a TYMSAT or BASE port. The process will be described for a circuit originating at a TYMSAT, since this will permit the demonstration of some of the interface functions. A circuit origination from a base is conceptually the same, but varies in implementation.

Consider a user who dials up a TYMSAT node. The TYMSAT answers the call, but knows nothing of the characteristics of the caller. A message is transmitted to the terminal, requesting that the terminal be identified. The message assumes that the terminal is a 300 baud ASCII terminal. The user now types a single character which is used both to determine the terminal speed, and to identify both the parameters and processes to use for this terminal. (E.g., an ASCII terminal will have a different process ~~than~~ fr an IBM 2741, because the character codes are different, as well as the interaction mechanisms.) As soon as the terminal is identified, the message "please log in" goes out to the terminal, and the port is placed into a log-in mode. The user types the account name, an optional destination (host), and a password. This stream of characters is transmitted to the supervisor, along with information as to the node and port originating the information. The supervisor examines and validates this information, and, if any errors are found, the user is informed and placed back in log-in mode (errors may be bad account name, bad password, or destination inaccessible). If no errors are found, the supervisor examines the network topology for the best path from the originating node to the destination node, taking into account a variety of parameters. (Note that the destination specified is a host, not a node. This independence permits the free movement of hosts from one base to another if the situation demands. A host which has hardware failures can be readily moved to standby hardware, even if on a different base.) When the best path is found, resources are allocated all along that path, and appropriate messages are sent by the supervisor to each node along the path as to the linkages required for that node. The resources allocated by the supervisor consist of the channel on each line along the path, a pair of buffer descriptors for each intermediate node (passthroughs), and the port buffers at the destination. In addition to building the circuit, the supervisor also places information into the input buffer at the destination, concerning the account name, origination node and port, and terminal type, for the host. Once this is completed, the supervisor has no further interaction with the circuit—other than avoiding the re-use of the resources—until the session finally terminates, and the supervisor is informed that the circuit (and thus the resources used by the circuit) is no longer in use. When a user logs out, or is disconnected from the destination for any reason, the interface at the

TYMSAT reverts to the log-in mode, and he can then hang up, or log back in to the same or a different destination.

NETWORK INTERCOMMUNICATION

In any large, distributed system, a mechanism must be provided for intercommunication. In TYMNET, this intercommunication takes two distinct forms. Each line always has two channels preallocated for network usage.

One of these is used only for communication between a node and its neighbor. The information transferred in this case concerns such parameters as the additional traffic the node can accept on each channel on that line. The data path for these circuits is only one line long, and may only indirectly be passed further.

The second form is used for supervisor communications. As the supervisor takes over a node, the direction toward the supervisor is recorded, and the effect is that takeover builds a tree-shaped path to each node. All communication between a node and the supervisor is over this path, and as the network changes, this path is dynamically re-established.

The supervisor and nodes communicate by sending messages back and forth over the supervisor control path. Messages going to the supervisor are distinguished by one bit from those coming from the supervisor, and all messages include a field which specifies the destination (or origination) node. Messages from the supervisor consist of several different types, and perform functions such as building circuits, reading (and changing) locations in the node memory, placing characters into a buffer, and other house-keeping functions.

Messages originating at the nodes include streams of log-in information, responses to supervisor commands, and information concerning changes in the network (such as circuit termination, line errors, and host state changes, etc.). On the basis of this information, the supervisor modifies its description of the network, and the usage of the network resources. For example, if a line goes out, the supervisor may have lost control of part of the network, has probably lost some circuits (thus freeing resources distributed through the network), and has lost some choice as to alternate circuit routing. In addition, the loss of that line may have made some possible destinations inaccessible, thereby restricting the range of choice on logging in.

When the supervisor loses part of the takeover tree, it discards the secondary descriptors for the nodes lost, and then searches the neighbors of the remaining descriptors for nodes it does not have. These nodes are re-taken through the alternate, remaining nodes, and new descriptors are built. These reflect the circuits existing through those nodes at the time they are re-taken, but the existing circuits are not affected by the temporary loss of supervisor control. The only effect is the inability to build new circuits during this time.

The same process is used when a new node comes up on the network. When a node comes up, one or more lines connecting it with the rest of the network start carrying data, and the supervisor is informed. The supervisor will not use a line for two minutes after it comes into service, since the line may be bad, but after the two minutes are up, the line is declared usable, and if the node is not yet known to the supervisor, it is taken over. This capability permits the network to operate in the face of extreme failure rates, such as a large storm with massive line-outage and power failures.

FLOW CONTROL

A network circuit is a pipeline. If no buffering is involved, the input rate is equal to the output rate, and the pipeline is rigid. Once buffering is introduced, the input rate no longer need be the same as the output rate, and the pipeline becomes ''soft'', able to accept input at a greater rate than output. However, there is an upper limit to this process, equal to the buffering capacity.

TYMNET permits the instantaneous input rate to differ from the output rate, but controls the average rate by a BACK-PRESSURE mechanism. Periodically, each node examines all the buffers associated with an input process, and if a buffer exceeds some threshold, the process is informed to stop inputting until further notice. For a host interface process, this consists of a message to stop output for that port. For line processes, it takes the form of a bit-string sent over the line periodically, with a one-to-one correspondence between bits and channels, telling the neighboring node which of the channels can transmit more data. For a terminal interface process to a device such as a cassette player, this may take the form of transmission of XOFF and XON characters to the terminal.

Because of back-pressure, the host need not concern itself with details of terminal operation, such as the need for padding characters to account for carriage return delays, which are terminal dependent. The host simply starts filling the pipeline, and, if the output rate is lowered because of added padding, or because the terminal is very slow, eventually the back-pressure backs up to the host, and will henceforth throttle host output to the level of terminal output. The same mechanism is applied if a cassette terminal is inputting to a host which is so heavily loaded that it cannot accept data at the rate the cassette can generate it. This is even more important as the data-origination capacity goes up, as for a high-speed terminal or another host.

CONCLUSION

TYMNET has been in continuous operation since November, 1971, and through continuous evolutionary development has grown to a 200 node network. During this period, it has successfully handled all of the problems encountered, and shown an excellent level of reliability. Its commercial success and demonstrated abilities have led to its successor, TYMNET-II, being based upon the same general principles.

REFERENCES

1. Tymes, L., "TYMNET—A Terminal Oriented Communications Network," *AFIPS Conference Proceedings,* Vol. 38, Spring 1971.
2. Rinde, J., TYMNET-I—"An Alternative To Packet Switching Technology," *Third International Conference On Computer Communications,* August 1976.
3. Rinde, J., "Routing And Control In A Centrally Directed Network," *AFIPS Conference Proceedings,* Vol. 46, 1977 National Computer Conference.

Packet switched network in Japan

by TOSHIHARU TAKATSUKI, JIRO IIMURA, MASATO CHIBA and MASAYUKI ABE

Nippon Telegraph and Telephone Public Corporation
Tokyo, Japan

ABSTRACT

Research and development of public data networks have been conducted by various countries to cope with recent diverse data communication demands. Nippon Telegraph and Telephone Public Corporation (NTT) has also been advancing developmental projects on a packet switched network and a circuit switched network. Commercial tests on each will be begun in 1979.

Public data networks require the use of standard device-independent interfaces between networks and user devices. Especially, standard interfaces used in packet switched networks need to be specified even concerning an information transfer phase. As the fruits of energetic discussions made by CCITT, ISO etc., CCITT Recommendation X.25 for packet-mode terminals has been completed recently. On the other hand, the interface for non-packet-mode terminals will be discussed in CCITT SG VII during this study period 1977-1980.

This paper describes NTT's packet switched network, laying emphasis on details of communication protocols for packet-mode and non-packet-mode terminals.

INTRODUCTION

NTT has been carrying out developmental research on a new switched data network since 1971. As the first stage of development, an experimental data switching system, DDX-1, was designed and installed at the Musashino Electrical Communication Laboratory in June 1973. The DDX-1 was a hybrid switching system, which could serve a subscriber with either circuit switching or packet switching, according to the call type. Based on experience gained by the DDX-1, an improved experimental data switching system, DDX-2, was developed as a prototype for commercial use. In the DDX-2, circuit switching and packet switching functions are separately embodied, considering feasibility of system configuration advancement. Field trial of the DDX-2 circuit switching system was begun on a twenty-four-hour full operation basis in March 1976. With regard to a packet switching function, basic technology required for computer communications has been preliminarily tested in the laboratory. The DDX-2 packet switching system will be put into full operation by late 1977. Commercial switching systems for a circuit switched network and a packet switched network are now being designed, in parallel with the developmental research. Each system will be put into commercial service in 1979.

The circuit switched network and the packet switched network have their own optimum application fields, respectively. Judging from the communication cost viewpoint, the circuit switched network is appropriate to rather long-message transmission, while the packet switched network is appropriate to rather short-message transmission. The result of NTT's market research concerning future demand growth shows that each network will be used by more than several thousand customers even at the initial service stage. Therefore, it was decided to develop the circuit switched network and the packet switched network independently for the present, in view of network extension flexibility.

There are two types of basic packet switched service: virtual calls and datagrams. In datagram service, datagrams may not be delivered to a destination address in the same order in which they were input to the network. Also, datagrams sent from different subscriber terminals or computers may be simultaneously delivered to the same destination address, resulting in data from originating subscribers being tangled with each other. Consequently, users are required to individually take steps to manage these phenomena. Moreover, datagrams have another problem to be taken into account by the network, wherein no flow control and delivery confirmation technique relating to datagram service has been established as yet. On the other hand, virtual call service enables the network to provide common flow control procedures to avoid the above-mentioned phenomena. Therefore, NTT has adopted virtual calls as basic packet switched service. Although one of datagram service benefits is that subscribers can omit call establishment and clearing procedures, NTT expects that the same benefit would be given to subscribers by offering permanent virtual circuit service as well as virtual call service. However, there may be possibility of providing datagram service in the future, although further study is required.

For the purpose of international standardization concerning virtual calls and permanent virtual circuits, the specification of the packet-mode interface was submitted as a draft recommendation to CCITT SG VII during the last study period. This was finally declared as CCITT Recommendation X.25. NTT's packet switched network incorporates the Recommendation at almost all points. In addition, NTT has specified communication protocols for the non-packet-mode interface, by which effective data communications between packet-mode terminals and non-packet-mode terminals would be actualized.

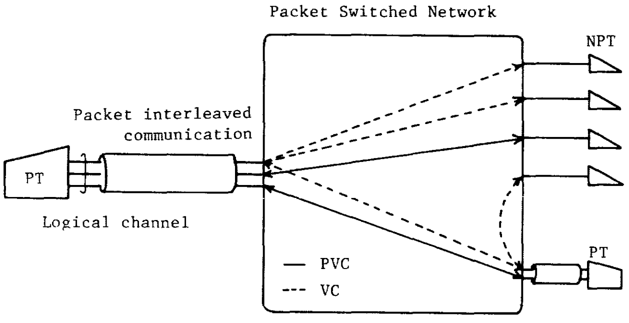
GENERAL DESCRIPTION

NTT's packet switched network outline is presented hereafter.

Service specifications

All data communications within the network are provided by means of full duplex virtual circuits. A virtual circuit may be permanent or switched. A permanent virtual circuit allows data transmission between two pieces of equipment, which are assigned at the time of subscription to the service, at any time without using call establishment and clearing procedures. On the other hand, a switched virtual circuit, called a virtual call, allows data transmission after a temporary logical connection between calling and called parties has been established using an access protocol.

Two types of terminals are accommodated to the network: packet-mode terminals and non-packet-mode terminals. A packet-mode terminal can send or receive data in the form of packets. Consequently, it can simultaneously communicate with a number of terminals through a single subscriber line by establishing many virtual circuits. This system is called packet interleaved communication. A non-packet-mode terminal cannot manage packets by itself. Messages transmitted from a non-packet-mode terminal are assembled into packets by the network, while packets addressed to a non-packet-mode terminal are disassembled into messages by it. A non-packet-mode terminal cannot perform packet interleaved communication. Figure 1 illustrates a communication example within the network using virtual circuits.



Note PT : Packet-mode terminal

NPT : Non-packet-mode terminal

Figure 1—A communication example in the packet switched network

Data signaling rates of the terminals accommodated to the network are 2.4 kb/s, 4.8 kb/s, 9.6 kb/s and 48 kb/s for synchronous packet-mode terminals, 2.4 kb/s, 4.8 kb/s and 9.6 kb/s for synchronous non-packet-mode terminals and 200 b/s, 300 b/s and 1.2 kb/s for start-stop non-packet-mode terminals.

Optional user facilities on a virtual call are direct call, closed user group with or without outgoing access, calling line and called line identification, abbreviated address calling and lump-sum center payment.

Physical characteristics of the interface between data terminal equipment (DTE) and data circuit-terminating equipment (DCE) are based on those of CCITT Recommendations X.20 and X.20bis for start-stop terminals and X.21 and X.21bis for synchronous terminals. Other characteristics of the DTE/DCE interface are described in the following sections, including procedures for physical link establishment, call control and data transfer.

Transmission control procedures, which can be used for access to the network, are the High Level Data Link Control (HDLC) procedure, the conversational basic mode control procedure, the full-duplex basic mode control procedure and the delimiter procedure. Although a start-stop non-packet-mode terminal can select one out of the above-mentioned procedures except the HDLC procedure, a synchronous packet-mode or non-packet-mode terminal should use the HDLC procedure. In the case of the delimiter procedure, the end of a message should be indicated by special characters called delimiters, which are specified by the network, and there are no other restrictions. In principle, mutual communications between all terminals within the network may be possible by means of speed, code and procedure conversion. However, code or procedure conversion will cause communication cost increase, because a considerable quantity of software is needed at each Packet Assembler Disassembler (PAD) in order to offer this facility to all combination of data terminals. Therefore, it is provisionally desirable to set some limit to the possible combination of transmission control procedures. Table I shows the allowed combination of the transmission control procedures in the network.

Charging scheme

The tariff in the network is composed of monthly charge and packet charge. The monthly charge changes, depending on DTE data signaling rates. The following description is concerned only with the packet charge.

In general, there are two basic charging techniques which might be used within public packet switched networks to charge for packet traffic. One is uni-directional charging, in which only one party is charged for the entire call or transaction. The other is bi-directional charging, in which the charge for the call is split between both parties, according to which party sent or received the packet.

TABLE I—Possible DTE transmission control procedures combination

Called Party	Calling Party	PT	NPT			
	Procedure	HDLC	(1)	(2)	(3)	(4)
PT	HDLC	○	○	○	○	○
NPT	(1) HDLC	○	○	—	—	—
	(2) Basic Mode, Conversational	○	—	△	—	—
	(3) Basic Mode, Full Duplex	○	—	—	△	—
	(4) Delimiter	○	—	—	—	△

Note ○ : Possible

△ : Possible with conditions

Technically, either of these packet charging techniques could be utilized successfully in the case of virtual calls. However, NTT's packet switched network has adopted uni-directional charging, in view of future possibility of interworking between the packet switched network and existing telephone or telex networks.

On a normal virtual call, all packets are charged to the calling party. In addition, if the called party has contracted lump-sum center payment, which is one of the optional user facilities in virtual call service, all packets during the course of a virtual call are charged to the called party. Namely, ~~lump-sum center payment means reverse charging on an~~ agreed contractual basis.

Each message is charged for according to the number of packets required for its transmission. The packet charge will slightly depend on the distance between calling and called parties, although it is independent of the call duration and other factors, such as DTE data signaling rates.

In the case of permanent virtual circuit service, all packets are charged to one of the two parties, according to a determination made at the time of subscription to the service. In other words, the packet charge for the permanent virtual circuit is linked to the volume of packets transmitted and is the same as that of the virtual call.

System configuration

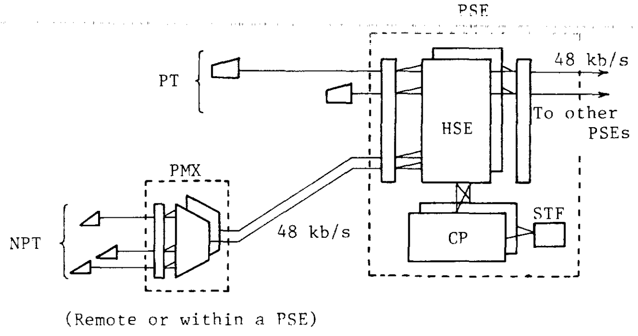
A packet-mode terminal is directly accommodated to a Packet Switching Exchange (PSE) and a non-packet-mode terminal is accommodated to a Packet Multiplexer (PMX) which operates as a PAD.

Access to the network is possible through analog or digital leased circuits as well as four-wire subscriber lines. Access to the network via the telephone network or the telex network is not available for the present. Subscriber accommodation patterns are illustrated in Figure 2.

The PSE stores packets sent from/to packet-mode terminals, PMXs or other PSEs in its processor memories via the High-speed Signal Control Equipment (HSE) and then forwards them, as shown in Figure 2. The PSE processor is the same as that of D10, which was developed for stored program control electronic telephone switching systems and has achieved satisfactory operation results at more than one hundred telephone exchanges in Japan. The PMX assembles characters or bit streams sent from non-packet-mode terminals into packets on an octet basis and disassembles packets sent from the PSE into original form. All transmission lines between PSEs or between PSE and PMX are duplicated 48 kb/s channels. Each PSE and PMX function unit is also duplicated to realize high system reliability.

Traffic control

In the packet switched network, a feedback signal regulating input packet flow is needed to keep step with output packet flow, because the number of network buffers is finite. Traffic control adopted in the network is classified into window control and packet buffer allocation. The window control is used for setting some limit to the number of packets travelling on a logical link. In addition, receiving packet buffers in the PSE are allocated individually to each PMX or packet-mode terminal. The number of receiving packet buffers for a packet-mode terminal is determined at the time of subscription to the service. In case of buffer shortage, input of packets from remote terminals is suppressed for a period, independently of window control.



Note PSE : Packet Switching Exchange

PMX : Packet Multiplexer

HSE : High-speed Signal Control Equipment

CP : D10 Processor

STF : Supervisory Test Frame

Figure 2—Packet switching system configuration

All packets during the course of a call are numbered at the source PSE. The packet sequence number is checked at the destination PSE and the packets are sent to the destination PMX or packet-mode terminal, being rearranged into the correct sequence order. If there is a missing packet in the sequence, the destination PSE waits for the packet to be transmitted for one second. After the packet concerned arrives at the destination PSE, it is sent to the PMX or the packet-mode terminal with the remaining packets. If it is not transmitted within the pre-ordained time limits, the destination PSE discards the subsequent packets to the packet concerned. In this case, all undelivered packets should be retransmitted by means of retransmission procedures of the destination PMX or packet-mode terminal.

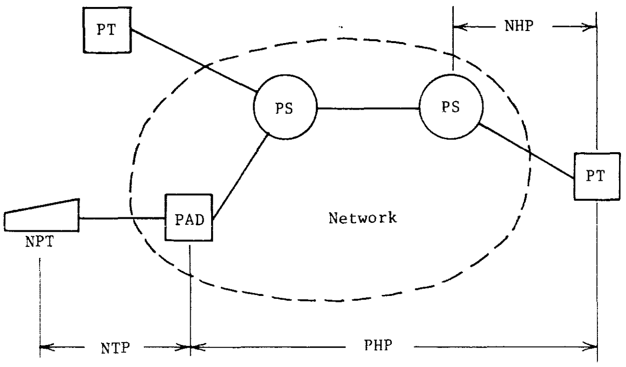
Service quality

Sixteen bit length Frame Check Sequence (FCS), based on the HDLC procedure, is annexed to each packet in the network. Bit errors are detected with high accuracy at the receiving side. Therefore, bit errors are caused only by packets where FCS cannot detect error occurrences. If the lines with an order of 10^{-6}-10^{-7} bit error rate are used, the bit error rate through the network is expected to be on the order of 10^{-11}-10^{-12}.

Each packet is transferred by store and forward technique. Therefore, the packet transfer time becomes longer in comparison with circuit switching. Packet transfer time consists of electrical transmission delay, handling time in the switching equipment, waiting time for transmission and serial-parallel conversion time. The target value of this transfer time through the network is 150 msec on an average, which is much shorter than the allowed time in conversational communications.

NETWORK ACCESS PROTOCOL SCHEME

Subscriber interfaces, which is required to access NTT's packet switched network, include three types of protocol shown in Figure 3, the Network-Host protocol that specifies the packet-mode interface, the PAD-Host protocol that defines the interface between the Packet assembly/disassembly function (PAD) and the packet-mode terminal, and the Network-Terminal prctocol which is required to enable non-packet-mode data terminals to be accommodated to the network.



NHP : Network-Host protocol
PHP : PAD-Host protocol
NTP : Network-Terminal protocol

Figure 3—Network access protocol scheme

Network-host protocol

In order to make it easy to implement the end-to-end protocol from the viewpoint of hardware and/or software, the end-to-end protocol should be divided into three independent layered protocol levels, besides the physical interface level, as shown in Figure 4. Network-Host protocol defined by the network specifies two of them, that is, the Frame level protocol and the Packet level protocol. Frame level protocol has error detection and correction functions for transmission errors on the subscriber line between the DTE and the network. That is, the Frame level protocol ensures an error free link for the Packet level delivery of packets. The Frame level protocol is compatible with the HDLC procedure standardized by ISO. The Packet level is the highest protocol, by which virtual call control, data transmission, flow control, etc., are handled. The virtual call is set up with the Call Request packet, shown in Figure 5. In the Call Request packet, the logical channel number, chosen by the DTE, is indicated as well as the network address of the called DTE. The permanent virtual circuit is discriminated from the virtual call with a certain logical channel number specified by the network and the DTE. The facility field is needed only when the user wishes to request optional user facilities, such as the closed user group and the indication of window size for the flow control. When the DTEs wish to use the closed user group facility, each DTE should register the opposite DTE address mutually to the network by the Registration packet, shown in Figure 6, with an abbreviated number indicated in the facility field.

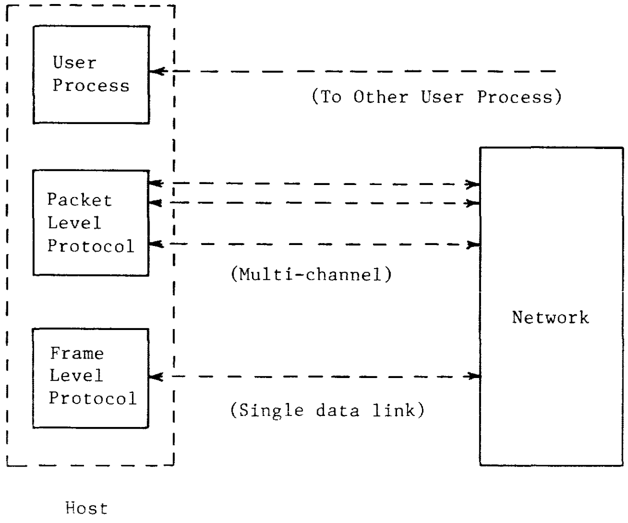


Figure 4—Network-host protocol architecture

		(Octet)
0 0 1 0	Logical	1
Channel Identifier		1
0 0 0 0 1 0 1 1		1
Calling DTE Address Length	Calling DTE Address Length	1
DTE Address Field		4
Facility Field		n
User Data		≤128

Figure 5—Call request packet

0 0 1 0	Logical
Channel Identifier	
0 1 0 0 0 0 1 1	
0 0 0 0	Called DTE Address Length
DTE Address Field	
0 0	Facility Length
1 0 0 0 0 0 0 1	
N1	N2
N3	N4

N1, N2, N3, N4: Abbreviated registration number

Figure 6—Registration packet

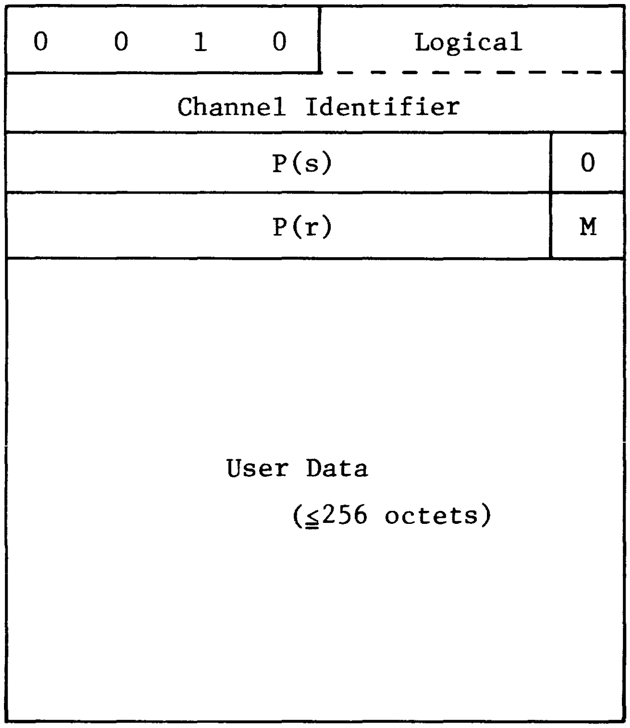
When the DTE finishes registering the opposite DTE address and abbreviated number properly, the DTE receives a Confirmation packet from the network. Through this mutual registration method, the DTE may request a call with the Call Request packet having the abbreviated registration number of the called DTE in the facility field, as shown in Figure 7. If the DTE, which has already accomplished the registration, wants to cancel it, the DTE can do with a Cancellation packet, and also may get the Confirmation packet from the network. User data may be added, following the facility field, up to a maximum of 128 octets.

Data packet, shown in Figure 8, should be used for data transmission after the virtual circuit has been set up. In the data packet, the packet send/receive sequence number has to be indicated similarly to the control byte of HDLC information frames. The modulo 128 is adopted as the sequence number, because the Data packet delivery is confirmed from end to end by the Receive Ready packet with the receive sequence number. Sequence numbers are indicated on the third octet of the packet header for the packet send sequence number P(s) and on the fourth octet for the packet receive sequence number P(r). User data length in the data field may be variable up to a maximum of 256 octets. For the present, only one maximum data field length, 256 octets, is adopted and More Data Indication may be used on a user basis.

0 0 1 0	Logical
Channel Identifier	
0 0 0 0 1 0 1 1	
0 0 0 0	0 0 0 0
0 0	Facility Length
1 0 0 0 0 0 1 1	
N1	N2
N3	N4

N1, N2, N3, N4 : Abbreviated Number

Figure 7—Call request packet with the abbreviated registration number



M : More Data Indication

Figure 8—Data packet

In order to prevent the receiving DTE buffer overflow, flow control is executed with the window size Ws, based on the packet sequence number, which is specified in the Call Request packet facility field. The calling DTE, which has received the Receive Ready packet with a sequence number Pr, may send Data packets with a sequence number up to Pr+Ws−1. The Receive Ready packet should be transmitted from one end to another, not link by link. End-to-end Receive Ready packet transmission means that the sending DTE can detect a lost packet, if any, certainly, when the Receive Ready/Receive Not Ready packet has been received from another end. When the call cannot be established or the call is cleared, the network delivers the Clear Indication packet which contains call progress signals, specified in X.25, indicating the reason for the clearing.

PAD-host protocol

As the communication between a packet-mode terminal and a non-packet-made terminal is executed through the PAD function provided by the network, the PAD-Host protocol should be specified with functions such as flow controls, lost packets recovery, conversion of data terminal control procedures, etc. A window size, which is indicated in Call Request packet being sent from the packet-mode terminal to the PAD, should be specified according to the data signalling rate of the called non-packet-mode data terminal. Non-packet-mode terminals accommodated to the network can be classified into either standard class or delimiter class. The terminal recognized as the standard class must be equipped with network standard transmission control procedure functions. This standardization gives an advantage for Hosts which provides the possibility to handle various standard transmission control procedures as one network virtual terminal. The PAD assembles data into packets with a virtual terminal control character which indicates transmission control command for communication between PAD and Host. Examples of the virtual terminal control characters are shown in Table II, and the virtual terminal control character is indicated in Call Request packet or Data packet for the virtual terminal control character transmission. Data transmission through PAD is shown in Figure 9.

TABLE II—Virtual Transmission Control Character

V.T.C.	Abbreviations	Bit patterns 8 … 1
Enquiry	ENQ	0 1 0 1 0 0 0 0
Acknowledgement	ACK	0 0 0 0 0 1 0 0
Negative acknowledgement	NAK	0 0 0 0 1 0 0 0
End of transmission	EOT	0 0 0 1 0 0 0 0
End of connection	EOC	0 0 1 0 0 0 1 0
Quit	QIT	0 0 1 0 0 0 0 0
Suspend	SPD	0 0 1 0 0 1 0 0
Wait before transmission	WBT	0 0 1 0 1 0 0 0
Request response	RQR	0 0 1 0 1 1 0 0
Acknowledgement and quit	AQT	0 0 0 0 1 1 0 0

Network-terminal protocol

Non-packet-mode data terminals accommodated to the network can be classified into either standard class or

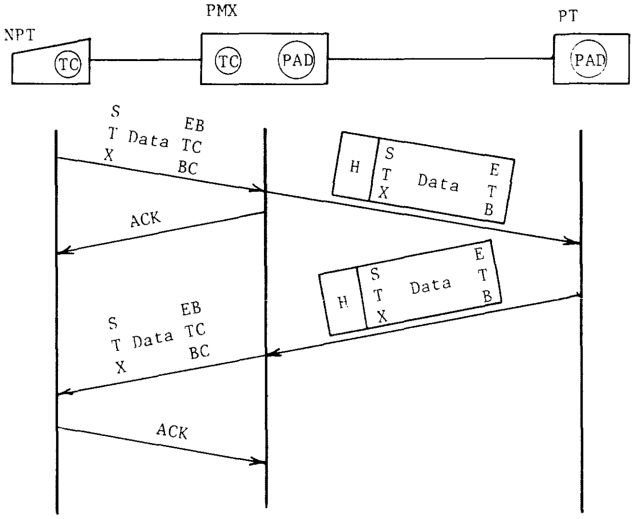


Figure 9—Illustration of data transmission through PAD

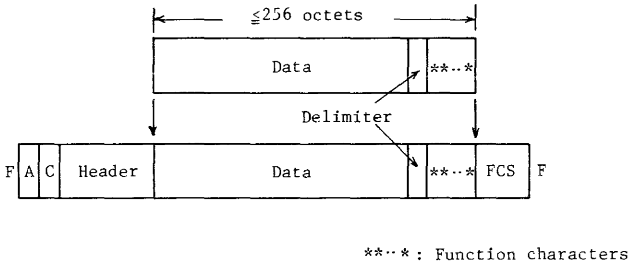


Figure 10—Packet assembling with delimiter

delimiter class, as described before. The standard class includes synchronous terminals with the HDLC procedure and start-stop terminals with Basic mode control procedures (conversation/full duplex). The data terminal HDLC procedure is also compatible with ISO standard. In order to set up and clear a virtual circuit, the terminal with the HDLC procedure has to have switching connection control, including dialing procedure. Signaling information, including selecting signals, is delivered to the network on UI command/response in the HDLC procedure, without adopting X21 signaling sequence. On the Basic mode control procedures of data terminals in standard class, selecting control is executed by X.20 or X.20bis with the Network Control Unit.

For the delimiter class which includes only start-stop terminals, the network defines a few sets of delimiters syntactically, which trigger packet assembling, and is not concerned with the algorithms of the transmission control procedures itself. In order to ensure network transparency for end-to-end functions such as error control, information for end-to-end functions can be put into packet format by assembling the packet within a certain time after a delimiter is received, as shown in Figure 10. Delimiter sets are shown in Table III.

TABLE III—Delimiter Sets

Set	Delimiter	8			Code			1
1	ETX	*	0	0	0	0	0	1	1
	ETB	*	0	0	1	0	1	1	1
	EOT	*	0	0	0	0	1	0	0
	ENQ	*	0	0	0	0	1	0	1
	ACK	*	0	0	0	0	1	1	0
	NAK	*	0	0	1	0	1	0	1
	DLE	*	0	0	1	0	0	0	0
2	ETX	*	0	0	0	0	0	1	1
	NL	*	0	0	0	1	0	1	0

* : 0/1

CONCLUSIONS

This paper has described service specifications, charging scheme, system configuration and communication protocols in NTT's packet switched network. Public packet switched networks require the adoption of internationally agreed upon standard device-independent interfaces between networks and user terminals. Therefore, it is urgently needed to standardize the non-packet-mode interface, which has not been specified by CCITT Recommendation as yet. NTT considers that the non-packet-mode interface presented in this paper is an approach to accommodating various data terminals, including existing ones, to the packet switched network effectively.

NTT's commercial packet switched network is now at the system design stage. Its hardware and software manufacture will be begun soon and its service will be cut over in 1979.

ACKNOWLEDGMENTS

The authors wish to heartily thank the many people who have participated in NTT's packet switched network project, especially Mr. J. Ono, Deputy Director of Engineering Bureau, and Mr. Y. Mima, Director of Switching Systems Development Division, Musashino Electrical Communication Laboratory, for their valuable suggestions.

REFERENCES

1. Chiba, M. et al., "A Commercial Test of Digital Data Network and Its Future Survey," *Proc. of International Switching Symposium*, October 1976.
2. Mima, Y. et al., "The DDX-2 Digital Data Switching System," *Proc. of International Switching Symposium*, October 1976.
3. Nakamura, R. et al., "Some Design Aspects of a Public Packet Switched Network," *Proc. of International Conference on Computer Communication*, August 1976.
4. Kato, M. and H. Ikeda, "An Experimental Digital Data Switching System—DDX-1—," *Proc. of International Switching Symposium*, September 1974.
5. Iimura, J. and T. Takatsuki, "Digital Data Network Field Trial," *Japan Telecommunications Review*, Vol. 18, No. 4, pp. 230–237, October 1976.

Modular programming conventions in assembly languages*

by SHY-MING JU

University of North Carolina at Charlotte
Charlotte, North Carolina

ABSTRACT

The methodology of modular programming has received increased attention in which the most basic yet important part is the intermodule communication. Circumstances may dictate the use of assembly language in software development. However, linkage conventions at this level are generally lacking or unworkable. This paper proposes a calling sequence convention and a calling sequence handler for intermodule communication. This scheme is simple to use and enhances good programming practices, such as simplicity, flexibility, comprehensibility and integrity.

INTRODUCTION

The methodology of modular programming has received increased attention in recent years.[1-4] It permits parallel development of modules and affords product flexibility and comprehensibility. Criteria for modularization have been recommended to design each module around a suitable abstraction[5] or to have each module hide only one design decision.[3] If the system configuration and software tools are adequate, modular programming is usually undertaken in high-level languages. However, when memory storage is a limiting resource, or no suitable high-level language is available, or run-time support software is insufficient, or execution efficiency is of paramount importance, then assembly language programming will be the last resort. Programming in assembly language especially prevails among minicomputer users because most of the existing minicomputers suffer from one or more of the above mentioned deficiencies.[6]

Indeed most programming languages directly or indirectly make use of assemblers, linkage editors, and possibly macro processors, therefore modularly designed software is nothing more than a set of independently assembled subroutines and data blocks when it is viewed as a runnable representation. Consequently the standards and conventions on intermodule communication at this level are the most basic yet important part of modular programming. The author has observed various manufacturers' software and found that such standards and conventions are generally lacking, poor or unworkable. This lack of standards and conventions in the software has resulted in incompatibility within the same installation, as well as between different installations.

In the following sections we will propose conventions and supporting mechanisms for intermodule communication which are simple to use and enhance good programming practice.

LINKAGE CONVENTIONS

A well-documented and well-known linkage convention is the one established for IBM System/360[7] which requires the called module to save the values contained in the general registers as soon as it gets control, and then restore those values prior to returning control. It also requires the calling module to provide a "save area" for that purpose, and to pass the address of the save area to the called module through Register 13. It further requires the calling module to use registers 15, 14, and 1 in the calling sequence, in which Register 15 will contain the entry point, Register 14 the return address, and Register 1 the parameter or the address of a list of parameters.

To an assembly language programmer, this kind of convention is certainly not enjoyable, especially if a macro facility is not available. As far as problem-solving is concerned, this linkage mechanism is not part of the algorithm; therefore it tends to be overlooked by the programmer and thus causes errors when intermodule communication takes place.

The save area reserved in each module automatically eliminates the potential for recursion or reentrancy, thus contradicting a goal of modular programming.

In our proposal, the responsibility for saving and restoring the contents of registers, and that of providing a save area, are delegated to a "calling sequence handler" which is invoked by a calling sequence through a software "trap" such as a supervisor call. To illustrate the complete process of intermodule communication, we will choose the Interdata/M70 minicomputer[8] as our base of representation, partly because it is similar to yet much simpler than an IBM System/360, partly because our idea has been thoroughly tested on it.

* This work was supported in part by a Faculty Summer Research Grant sponsored by the Foundation of the University of North Carolina at Charlotte.

The Interdata/M70 minicomputer is a halfword-oriented byte-addressable machine. It has sixteen general registers but does not support base-register addressing scheme. It accommodates up to sixteen supervisor call services; the SVC instruction has a format similar to that of a RX-type instruction in IBM System/360.

The format of the proposed calling sequence in a calling module is:

```
SVC    0,⟨entry point⟩
DC     ⟨arg1⟩, ⟨arg2⟩, . . . , ⟨argN⟩
```

```
          ENTRY CALLER,STACK
*   CALLING SEQUENCE HANDLER
*
*   REGISTER 15 IS RESERVED FOR STACK POINTER
*   REGISTER 0 AND 1 WILL NOT BE RESTORED
*
CALLER XHR     1,1
       CH      1,X'94'
       BE      EXIT
CALL   CHI     15,STACK+160
       BNL     STKOVF
       STM     0,0( 15)                       SAVE REGISTERS
       LH      0,X'94'
       LHI     2,X'E0'                        DON'T CHANGE R0 UNTIL EXIT
       LH      1,X'98'
       SIS     1,2
       STH     1,2( 15)                       STORE ADR OF CALLING SEQUENCE
CALL1  AIS     1,2
       LH      3,0( 1)                        GET NEXT PARAMETER
       CLHI    3,32
       BNL     CALL2                          JUMP IF NOT PARAMETER
       NHI     3,X'F'
       SLLS    3,1                            REMEMBER HALFWORD HAS 2 BYTES
       STH     3,LH+2                         TO MAKE UP A LH INSTRUCTION
       LH      3,H480F
       OHR     3,2
       STH     3,LH
LH     DS      4                              LOAD ARGUMENTS IN DESCENDING SEQUENCE
       SHI     2,X'10'                        START FROM R14
       B       CALL1
CALL2  AHI     15,32                          ADVANCE  STACK POINTER
       BR      0
EXIT   SHI     15,32                          DECREASE STACK POINTER
       CHI     15,STACK
       BL      STKUNF
       LH      1,2( 15)                       TO CHECK IF ANY CALL
       LHI     2,X'E0'                        BY REFERENCE PARAMETERS
CALL6  AIS     1,2
       LH      3,0( 1)
       CLHI    3,32
       BNL     CALL3                          JUMP IF NOT A PARAMETER
       CLHI    3,16
       BNL     CALL4                          JUMP IF CALL BY REFERENCE
CALL5  SHI     2,X'10'
       B       CALL6
CALL4  NHI     3,X'F'                         TO MAKE UP A STH INSTRUCTION
       SLLS    3,1
       STH     3,STH+2
       LH      3,H400F
       OHR     3,2
       STH     3,STH
STH    DS      4
       B       CALL5
CALL3  LM      2,4( 15)                       DON'T MESS R0
       BR      1
STKOVF DC      X'8000'
STKUNF DC      X'8000'
H480F  LH      0,15
H400F  STH     0,15
STACK  DS      320
       END
```

Figure 1—Listing of calling sequence handler

where ⟨entry point⟩ is usually an external symbol declared in the called module, and ⟨arg1⟩, ⟨arg2⟩, . . . , ⟨argN⟩ are register numbers representing any of the sixteen registers.

The format of return sequence in a called module is simply:

```
SVC    0,0
```

For each calling sequence the calling module should set to each register in the "argument list" either an operand (i.e., call by value) or an address of a list of operands (i.e., call by address), as dictated by the called module. The registers in the argument list will be called "argument registers." The calling module can further "flag" some of the argument registers with a '+16' to indicate that the contents of the flagged ones may be updated after the call is completed. Note that when a flagged argument register contains an operand, it is essentially a "call by value and result."[9]

Before getting into more detail, let us consider a module which searches for the first occurrence of a specified character through a character string in a 80-bytes buffer area pointed to by a pointer. If such a character is found, its relative position is reported, otherwise the relative position is set to 80.

The assembly language code of this module is listed below:

```
                ENTRY       FIND
* THE CALLING SEQUENCE IS:
*               SVC         0,FIND
*               DC          PTR,TARGET,INDEX+16
*
PTR             EQU         15
TARGET          EQU         14
INDEX           EQU         13
WRK             EQU         2
*
FIND            XHR         INDEX,INDEX          SET INDEX TØ 0
GØ              LB          WRK,0(PTR)           LØAD A CHARACTER FRØM STRING
                CHR         WRK,TARGET           CØMPARE IT WITH THE TARGET CHAR
                BE          RETURN               IF MATCHED, DØNE.
                AHI         PTR,1                INCREMENT PTR BY 1
                AHI         INDEX,1              INCREMENT INDEX BY 1
                CHI         INDEX,80             TEST IF BUFFER EXHAUSTED
                BNE         GØ                   IF NØT, GET NEXT CHARACTER
RETURN          SVC         0,0
                END
```

A calling module may be outlined as:

```
          EXTRN    FIND
          .
          .
          .
LENGTH    EQU      5
CHAR      EQU      8
PTR       EQU      10
          .
          .
          .
          LHI      CHAR,C'$'
          LHI      PTR, BUFFER
          .
          .
          .
          SVC      0,FIND
          DC       PTR,CHAR,LENGTH+16
          .
          .
          .
BUFFER    DS       80
          END
```

It should be noted that even though the module "FIND" has updated both PTR and INDEX, only the updated value of INDEX will be carried back to the calling module because it was flagged in the calling sequence. Also noted is that the called module accesses actual parameters through registers starting from Register 15 and in descending order, without knowing which registers were used by the calling module as argument registers.

To discuss the implications of the proposed calling sequence, we need to describe the functions of the calling sequence handler first.

THE CALLING SEQUENCE HANDLER

The calling sequence handler maintains a "save stack" for each "job." By job we mean a sequence of logically related processes, each process represents a running module. The stack area is inaccessible to programmers.

When the handler is invoked by a calling sequence, it first stores the contents of all registers and the return address into its stack, and adjusts the stack pointer. It then copies the contents of the argument registers into Register 15, 14, . . . , in descending order so that the correspondence between the actual argument registers and the formal argument registers is established. After this is done, it transfers control to the called module.

When the handler is invoked by a return sequence, it first copies the current contents of the corresponding registers into the flagged argument registers, and restores the remaining registers. It then retrieves the return address from the stack, adjusts the stack pointer, and transfers control to the instruction following the specific calling sequence.

A listing of the assembly language calling sequence handler is shown in Figure 1. In that implementation Register 15 was reserved for the stack pointer in order to eliminate the operations of loading and storing the stack pointer. Also, Register 0 and 1 were reserved as scratch registers so that intermediate results may be held there. Therefore, we will pretend that there were only fifteen general registers available to the programmer and Register 0 and 1 should only be used as scratch registers.

IMPLICATIONS OF THE PROPOSED SCHEME

In the proposed scheme, there is only one save stack for all modules of a job. Since it is unlikely to have all modules activated at the same time, the total memory storage allocated to the stack can be less than the total of all the save areas that would have been allocated to each of the modules.

The maximum number of argument registers allowed in a calling sequence is the number of registers available to the programmer in a specific implementation; for example, thirteen in our case. This limitation will not, however, impose a serious inconvenience when a large number of argument registers are required, because the parameters can be placed in a block of memory storage with only the block address passed through an argument register.

As assembly language programmers all know, efficiency in program execution and conservation of storage space can be attained by avoiding unnecessary memory accesses and by using RR-type instructions wisely. The convenient facilities of call-by-value and call-by-value-and-result for the calling sequence were designed to encourage this kind of good programming practice. Call-by-address was provided mainly for parameters which are structured data, such as an array.

It is interesting to note that if a module is pure and if its calling sequence involves only call-by-value and/or call-by-value-and-result, it will naturally be recursive and reentrant, as can be seen in the following example which computes the N-factorial:

```
                ENTRY     FACTØR
* THE CALLING SEQUENCE IS:
*               SVC       0,FACTØR
*               DC        RESULT+16,N
*
RESULT          EQU       14
N               EQU       13
M               EQU       2
*
FACTØR          LHR       N,N
                BZ        LBL                IF N=0 THEN RESULT IS 1
                CHI       N,1
                BE        LBL                IF N=1 RESULT IS ALSØ 1
                LHI       M,-1(N)            SET M TØ N-1
                SVC       0,FACTØR
                DC        RESULT+16,M        CØMPUTE FACTØRIAL ØF N-1
                LHR       1,RESULT
                MHR       0,N
                LHR       RESULT,1           SET RESULT TØ RESULT*N
                SVC       0,0                RETURN
LBL             LHI       RESULT,1           SET RESULT TØ 1
                SVC       0,0
                END
```

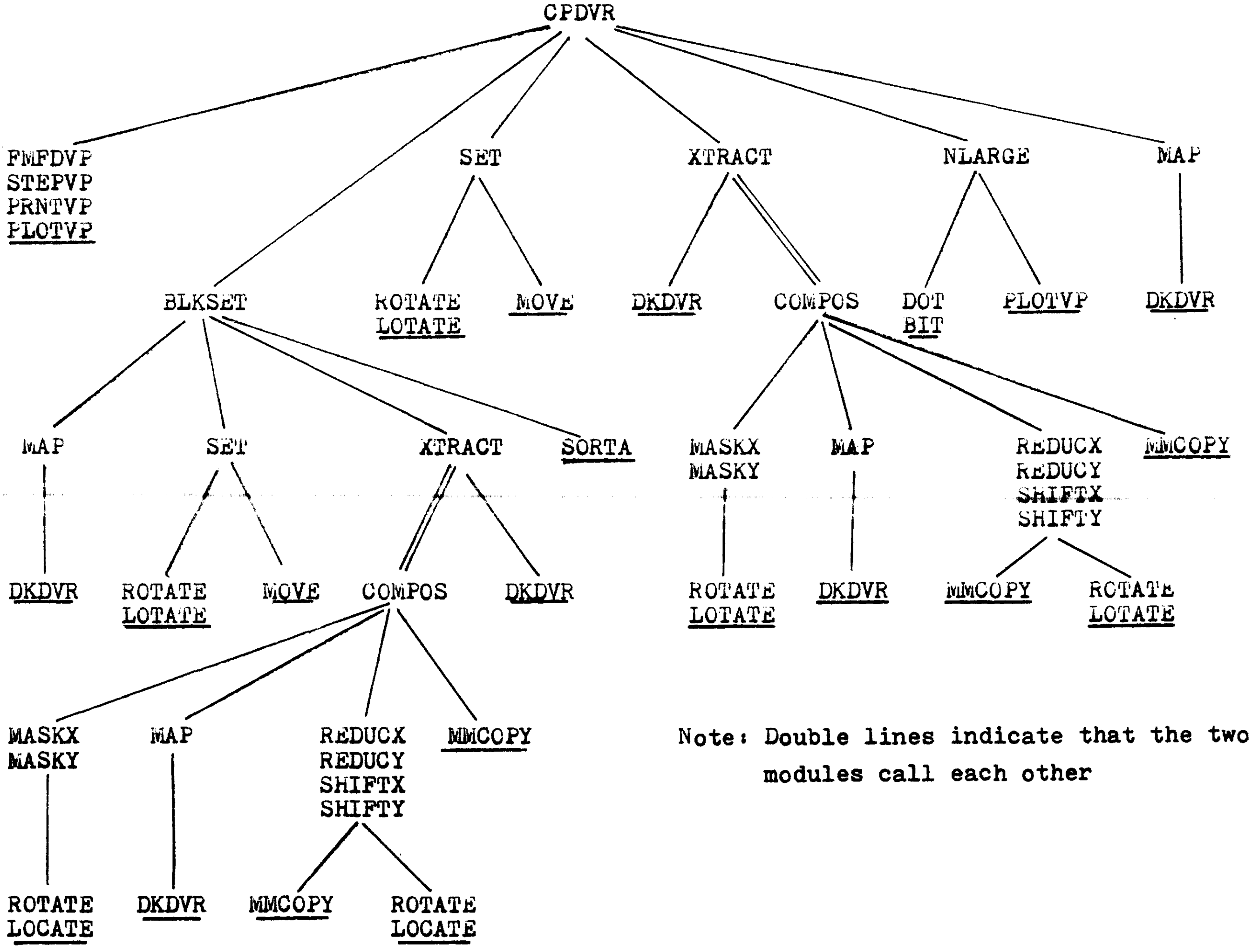
It is recommended that any communication between a module and its global environment should be through argument registers, and the conventional way of accessing common area through "external declarations" should be used with great discretion or better yet, should be prohibited. This practice would achieve two advantages; on the one hand, the fact that arguments are explicitly shown in the calling sequence and only flagged arguments can be updated by a called module greatly improves the readability of the module logic; on the other hand, the fact that a module does not know and does not need to know its global environment except through the argument registers enhances module integrity. This practice is especially important in large software development efforts undertaken by a team of programmers.

The calling sequence handler can help in system integration testing by inserting code that prints a message whenever control enters or exits from a module. Should a fatal error occur during execution, the module at fault can be determined and the execution history retained in the save stack can be dumped out for diagnosis. The calling sequence handler can even help in resource management; for example, allocation and deallocation of work area to a module. Since the handler monitors module entrance and exit, it can also participate in dynamic linking when appropriately modified.

CONCLUSIONS

The proposed calling sequence conventions and the calling sequence handler have been used successfully in a large software project which employed modular programming in a chief programmer team approach.[10,11] All modules were developed and tested independently, and during the system integration testing no major difficulties were encountered. The hierarchy of modules in its Chinese output subsystem is shown in Figure 2. By standardizing the calling sequence, some of these modules were able to be used without any modification in developing other software such as a text editor and a special-purpose language processor.

The proposed scheme should work equally well on machines with fewer general registers such as a PDP-11. In that case some storage locations can be set aside to simulate registers. A close look at the assembly language listing of the calling sequence handler should reveal that most of the instructions perform the functions which would be required in any linkage conventions. The only extra overhead is due to rearranging argument registers into descending ordered registers prior to entry and updating the flagged registers upon exit. However, this overhead is outweighed by gains in programming simplicity and enhancement. Ideally, the calling sequence handler should be hardwired or implemented in the form of a microprogram to further reduce overhead.



Note: Double lines indicate that the two modules call each other

Figure 2—Hierarchy of modules in the Chinese output subsystem

ACKNOWLEDGMENT

The author is indebted to the EDP Center of the Directorate General of Budget, Accounting and Statistics of the Executive Yuan, Republic of China, for giving him the opportunity to test the idea presented herein during the development of the software project MODEST.

REFERENCES

1. Freeman, P., *Software System Principles: A Survey*, Science Research Associates, Inc., Chicago, 1975.
2. Gauthier, R. and S. Pont, *Designing Systems Programs*, Prentice-Hall, Englewood Cliffs, N.J., 1970.
3. Parnas, D. L., "On the Criteria To Be Used in Decomposing Systems into Modules," *Comm. ACM*, Vol. 15, No. 12, December 1972, pp. 1053–1058.
4. Balzer, R. M., "PORTS—A Method for Dynamic Interprogram Communication and Job Control," *Proc. AFIPS 1971 SJCC*, Vol. 38, pp. 485–489.
5. Zilles, S., "Modularization Around a Suitable Abstraction," *Proc. AFIPS 1975 NCC*, Vol. 44, p. 279.
6. Waks, D. J. and A. B. Kronenburg, "The Future of Minicomputer Programming," *Proc. AFIPS 1972 SJCC*, Vol. 40, pp. 103–109.
7. Donovan, J., *Systems Programming*, McGraw-Hill, New York, 1972, pp. 465–470.
8. *Interdata Model 70 User's Manual*, Interdata Inc. 1971.
9. Gries, D., *Compiler Construction for Digital Computer*, John Wiley and Sons, Inc., New York, 1971.
10. Ju, S. M., "Design and Implementation of an Intelligent Chinese Data Entry and Report Generating System," *Proc. ICS 1975*, Vol. 1, pp. 166–174.
11. Ju, S. M., et al., "MODEST—A Modularly Designed Sino-Terminal System," Technical Reports CDPL-V3, Chinese Data Processing Lab., EDP Center, DGBAS, The Executive Yuan, Taiwan, Republic of China, May 1976.

The design and implementation of a simple programming language for microcomputers*

by J. C. CLEAVELAND and C. D. SATTEN
University of California
Los Angeles, California

ABSTRACT

GAMMA is a simple, interactive, expression-oriented programming language which grew out of a microcomputer development environment at UCLA. The language, its design and its implementation are described. GAMMA has three types of values: numbers, strings, and an undefined value, which is used for denoting errors. Numbers are variable precision decimal floating point and strings are of variable length. The only limitations on the number of digits of precision or string length is the amount of memory available. Memory management is automated leaving ease of use and response time as a user's only concerns. The major design goals of GAMMA were simplicity of the language and an easy implementation on a microcomputer.

INTRODUCTION

The rapid development of microprocessors in the past couple of years has outpaced programming language designs for the microcomputer. The large powerful programming languages of large machines are not appropriate for small machines due to the inherent complexity of the language. This complexity effectively prevents the construction of stand-alone compilers or interpreters. Cross-compilers for these languages are feasible, but require the use of a large machine.

The most successful stand-alone high-level language for microcomputers has been the family of languages derived from BASIC. Although not specifically designed for microcomputers, BASIC is an example of a simple language which is easy to implement on a microcomputer and which can be easily learned. However there are many disadvantages of BASIC. Structured programs are difficult to write and variable names are more restricted than in assembly languages. It is difficult to combine programs because all variables have the same scope and line numbers will usually conflict. Lack of parameters make this even more difficult. Many varieties of BASIC flourish and only slowly is a standard being adopted.[1,2,5,14]

BASIC is by no means the only activity in this area. PL/M[6] and its descendents PLuS,[12] MPL,[10] PL/Z[16] and others[8] have been designed to be compiled rather than interpreted. PL/M has a PL/I[7] like syntax and the effort in compiling is spent on optimization. Other languages proposed or implemented for microcomputers include FORTH,[11] a rather low level macro-like processor, MICHELLE,[3] an ALGOL like language, LITTLE,[13] a systems implementation language and TOY LISP,[4] a mini-version of LISP.

Continuing this research in language design we have selected the two goals "easy to learn" and "easy to implement" as the most important. These are similar to the goals for BASIC, FORTH and TOY LISP but are different from the goals of PL/M, MICHELLE and LITTLE which considers "efficiency" and "code generation" to be of highest priority. The resulting language which was designed and implemented has been called GAMMA. Sections of this paper describe how GAMMA evolved from a simple calculator design; presents the language with examples; and describes the implementation development.

DESIGN DEVELOPMENT

The initial ideas concerning GAMMA came from discussions on implementing some kind of simple calculator on a microcomputer equipped with keyboard and display. It was decided that a programmable variable-precision decimal floating point calculator would be a handy tool, which would not be too difficult to implement on a microcomputer. The first big decision was what kind of input the calculator should receive. The two major possibilities were infix notation (e.g., "2+3") or postfix notation (also known as reverse polish notation, e.g., "2 3 +"). Infix notation has the advantage of everyday familiarity while postfix notation has the advantage of a simple implementation. If the postfix notation were chosen, no parser would be necessary since the interpreter could execute one key at a time. The infix notation would require a more sophisticated implementation since an arbitrarily long sequence of characters (keys) must be read before execution can take place. In addition, a simple parser is necessary to determine the form of the input sequence. After much thought and discussion it was decided to attempt the more complicated implementation in favor of ease of use. more [illegible]

* Supported in part by ERDA Contract EY-76-S-03-0034, PA214

One of the original goals of this calculator was the ability to retain and execute stored programs. This implies the ability to store variable-length sequences of characters. Variable-length sequences also occur in the input routine and in numbers which have variable lengths due to the variable-precision. It was observed that if the implementation was going to manipulate strings, then the user may as well also be given the capability to manipulate strings. This decision transformed the calculator system into a language, which we have called GAMMA.

Numbers and strings became the data types of GAMMA. The inclusion of strings was quite natural and caused little added overhead. Both numbers and strings are variable-length data items and can be treated alike in terms of storage. It was also quite natural to store functions (or programs) as strings, and thus GAMMA functions merely became a subset of GAMMA string values. Only one other value was added to the language; the UNDEFINED value. This value is used to denote the result of performing an operation which has an undefined result, such as adding two strings or executing a string which is not a function.

The next major decision was how to handle variables. It seemed obvious that global variables should be represented by traditional identifiers stored in some symbol table. However managing parameters in the same way would add complexity to the implementation. Therefore it was decided to pass arguments by value and to represent parameters by a non-standard notation, which is described in a later section.

All that GAMMA lacked now was some flow-of-control constructs. Simple conditional, loop and exit constructs were added. Jumps and labels were not added because of their complexity. A secondary, but equally important reason was to encourage go-to-less programming. The addition of these flow-of-control constructs posed a new problem; since there is no Boolean type, how should Boolean values be represented? To conform to the simplicity of the language, comparison operators return the numbers zero and one. Zero represents the false Boolean value and everything else represents the true Boolean value. This also has the advantage that the operators "+" and "*" can be used for the Boolean operators "OR" and "AND."

THE LANGUAGE

GAMMA is a simple interactive language designed to be used on microcomputers. It has only a few constructs and is easy to learn. Despite the simplicity, algorithms are easy to express in GAMMA. The language GAMMA is centered around the concept that every expression results in some GAMMA value when it is evaluated (executed).

The only data types in GAMMA are numbers, strings, and UNDEFINED. GAMMA numbers can be of any size or precision and thus include both integers and reals. The user can change the number of decimal digits of precision at any time either interactively or under program control. A GAMMA string is a sequence of zero or more characters. The only restriction on the length of strings or the precision of numbers is the available memory. Finally, the UNDEFINED value is used to indicate some error in the evaluation of an expression.

All expressions are GAMMA strings, but not all strings are expressions. A grammar for the language GAMMA (given in Appendix A) tells which strings are expressions. When an expression is evaluated, the result is some GAMMA value.

Table I gives an incomplete and ambiguous, but highly readable summary of the syntax and semantics of the language GAMMA. Table II gives a list of GAMMA operators with their precedence and meaning. Table III gives a list of the pre-defined (built-in) GAMMA functions.

Each item in Table I will be briefly explained. A number (1) denotation is a sequence of decimal digits with an optional sign and optional decimal point. A string (2) denotation is a sequence of characters delimited by either quotes or by left and right square brackets. Quotes cannot appear within strings delimited by quotes and square brackets must be balanced in strings delimited by square brackets. The delimiters are not part of the string but only serve in recognizing the string.

An identifier (3) is a sequence of alphanumeric characters beginning with an alphabetic character. An identifier represents a global variable whose value is the value most recently assigned to that variable by the assignment operator, ":=." A parameter (4) is a sharp sign, "#", followed by one or two digits. A parameter represents the value of an argument passed to a function. In GAMMA the term "function" is synonymous with the term "expression." "#1" refers to the first argument; "#2" refers to the second argument, etc. If a parameter does not have a corresponding argument, then it is initialized to UNDEFINED. Parameters can be assigned new values, but of course this has no effect on the corresponding argument since all arguments are passed by value. This allows the use of parameters as local variables.

The Boolean value "false" is represented by the number zero and the Boolean value "true" is represented by all other GAMMA values. The four flow-of-control expressions (5-8) are self-explanatory and all return a value. Note that if the WHILE part of a while loop is false the very first time then the value of the loop expression is UNDEFINED. Similarly if the IF part is false in an IF expression which has no ELSE part then the value is UNDEFINED. Another flow of control construct (posing as a function) is "EXIT(e)" (see 12) which exits from the current function being executed with the value V(e).

Operators (9) are described in Table II. Any GAMMA expression may be enclosed in parentheses (10) to explicitly constrain the order of evaluation, as in "(5-2)*3".

Finally any string which results from the evaluation of a GAMMA expression may be evaluated by placing a pair of parentheses after it (11). If the string is not a valid GAMMA expression then the value is undefined. Note that these parentheses have a different meaning than the parentheses of item 10. Arguments can be passed to the function by placing a list of expressions separated by commas within the parentheses. If no arguments are passed then the parentheses have nothing inside.

TABLE I—A GAMMA expression can be any one of the following 12 constructs where "e" means any GAMMA expression and "V(e)" means the value of the evaluated expression "e"

	e "syntax"	V(e) "semantics"	Remarks
1.	a number	The numerical value of the number	A number is a sequence of digits with an optional sign and decimal point.
2.	a string	The string value. Delimiters are not part of the string value.	A string is a sequence of characters delimited by 'and', or [and].
3.	an identifier	a) The value last assigned to the identifier b) UNDEFINED if the identifier has never been assigned a value.	An identifier is a sequence of alphanumeric characters which represents a global variable.
4.	a parameter	same as identifier. (Also see 11)	A parameter is a # symbol followed by one or two digits which represents a local variable.

5.	IF e_1
	THEN e_2
	FI
		a) Undefined if $V(e_1)=0$
		b) $V(e_2)$ if $V(e_1)\neq 0$
			Conditional expression.
			All flow of control keywords are reserved and cannot be used as identifiers.

6.	IF e_1
	THEN e_2
	ELSE e_3
	FI
		a) $V(e_2)$ if $V(e_1)\neq 0$
		b) $V(e_3)$ if $V(e_1)=0$
			Conditional expression.
			Note that only one of the two expressions e_2 and e_3 is evaluated.

7.	WHILE e_1 DO e_2 OD	a) UNDEFINED if $V(e_1)=0$ b) V(DO e_2 UNTIL $e_1=0$ OD) if $V(e_1)\neq 0$	Loop expression.
8.	DO e_2 UNTIL e_1 OD	a) $V(e_2)$ if $V(e_1)\neq 0$ b) V(DO e_2 UNTIL e_1 OD) if $V(e_1)=0$	Loop expression. e_1 and e_2 are repeatedly evaluated.
9.	e_1 operator e_2	$V(e_1)$ operator $V(e_2)$	See Table 2 for a list of operators.
10.	(e)	V(e)	Used for re-ordering evaluation of expressions.
11.	$e_0(e_1, \dots ,e_n)$	a) UNDEFINED if $V(e_0)$ is not a GAMMA expression b) $V(V(e_0))$ with $\#1:=V(e_1); \dots ; \#n:=V(e_n)$	Function calls and execution of strings. $0\leq n<100$
12.	EXIT (e)	V(e)	Exits from the current function and returns the value V(e)

Some examples of GAMMA programs will now be given. The first example displays the factorial of a number read from a keyboard and displayed on a TV screen. Note that comments are delimited by dollar signs.

Example 1:

```
X:=READ( ) ( );   $ read a number  $
Y:=1;             $ set y to 1     $
WHILE X>1
DO Y:=Y*X;
   X:=X-1
OD;
TV(Y) $ display the answer $
```

In the first line of example 1, evaluation of the built-in GAMMA function, i.e., "READ( )" means read a string from the keyboard and "READ( ) ( )" means to evaluate the string read from the keyboard. This means that any expression which evaluates to a number would be acceptable input for this program. Example 2 below illustrates how this program itself can be kept as a value of a variable "F" which can subsequently be evaluated (called).

Example 2:

```
F:=[X:=READ( ) ( );
    Y:=1;
    WHILE X>1
    DO Y:=Y*X;
       X:=X-1
    OD;
    TV (Y)]
```

Example 3 shows how this function stored as a string in the variable F can be evaluated (executed).

Example 3:

```
F( )
```

As a matter of good design, one might desire to separate the function F into two functions F and P, where F computes the factorial of a number and P controls the input and output. Example 4 creates these two functions. So that these examples do not become monotonous, the factorial is computed recursively rather than with a loop.

TABLE II—List of GAMMA Operators

operator symbol	precedence	semantics	
**	9	exponentiation	(numbers only)
*	8	multiplication	(numbers only)
/	8	division	(numbers only)
+	7	addition	(numbers only)
−	7	subtraction	(numbers only)
=	5		
/=	5	comparison operators	
<	5	1 is returned if the comparison is true.	
<=	5	0 is returned if the comparison is false.	
>	5	Undefined is less than all Gamma values.	
>=	5	All GAMMA numbers are less than all GAMMA strings	
&	4	(and) 0 is returned if an operand is 0; otherwise 1	
!	4	(or) 0 is returned if both operands arc 0; otherwise 1	
: =	3	Assignment. The first operand must be either an identifier or a parameter, which is assigned the value of the second operand. The value of the assignment expression is the value of the second operand.	
;	2	Sequencing. This operator is used for sequencing GAMMA expressions, as in: A: =2; B: =3; A+B The value of this expression is the value of the last operand, which is "A+B", which has a value of 5.	

Example 4:

```
F: =[IF #1<=2
    THEN #1
    ELSE #1*F(#1-1)
    FI]
P: =[TV('ENTER A NUMBER');
    X: =READ( ) ( );
    Y: =F(X);
    TV(Y)]
```

The next example is the function which controls the whole GAMMA system and is called the control program.

Example 5:

```
CONTROL: =[WHILE TRUE
            DO #1: =READ( );
               TV(#1( ))
            OD]
```

The control program is an infinite loop which reads a string from the keyboard. This string is evaluated and the resulting value is displayed on the TV screen. The next example is a scenario taken under this control program. U is the user's input and C is the response displayed on the TV screen.

Example 6:

```
U: 2+3
C: 5
U: [2+3] $ a string of 3 characters $
C: 2+3
U: [2+3]( ) $ now execute it $
C: 5
U: P $ what is the value of P? $
C: TV('ENTER A NUMBER');
C: X:=READ( ) ( );
C: Y: =F(X);
C: TV(Y)
U: P( ) $ now execute P $
C: ENTER A NUMBER
U: 5
C: 120 $ this is printed by the function P $
C: 120 $ this is printed by the function CONTROL $
U: [#1+#2] $ a string of 5 characters $
C: #1+#2
U: [#1+#2](2, 3) $ execute with 2 arguments $
C: 5
U: A: =[#1+#2] $ assign a value to A $
C: #1+#2
U: A(2, 3) $ execute the function stored in A $
C: 5
```

IMPLEMENTATION DEVELOPMENT

The first implementation of GAMMA was written in ALGOL 68.[15] This implementation was not for a microcomputer, but instead modeled the intended implementation on a microcomputer by representing the memory as an array of characters. The implementation can be easily divided into four parts:

(1) the interpreter
(2) the parser
(3) memory management
(4) the system functions

The interpreter executes code generated by the parser. The parser uses a recursive descent parsing strategy to generate simple interpretive code consisting of 11 op codes. As the interpreter and parser are straightforward programs with no unique features they are not further detailed here.

The most interesting part of the implementation is the memory management. The memory is divided into four stacks (see Figure 1). The traditional expression stack is divided into two stacks for ease of implementation. A third stack is used for the symbol table and the last stack is used for all GAMMA values and is called VAL. The symbol table and expression stack use only pointers into VAL, called valpointers.

The symbol table and VAL are not really stacks since items are only added to the top and not taken away. New identifiers are added to the symbol table (and initialized to

TABLE III—A List of Builtin GAMMA Functions

Function Name	Semantics
READ	Read a string from the keyboard.
EDIT	Edit a string using the builtin editor.
TV	Display all arguments on the CRT.
TV2	Display a character in selected location on CRT.
PRINTER	Print all arguments on hard-copy printer.
TAPEOUT	Write a GAMMA value onto cassette tape.
TAPEIN	Read a GAMMA value from cassette tape.
REWINDOUT	Rewind the output cassette tape.
REWINDIN	Rewind the input cassette tape.
CAT	Concatenate all arguments into one string value.
SUB	Returns a substring of the first argument. (PL/1 SUBSTR)
INDEX	Finds the first occurrence of #2 in #1. (PL/1 INDEX)
VERIFY	Finds the first character in #1 not in #2. (PL/1 VERIFY)
REFLECT	Returns the left-right mirror image of a string.
LEN	Returns the length of a string.
STR	Converts a number to a string.
CHAR	Converts a number to an ASCII character.
ASCII	Converts a character to its ASCII code number.
DIGITS	Sets the maximum number of decimal digits to be used in arithmetic operations.
CHOP	Returns the integer portion of a number.
ABS	Returns the absolute value of a number.
ODD	Returns TRUE if an odd number; otherwise FALSE.
GUESS	Returns a number close to 1/#1.
PEEK	Returns the contents of memory location #1.
PEEK2	Returns PEEK(#1) + 256*PEEK(#1+1).
EXIT	Exits the currently executing function with the value of #1
RESTART	Restarts the GAMMA system. Useful if you don't want to continue execution of an interrupted program.
TRACE	Sets dynamic flow tracing. Any combination of the following five options can be specified. Trace function calls; Trace function exits; Trace assignment to identifiers; Trace assignment to parameters; Single step execution.
CONTROL	Reads input from user; executes it; and displays results.
INTERRUPT	Called by interrupt key. Interrupts currently executing function, reads input from user, executes it, and displays results. Allows return to previous function.
TYPE	Finds if a GAMMA value is a number or a string.
PROTECT	Prevents assignment to a variable.
UNPROTECT	Removes protection attribute from a variable.
PARSE	Attempts to parse a string. Returns TRUE if successful, otherwise returns FALSE.
COMPACT	Requests compaction, and also compaction of the symbol table.
FIRSTVAR	Returns the name of the first symbol in the symbol table.
NEXTVAR	Returns the name of the next symbol in the symbol table.
PARM	Returns the value of a parameter any number of levels back.
VARVAL	Returns the value of the identifier returned by NEXTVAR.

UNDEFINED) and new GAMMA values are added to VAL. To change the value of a variable only its valpointer needs to be changed and not the value that the valpointer is pointing to. Values in VAL are never changed so that the value of a valpointer is never changed. All new GAMMA values are created at the top of VAL, which is called the "workspace." VAL and the symbol table are thus always growing. When not enough workspace is left for the creation of a new GAMMA value, compaction takes place. Compaction goes through VAL and deletes all those values not referenced by either the symbol table or the expression stack. All the remaining values are moved down so that all the recovered space becomes part of the workspace. Likewise the symbol table can also be compacted by deleting all those identifiers whose value is UNDEFINED.

In the ALGOL 68 implementation, the memory management was developed using the concept of a cluster.[9] Although the cluster and the ALGOL 68 implementation in general was just an exercise, it proved to be extremely beneficial in the next implementation.

The second implementation was done on a microcomputer based on the Intel 8080 microprocessor. Only a few minor revisions were made to the implementation design for reasons of efficiency. The first implementation, in ALGOL 68, gave a basis on which to write the specifications for the second implementation. Thus the documentation for the second implementation was written before, instead of after the programming. This experimental approach proved to be quite interesting. Having the documentation and the first implementation as guides made the second implementation very simple. In a mere two weeks, most of the first implementation was recoded into 8080 assembly language and a working version of GAMMA was implemented on the microcomputer.

The most time-consuming part of the second implementation, which took about four man-months, was the set of system functions, which consisted of all the code for operators and built-in GAMMA functions written in assembly code. The first implementation only implemented some of the system functions and thus most of this code was of original design in the second implementation. Each individual function had its own problems, but was usually an easy programming task. However the net sum of all the functions turned out to be a large job.

In implementing a basic function, two approaches could be taken. It could be written with some difficulty in assembly language which is efficient and called a system function. Or it could be easily written in GAMMA, if it could be composed from the system functions. Some func-

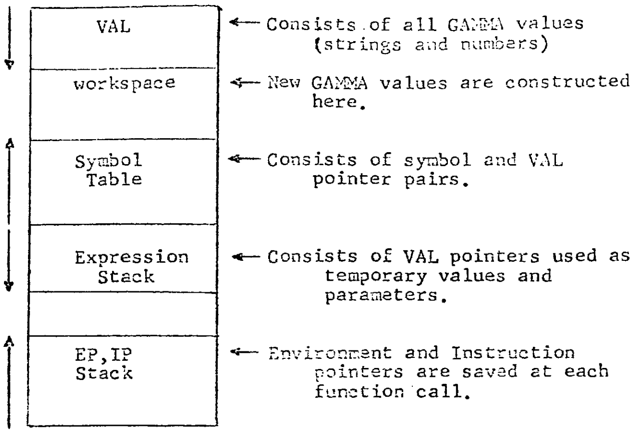

VAL
Consists of all GAMMA values (strings and numbers)
workspace
New GAMMA values are constructed here.
Symbol Table
Consists of symbol and VAL pointer pairs.
Expression Stack
Consists of VAL pointers used as temporary values and parameters.
EP,IP Stack
Environment and Instruction pointers are saved at each function call.

Figure 1—Memory management overview

tions such as I/O had to be written in assembly language. Others, like INDEX could be expressed in GAMMA using system functions such as SUB. The functions given in Table III are the functions we chose to be system functions.

In addition to the system functions, interrupt, measurement (time and space), tracing and edit features were added to enhance the utility of the system. The first program packages written in GAMMA include:

(1) Mathematical functions including trig, log and integration functions.
(2) Pretty Print functions which format GAMMA expressions.
(3) a programmed instruction course in GAMMA, complete with exercises and answers.

By the time the systems functions were complete, the only significant design change made was that the exponent field of a number was increased from one byte to two bytes.

Some statistics concerning the two implementations may be of some interest. The first GAMMA implementation consists of about 5000 lines of ALGOL 68 code. However this is misleading since a large percentage of the lines are either comments or are very short. The second implementation consists of about 5500 lines of 8080 code and occupies about 10k of storage with the following breakdown:

(1) interpreter	550 bytes
(2) parser	1600 bytes
(3) memory management	1050 bytes
(4) utility routines	1750 bytes
(5) system functions	3200 bytes

WHY ANOTHER LANGUAGE?

Microprocessors challenge the field of language design. Large languages do not satisfy the requirements of a microcomputer environment. A small language with few constructs and easy implementation, yet with enough power to easily express algorithms is the ideal. Does BASIC, which is rapidly becoming the standard language of microcomputers, satisfy these requirements? Certainly in part BASIC does, but few will argue that it is an ideal language. We do not wish to describe the shortcomings of BASIC, nor the merits of GAMMA. Obviously we feel that GAMMA is better, otherwise we would not have undertaken the immense work of designing and implementing a new language. As GAMMA embodies the principles we value highly in a language it would be difficult to objectively judge the two languages. We simply like to view GAMMA as a step forward and keeping this in mind we wish to point out the problems of GAMMA, so that the next undertaking towards a better language can benefit not only from our achievements but also the shortcomings of GAMMA.

GAMMA is by no means ideal. The variable precision arithmetic is more a toy than a tool. There are few applications that would require more than 20 digits of precision, let alone 200 or 2000. If the cost of variable precision arithmetic was small in terms of compute time it may be justified. It is by no means clear that expressiono-riented languages are better than statement-oriented languages. Expression-oriented languages are simpler and more general, but are initially harder to understand if one is more used to statement-oriented languages. The major programming tools which GAMMA lacks include arrays and call by reference. Call by reference and arrays can be simulated in GAMMA, but are expensive. Certainly an ideal language would need these or the equivalent.

ACKNOWLEDGMENTS

The first implementation in ALGOL 68 was an important development step in GAMMA. Dan Berry, Paul Eggert, Bill Fisher, Mike Urban, and Bruce Walker were all personally involved and were responsible for its success. The second implementation was supported and encouraged by Gerald Estrin who explored many new ideas with us.

BIBLIOGRAPHY

1. Altair BASIC Reference Manual, MITS, 1975.
2. ANS Committee, Draft Proposed American National Standard Programming Language Minimal BASIC, Prepared by Technical Committee X3J2-BASIC, January 1976.
3. Burgess, H. W., R. S. Fenchel, G. D. Nunes and K. H. Page, "MICHELLE: A Microcomputer Higher Level Language," in *Four Languages: Experiments in Computer Language Design* R. C. Uzgalis (Editor), U.C.L.A. Computer Science Department, Technical Report UCLA-ENG-7544, July 1975.
4. Chaitin, G. J., "A Toy Version of the LISP Language," IBM Thomas J. Watson Research Center RC5924(#25634) March 1976.
5. DEC, "BASIC/TR11 Language Reference Manual," Digital Equipment Corporation, October 1974.
6. Intel Corp. *8008 and 8080 PL/M Programming Manual,* 1975.
7. IBM *PL/I (F) Language Reference Manual* Order No. GC28-8201-4, IBM United Kingdom Laboratories Ltd., Publications Department, Hursley Park, Winchester, Hampshire, England.
8. Leach, G. C., "Microprocessor Language Design," *Proceedings DISE Workshop on Microprocessors and Education,* pp. 22-27. August 16-18, 1976, Colorado State University, Fort Collins, Colorado.
9. Liskov, B. and S. Zilles, "Programming with Abstract Data Types," *Proceedings of ACM SIGPLAN Conference on Very High Level Languages,* SIGPLAN Notices, Vol. 9, 4 pp. 50-59, April 1974.
10. Motorola, "MPL Language Reference Manual," Motorola Microsystems, 1976.
11. Rather, E. D. and C. H. Moore, "FORTH High-Level Programming Technique on Microprocessors," Electro 76, Boston, May 1976.
12. Signetics, "PLuS Reference Manual," Signetics Corp. March 1976.
13. Stuart, Thomas, "A Minicomputer Systems Language," AESOP XIV-SCIE Session, April 1976.
14. Tymshare, "SUPER BASIC," Tymshare Inc., May 1971.
15. VanWijngaarden et al., "Revised Report on the Algorithmic Language ALGOL 68," *Acta Informatica,* Vol. 5, 1975.
16. Zilog, "PL/Z," Zilog Corporation.

APPENDIX A

Backus Normal Form (BNF) description of GAMMA.

```
⟨exp⟩ ::= ⟨number⟩
        |⟨string⟩
        |⟨identifier⟩
        |⟨parameter⟩
        |IF ⟨exp⟩ THEN ⟨exp⟩ FI
        |IF ⟨exp⟩ THEN ⟨exp⟩ ELSE ⟨exp⟩ FI
        |WHILE ⟨exp⟩ DO ⟨exp⟩ OD
        |DO ⟨exp⟩ UNTIL ⟨exp⟩ OD
        |⟨exp⟩ ⟨operator⟩ ⟨exp⟩
        |(⟨exp⟩)
        |⟨exp⟩ (⟨exp list option⟩)
⟨number⟩ ::= ⟨optional sign⟩ ⟨unsigned number⟩
⟨optional sign⟩ ::= +|−|⟨empty⟩
⟨unsigned number⟩ ::= ⟨digits⟩
                    |⟨digits⟩ ·
                    |⟨digits⟩ · ⟨digits⟩
                    |· ⟨digits⟩
⟨digits⟩ ::= ⟨digit⟩|⟨digits⟩ ⟨digit⟩
⟨digit⟩ ::= 0|1|2|3|4|5|6|7|8|9
⟨string⟩ ::= ⟨quoted string⟩
           |⟨bracket string⟩
⟨quoted string⟩ ::= '⟨quoted items⟩'
⟨quoted items⟩ ::= ⟨empty⟩
                 |⟨quoted items⟩ ⟨any character except quote⟩
⟨bracket string⟩ ::= [⟨bracket items⟩]
⟨bracket items⟩ ::= ⟨empty⟩
                  |⟨bracket items⟩ ⟨any character except brackets⟩
                  |⟨bracket items⟩ ⟨bracket string⟩
⟨identifier⟩ ::= ⟨letter⟩
               |⟨identifier⟩ ⟨letter⟩
               |⟨identifier⟩ ⟨digit⟩
⟨letter⟩ ::= A|B|C|D|. . .|X|Y|Z
  {identifiers do not include the reserved flow-of-control keywords}
⟨parameter⟩ ::= # ⟨digit⟩
              |# ⟨digit⟩ ⟨digit⟩
⟨operator⟩ ::= **|*|/|+|−|=|/=|
             ⟨|⟩|<=|>=|:=|;
⟨exp list option⟩ ::= ⟨empty⟩
                    |⟨exp list⟩
⟨exp list⟩ ::= ⟨exp⟩
             |⟨exp list⟩, ⟨exp⟩
⟨empty⟩ ::=
```

Cm*—A modular, multi-microprocessor†

by R. J. SWAN, S. H. FULLER and D. P. SIEWIOREK
Carnegie-Mellon University
Pittsburgh, Pennsylvania

ABSTRACT

This paper describes the architecture of a new large multiprocessor computer system being built at Carnegie-Mellon University. The system allows close cooperation between large numbers of inexpensive processors. All processors share access to a single virtual memory address space. There are no arbitrary limits on the number of processors, amount of memory or communication bandwidth in the system. Considerable support is provided for low level operating system primitives and inter-process communication.

INTRODUCTION

Cm* is an experimental computer system designed to investigate the problems and potentials of modular, multi-microprocessors. The initial impetus for the Cm* project was provided by the continuing advances in semiconductor technology as exemplified by processors-on-a-chip and large memory arrays. In the near future processors of moderate capability, such as a PDP-11, and several thousand words of memory will be placed on a single integrated circuit chip. If large computer systems are to be built from such chips, what should be the structure of such a "computer module?"

Initial versions of the Cm* architecture[1] grew in part as an extension to the modular design of systems from register transfer modules, or RTMs.[2] In addition there was substantial interest in the development of large multiprocessor systems such as Pluribus[3] and C.mmp.[4] Cm* is intended to be a testbed for exploring a number of research questions concerning multiprocessor systems, for example: potential for deadlocks, structure of inter-processor control mechanisms, modularity, reliability, and techniques for decomposing algorithms into parallel cooperating processes.

The structure of Cm* is very briefly described. There is a description of the address structure and discussion of the support given for the operating system. The use of the addressing structure for inter-process communication and control operations is discussed. A companion paper[5] discusses the various mechanisms used to implement the complex address mapping and routing structure of Cm*. Some results from the performance modelling of Cm* are also presented. A second companion paper[6] describes the structure of the basic operating system and support software.

THE STRUCTURE OF Cm*

There is a surprising diversity of ways to approach the interconnection of processors into a computing system.[7] The processors could be interconnected with several serial I/O links to form a computer network; they could be interconnected in a tight synchronous fashion to build an array processor; or the processors could be organized to share primary memory. This last approach, a multiprocessor organization, was chosen for Cm* because it offers a closer degree of coupling, or communication, between the processors than would a multicomputer or network configuration. Multiprocessors also have a more general range of applicability than other multiple processor systems.

During the development of the Cm* structure a wide variety of multiprocessor switch structures were considered.[8] ~~The basic structure selected is represented in Figure~~ 1. The essential feature which distinguishes it from other multiprocessor structures is that shared memory is not separated from the processing elements, but rather a unit of memory and a processor are closely coupled in each module and a network of buses gives a processor access to nonlocal memory. This structure allows modular expansion of the number of processors and memory modules without a rapid increase in the interconnection costs. Memory can be shared even though there is no direct physical connection between the requesting processor and the required memory. For example, consider a request by processor, P1, to the memory, M4, in Figure 1. The address mapping element, K1, directs the reference from P1 onto the inter-module bus. The address is recognized by K2, which directs it onto a second inter-module bus. The reference is finally accepted by K4, which accesses the request memory location and passes back an acknowledgment or data to the requesting processor. The need for high inter-module

† This work was supported in part by the Advanced Research Projects Agency under contract F44620-73-C-0074, which is monitored by the Air Force Office of Scientific Research, and in part by the National Science Foundation Grant GJ 32758X. The LSI-11's and related equipment were supplied by Digital Equipment Corporation.

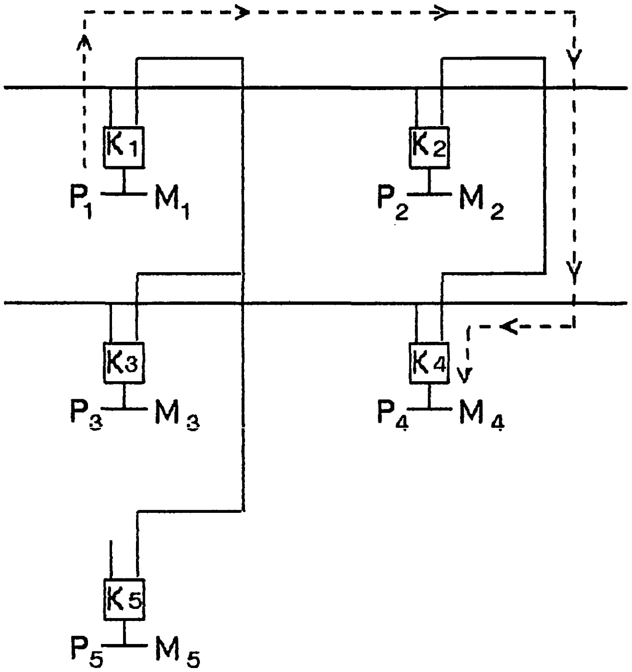


Figure 1—Canonical computer module structure

communication rates will be minimized if a large fraction of each processor's references to primary memory 'hit' the section of memory local to the processor. (Preliminary experiments in the fall of 1976 indicate that hit ratios of better than 90 percent can be expected provided that the code executed is normally held local to the processor.)

Deadlock with references to nonlocal memory

Almost all computer systems implement accesses from processor to primary memory with *Circuit Switching,* that is, a complete path is established from a processor to the memory being referenced. Circuit switching is not feasible for a structure like Cm* where local memory is also accessible as shared memory. Figure 1 shows the path used for P1 to access M4 via K2. Consider a concurrent attempt by P4 to access M1 via K2. With a circuit switch implementation, a situation could arise where P1 held its local memory bus and the bus connecting K2, while P4 also holds its own memory bus plus the bus connecting K4 to K2. Neither memory reference could complete without one processor first releasing the buses it holds. There are numerous situations where deadlock over bus allocation can occur. Resolving this deadlock requires, at the very least, a timeout and retry mechanism.

The alternative to circuit switching is *Packet Switching.* In a packet switched implementation, the address from the processor is latched at each level in the bus structure. Buses are not allocated for the full duration of a memory reference, but just for the time taken to pass a "packet," containing an address and/or data, from one node on the bus to another. Therefore packet switching allows significantly better bus utilization and significantly reduced bus contention in Cm*-like structures. The use of packet switching eliminates the possibility of deadlock over bus allocation but introduces the possibility of deadlock over buffer allocation.[1,9] Buffers, or intermediate registers, are resources which can be provided very cheaply, relative to providing additional inter-Cm buses, with present technology.

*The actual structure of Cm**

Design studies indicated that very little performance loss would result from combining several individual Computer Modules into a cluster and providing a shared address mapping and routing processor, Kmap, which allowed communication with other clusters. Because the cost of the Kmap is distributed across many processors it can be endowed with considerable flexibility and power at relatively little incremental cost. Because of its commanding position in the cluster, the Kmap can ensure mutual exclusion on access to shared data structures with very little overhead.

The full structure of Cm* is shown in Figure 2. Individual Computer Modules, or Cm's, consist of a DEC LSI-11 processor, an Slocal and standard LSI-11 bus memory and devices. The processor is program compatible with PDP-11s; thus a large body of software is immediately available. The prime function of the Slocal, or local switch, is to direct references from the processor selectively either to local memory or to the Map Bus, and to accept references from the Map Bus to the local memory.

Up to 14 Computer Modules and one Kmap form a cluster. The Kmap, or mapping processor, consists of three major components. The Kbus arbitrates and controls the Map bus. The Pmap is a horizontally microcoded 150 ns cycle time processor. The basic configuration has 1 K × 80 bits of writable control store and 5K × 16 bits of bipolar RAM for holding mapping tables, etc. The third level of the Cm* structure is provided by the intercluster buses which allow communication between clusters. The Linc provides the interface to two intercluster buses.

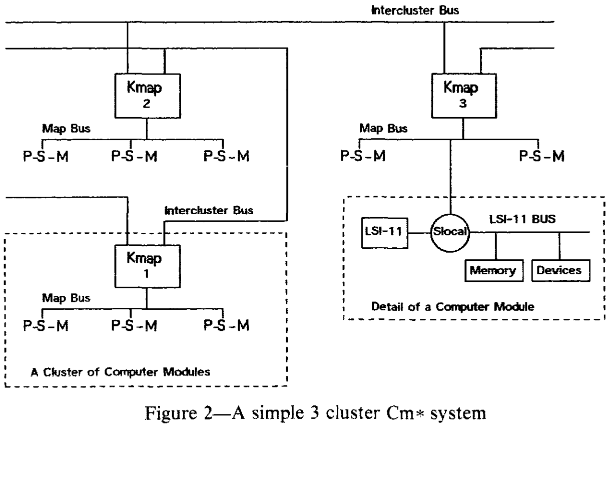


Figure 2—A simple 3 cluster Cm* system

There are no arbitrary limits to the size of a Cm* system. Memories, processors and Kmaps can be incrementally added to suit needs. Any processor can access any memory location in the system. The routing of a processor's reference to a target memory is transparent to the program, thus the system can be reconfigured dynamically in response to hardware failures.

ARCHITECTURE OF THE Cm* ADDRESS TRANSLATION MECHANISMS

Many of the more conventional aspects of the architecture of the Cm* system are consequences of using LSI-11's for the central processing elements. The organization and encoding of the instructions, interrupt and trap sequencing, and the 64K byte processor address space of a Cm* system are all a result of the PDP-11 architecture as implemented on the LSI-11. By selection of the LSI-11, however, we do not want to imply that the PDP-11 architecture is ideally suited to multiprocessor systems. The ideal solution would have been for us to have designed our own processors. However, practical considerations of time, money, and existing support software led us in early 1975 to recognize that by choosing the LSI-11 we could concentrate on those aspects of the Cm* architecture unique to multiprocessor systems. This section, and the following section on control structures, will discuss the Cm* architecture as we extended it beyond the standard PDP-11 architecture.

The addressing structure is one of the most important aspects of any computer architecture, it is even more significant when cooperation between multiple processors is to be achieved by sharing an address space. Denning[10] lists four objectives for a memory mapping scheme:

(a) Program modularity: the ability to independently change and recompile program modules.
(b) Variable size data structures.
(c) Protection
(d) Data and program sharing: allowing independent programs to access the same physical memory addresses with different program names.

For Cm*, where we are using processors with only a 64K byte address space, we must add the following requirement:

(e) Expansion of a processor's address space.

Cm* has a 2^{28} byte segmented virtual address space. Segments are of variable size up to a maximum of 4K bytes. There is a capability-based protection scheme enforced by the Kmap. The addressing structure provides considerable support for operating system primitives such as context switching and interprocess message transmission.

The path from processor to memory

The Slocal (see Figures 2 and 3) provides the first level of memory mapping. A reference to local memory is simply relocated, on 4K byte page boundaries, by the relocation table in the Slocal. As discussed above, it is assumed that most memory references will be made by processors to their local memory. Relocation of local memory references can be implemented with no performance overhead because the synchronous processor has sufficiently wide timing margins at the points where address relocation is performed. For segments which are not in a processor's local memory the relocation table has a status bit which causes the address to be latched, the processor forced off the LSI-11 bus, and a Service Request to be signalled to the Kmap. All transactions on the Map bus are controlled by the Map bus controller, or Kbus, which is a component of the Kmap. The address generated by the processor is transferred via the Map bus to the Pmap, the microprogrammed processor within the Kmap. If the reference is for memory within the cluster then the Pmap generates a physical address and sends it to the appropriate Slocal. If it is a write operation, data is passed directly from the source Slocal to the destination Slocal; the data does not have to be routed through the Kmap. The selected destination Slocal performs the requested memory reference and the processor in the destination Computer Module is not involved. When the reference is complete the Kbus transfers the data read from the destination Slocal directly back to the requesting processor via the Map bus and its Slocal.

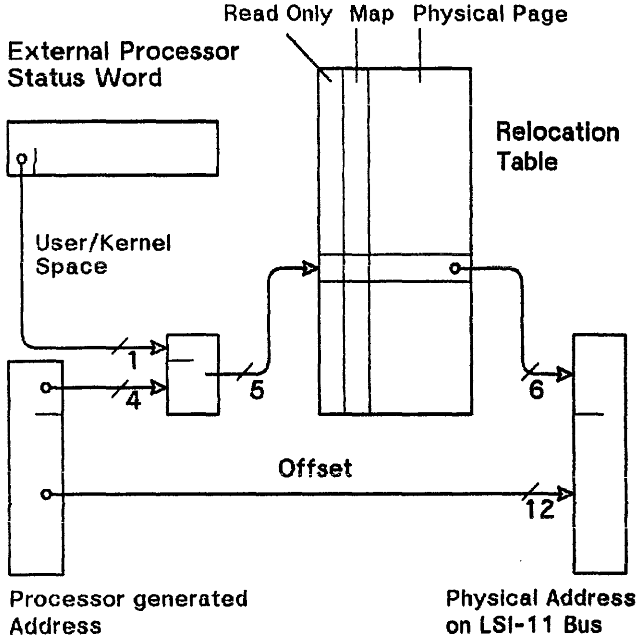

Read Only
Map
Physical Page
External Processor Status Word
Relocation Table
User/Kernel Space
1
4
5
6
Offset
12
Processor generated Address
Physical Address on LSI-11 Bus

Figure 3—Addressing mechanism for local memory references

If the processor references a segment in another cluster then the Pmap will transmit a request to the desired cluster via the Linc and the Intercluster buses. (See Figure 2.) If the destination cluster is not directly connected to the source cluster, that is, if it does not share a common intercluster bus, then the message will be automatically routed via intermediate clusters. When the message reaches the destination cluster, the memory reference is performed similar to a request from a processor within the cluster. An acknowledgment, or Return, message (containing data in the case of a read) is always sent back to the source cluster and subsequently to the requesting processor.

The addressing environment of a process

The virtual address space of Cm* is subdivided into up to 2^{16} *Segments*. Each segment is defined by a *Segment Descriptor*. The standard type of segment is similar to segments in other computer systems; it is simply a vector of memory locations. The segment descriptor specifies the physical base address of the segment and the length of the segment. Segments are variable in size from 2 bytes to 4 K bytes. However, other segment types may be more than simple linear vectors of memory; references to segments may invoke special operations. Segments may have the properties of stacks, queues or other data structures. Some segments may not have any memory associated with them, and a reference to the segment would invoke a control operation. For each segment type, up to eight distinct operations can be defined. For normal segments the operations are Read and Write. Conceptually, segments are never addressed directly; they are always referenced indirectly via a *Capability*. A capability is a two-word item containing the name of a segment and a *Rights* field. Each bit in the rights field indicates whether the corresponding operation is permitted on the segment.

To provide efficient support for context swapping, message-sending, etc., it is necessary for the Kmap microcode to understand some of the structure of an executable software module (variously called a process, activity, address space, etc.). Each executable software module is represented by an *Environment*, Figure 4. An environment is a three-level structure composed of segments. The first level in the structure is a *Primary Capability List*, CL[0]. The first entry in CL[0] is a Capability for a *State Vector*, which holds the process state while it is not executing on a processor. Entries CL[0](1) to CL[0](7) in the Primary Capability list may contain Capabilities for *Secondary Capability Lists* referred to as CL[1] through CL[7] respectively. The remaining entries in the Primary Capability List and all the entries in the Secondary Capability Lists contain Capabilities for segments which can be made directly addressable by the process when it executes. These may be code, data or any other type of segment. The provision of up to eight Capability Lists facilitates the sharing of segments and sets of segments by cooperating processes. A software module can only access those segments for which it has capabilities and perform only those operations permitted by the capabilities.

Virtual address generation

The processors in Cm*, LSI-11s, can directly generate only a 16 bit address. This 64 K byte address space is divided into 16 pages of 4 K bytes each. Each page provides a window into the system-wide 2^{28} byte virtual address space, (see Figure 5) and can be independently bound to a different segment in the virtual address space. The top page in the processor's address space, page 15, is reserved for direct program interaction with the Kmap. This mechanism is analogous to the I/O page convention in standard PDP-11s. In page 15 there are 15 pseudo registers, called *Window Registers*. These define the binding between page frames in the processor's immediate address space and segments in the virtual address space. This binding is done indirectly via capabilities. Each window register holds an index for a capability in the currently executing software module's capability list structure. A Capability List index consists of a three bit field to select one of the up to eight Capability Lists, plus an offset within the C-List.

To overlay the processor's address space, i.e. to change the mapping from page frame to segment in the virtual address space, a program simply writes a new capability index into the appropriate window register. This overlay operation is completely protected; the program can only reference segments for which it has a Capability. The act of writing the Capability index into the window register activates the Kmap. The Kmap retrieves the selected Capability from main memory and places it in its "Capability cache." The Kmap adjusts its internal tables so that subse-

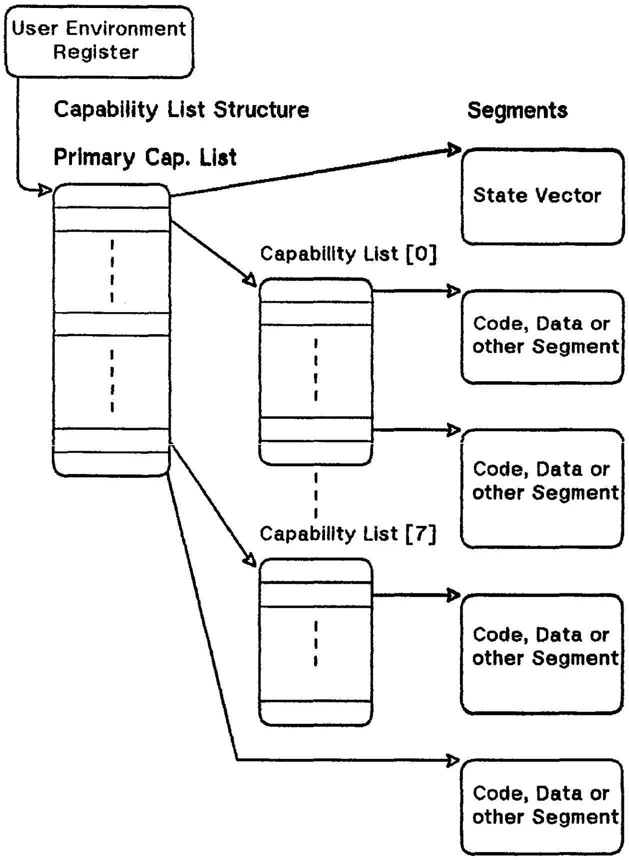

User Environment Register
Capability List Structure
Segments
Primary Cap. List
State Vector
Capability List [0]
Code, Data or other Segment
Code, Data or other Segment
Capability List [7]
Code, Data or other Segment
Code, Data or other Segment

Figure 4—The environment of a user software module

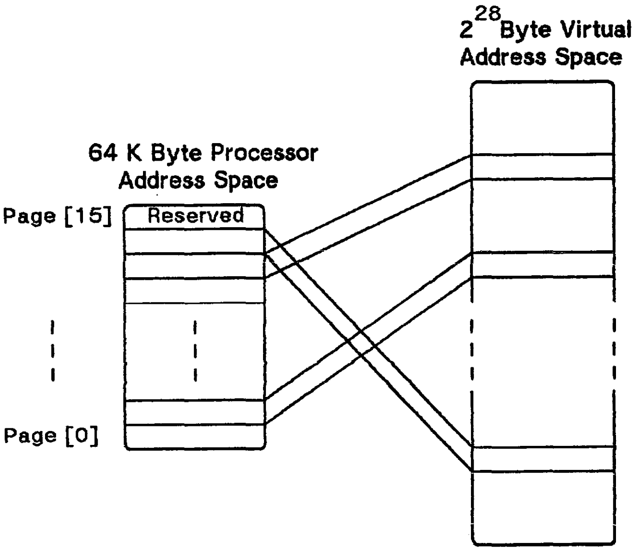

2^28 Byte Virtual Address Space
64 K Byte Processor Address Space
Page [15]
Reserved
Page [0]

Figure 5—Windows from the processor's immediate address apce to the virtual address space

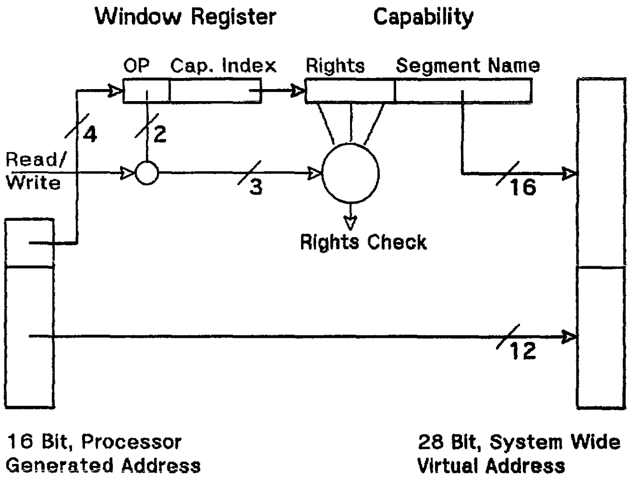


Figure 6—Conceptual virtual address generation and rights checking

quent references to the page frame will map to the segment specified by the Capability. If the segment is local to the processor then the Kmap may also change the relocation register in the Slocal so that references to the segment can be performed at full speed without the intervention of the Kmap. The Slocal, for cost and performance reasons, does not have the hardware necessary for bounds checking on variable sized segments. Thus only fixed size 4 K byte segments can be accessed without Kmap assistance.

The Cm* mechanism for address space overlaying should be contrasted with mechanisms in other computer systems. When executing a large program on a processor with a small immediate address space, the time taken to overlay the address space can have a crucial effect on performance. Measurements made of the execution of the operating system HYDRA on the C.mmp multiprocessor showed that relocation registers were being changed approximately every 12 instructions. (This does not, however, imply that user programs perform overlay operations this frequently.) Within the operating system this overlay operation is a single PDP-11 MOVE instruction because no protection is involved. However for user programs running under HYDRA, an overlay operation requires invocation of the operating system with several hundred instructions of software overhead. Subsequent optimization, and partial microcoding, have greatly reduced this overhead.

Figure 6 shows the conceptual translation from a 16 bit processor-generated address to a virtual address. The four high order address bits from the processor select one of 15 Window registers. The Window register holds an index for a Capability in the executing software modules Capability List structure. The 16 bit segment name from the selected Capability is concatenated with the 12 low order bits from the processor to form a 28 bit virtual address. Figure 6 also shows the read/write indicator from the processor being concatenated with two bits in the address expansion registers to form a three bit opcode. The correspondind bit in the Capability rights field is selected and tested. If the operation is not permitted then an error trap is forced.

Virtual to physical address mapping

The mapping from virtual to physical address depends on the location of the segment in the network and, of course, on the type of the segment. We begin with the case of a simple read/write segment residing within the same cluster as the processor referencing the segment. This mapping is shown in Figure 7. The segment name is used to access the corresponding segment descriptor. The descriptor provides

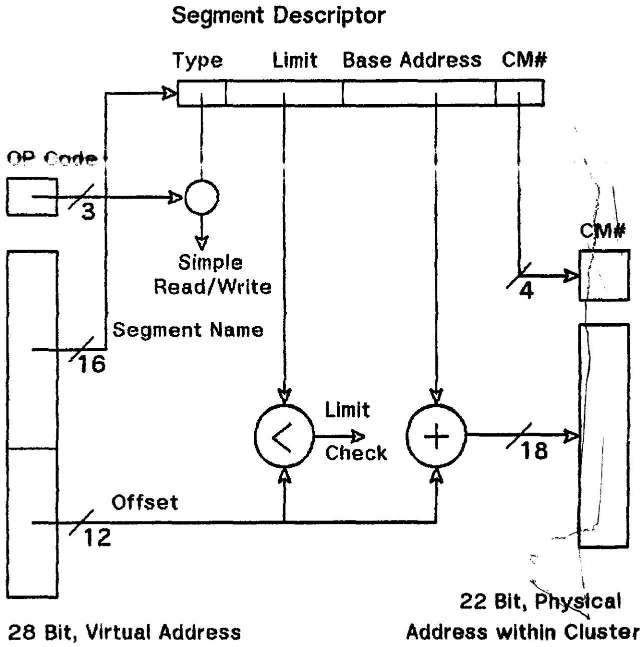


Figure 7—Virtual to physical address mapping for a variable sized segment

a limit value which is checked against the 12 bit offset in the virtual address. If the reference is out of the bounds of the segment then an error trap occurs. The offset is added to the physical base address from the descriptor. The resulting 18 bit value is a physical address within the 256 K byte address space of the computer module also specified in the descriptor.

If the virtual address references a segment outside the source cluster then the segment name is used to access an *Indirect Descriptor Reference* rather than the descriptor itself. The indirect reference simply indicates in which cluster the segment resides. The Kmap then passes the virtual address to that cluster via the inter-cluster buses. An alternative approach would be to have duplicate copies of the segment descriptors in every cluster. Thus the virtual-to-physical mapping could be done at the source cluster, with possibly some savings in overhead. However, any attempt to change the virtual-to-physical binding of a segment (e.g., moving it to a different memory module or onto backing store) would require an effectively simultaneous change to all copies of the segment descriptor. In a large network this operation would be slow and cumbersome, if not impossible. A further advantage to ensuring that only a single descriptor exists for each segment is that a *Lock Bit* can be provided in the descriptor. The lock bit can be used to ensure mutual exclusion for special segment operations.

The kernel address space

Each processor can execute in either of two address spaces. One is the *User Address Space* which was described above. The second is the *Kernel Address Space,* which is similar to a user address space with the addition of some mechanisms reserved for the operating system. The currently executing address space is selected by a bit in the Processor Status Word of the LSI-11. A *Kernel Environment* is similar to a User Environment; however segments at the third level of the Capability List structure (Figure 4) can be User Primary Capability Lists. That is, a Kernel Capability list structure can have user environments as substructures.

There are several additional pseudo registers provided in page 15 of the kernel address space. One of these, the *User Environment* register, holds an index for a Capability in the kernel environment which points to a user environment. This register specifies the current user environment for this processor. If the kernel writes a new index into the register the addressing state of the old user process is saved by the Kmap in the state vector part of the old user environment. The addressing state of the new user is then loaded from the specified new user environment. The addressing state is the value of the window and other system registers in page 15 of the executing program. Ideally, this operation, which performs a context swap by saving one addressing state and loading another, would also save the internal processor registers. Unfortunately there is no way for the Kmap to access the internal registers of an LSI-11. Thus internal registers must be saved and restored under program control.

THE USE OF THE ADDRESSING STRUCTURE FOR CONTROL OPERATIONS

The philosophy in Cm* is to implement all special control operations, such as interprocessor interrupts, by references to the physical address space. This not only avoids a proliferation of special control signals, but also allows the power of the system's address mapping and protection mechanisms to be applied to control operations.

The Slocal provides a three priority level interrupt scheme. An interrupt is invoked by writing into the appropriate physical address on the LSI-11 bus of the target processor. Thus an interrupt can be requested by a process anywhere in the network, provided the process has a Capability for a segment which maps to the correct physical address. Another example is the abort operation. If the appropriate bit is written, a NXM (Non Existent Memory) trap by the local processor is forced. This mechanism will be used when an error occurs during a remote reference by the processor.

The following examples show how references to special typed segments, or special operations on standard segments, are used to invoke microcoded operations in the Kmap.

Primitive lock operations

For processors in the PDP-11 family, most write operations are part of a read-modify-write sequence. In standard PDP-11s (including LSI-11's) this sequence is implemented as an indivisible, single bus operation. This improves performance by reducing bus overhead and allowing optimization of references to memory with destructive read operations (e.g., core and dynamic MOS memory). In C.mmp the indivisibility of these operations is maintained through the switch to shared memory. This allows the implementation of Locks and Semaphores because a memory location can be both tested and set without fear of an intervening access by some other processor. Indivisible read-modify-write operations to nonlocal memory will not be implemented in Cm* because of increased bus and memory contention and hardware complexity. We will provide an equivalent function by making use of the Kmap's ability to lock a segment descriptor while it makes a series of references to the segment. To implement a basic lock mechanism two special segment operations are defined:

Inspect the word addressed. If greater than zero, then decrement. Return the original value.

Increment the word addressed. Return the original value.

An inter-process message system

Message systems can provide particularly clean mechanisms for communication between processes.[11,12] In the past, a drawback to message systems has been the substantial operating system overhead in transferring a message from one process to another in a fully protected way. The architecture of Cm* provides an opportunity to build a fully protected message system which can be used with very low overhead.

A message port, or mail box, will be a special segment type. Messages will either be entire segments, passed by transferring capabilities, or will be single data words encoded as data capabilities. Two representative operations on Mailbox segments are:

Send(Message, ReplyMBox, MailBox)
This transfers capabilities for a message and a reply mail box from the caller's Capability List to the Mailbox. If the Mailbox is full then the caller is suspended.

Receive(MailBox)
If the mailbox contains a message then a Capability for the message and a Reply Mailbox will be transferred into the caller's Capability List. Otherwise the caller is suspended.

Provided that the above operations are successful, they are performed completely in Kmap microcode, and messages may be passed with probably less than 100 microseconds delay. If the operation cannot be completed because the Mailbox is full or empty, then the operating system is invoked to suspend the requesting process. The Kmap can also request the operating system to wake up a suspended process when the operation is complete.

DEVELOPMENT AIDS

The development of hardware and software for a new computer system is a major undertaking. We have attempted to ease this burden by using a variety of aids. All the major hardware components were drafted using an interactive drawing package (a version of the Standard Drawing Package). To facilitate the development of software, prior to the availability of hardware, a functional simulation of Cm* was programmed, which executes on C.mmp. Development of the Kmap hardware and microcode has been greatly benefited by the use of the "hooks" mechanism in the Kmap. This connection to the Kmap allows a program executing on an LSI-11 almost complete access to the internal state of the Kmap.

In order to expedite hardware debugging and software development, a host program development system was constructed. The host is connected to each Cm in the system by a Serial Line Unit (SLU) to allow down line memory loading and dumping from the associated Cm. In addition, the SLU makes console control functions for each LSI-11 available to the host computer.[16] The Host in turn is connected to a PDP-10 timesharing system.

CONCLUDING REMARKS AND PROJECT STATUS

Cm* is projected to be constructed in three stages. The first stage is a ten-processor, three Kmap system. The subsequent stages will include 30-processors and later 100-processors. Detailed hardware design began in late July, 1975. As of late summer, 1976, a three-processor, one-Kmap system was operational. It is expected that the first stage Cm* configuration will be operational in the second quarter of 1977. The initial operating system is described in Reference 6 and is being developed both on the Cm* simulator which runs on C.mmp and on the real hardware with the support of the Host Development system.

The essential features of the Cm* architecture have been presented. Both the coupling of a processor directly with each unit of shared memory and the three level bus structure which makes all memory accessible by every processor are primary features of the Cm* structure. Much of the sophistication in the architecture is associated with the address translation mechanisms. A description has been given of how the small processor address space of the PDP-11 is mapped into the larger global virtual address space of the Cm* system and how the global virtual address space is mapped onto the distributed physical address space of the Cm* system. A number of important aspects of the Cm* project are outside the scope of this paper and interested readers are referred to other papers for a more complete discussion.[5,6,8,9,13–15] Reliability and performance models have been developed concurrently with the hardware design of the system and have been used to guide several important decisions concerning the structure of the Cm* implementation.

ACKNOWLEDGMENTS

During the years its its initial development, many individuals have contributed to this project. Gordon Bell, Bob Chen, Doug Clark and Don Thomas contributed ideas to earlier versions of this architecture. Anita Jones and Victor Lessor have contributed to the present architecture. Miles Barel, Paulo Corrulupi, Levy Raskin and Paul Rubinfeld have all contributed to bringing the hardware to an early fruition. Kwok-Woon Lai and John Ousterhout are largely responsible for the successful development of the Kmap. Andy Bechtolsheim designed the Linc. Lloyd Dickman, Rich Olsen, Steve Teicher and Mike Titelbaum at Digital Equipment Corporation have provided information, ideas, and support critical to the success of the project.

REFERENCES

1. Fuller, S. H., D. P. Siewiorek, and R. J. Swan, "Computer Modules: An Architecture for Large Digital Modules," *Proceedings of the First Annual Symposium on Computer Architecture,* University of Florida, Gainesville. Also in ACM SIGARCH, *Computer Architecture News,* Vol. 2, No. 4, December 1973, pp. 231–236.
2. Bell, C. G., J. L. Eggert, J. Grason, and P. Williams, "The Description and the Use of Register Transfer Modules (RTMs)," *IEEE Transactions on Computers,* Vol. C-21, No. 5, May 1972, pp. 495–500.
3. Heart, F. E., S. M. Ornstein, W. R. Crowther, and W. B. Barker, "A New Minicomputer/Multiprocessor for the ARPA Network," *AFIPS Conference Proceedings,* Vol. 42, NCC 1973, pp. 529–537.

4. Wulf, W. A. and C. G. Bell, "C.mmp—A Multi-Mini-Processor," *AFIPS Conference Proceedings,* Vol. 41, part II, FJCC 1972, pp. 765-777.
5. Swan, R. J., A Bechtolsheim, K. Lai and J. Ousterhout, "The Implementation of the Cm* Multi-Microprocessor," *AFIPS Conference Proceedings,* Vol. 46, 1977 National Computer Conference.
6. Jones, A. K., R. J. Chansler, I. Durham, P. Feiler and K. Schwans, "Software Management of Cm*, a Distributed Multiprocessor," *AFIPS Conference Proceedings,* Vol. 46, 1977 National Computer Conference.
7. Anderson, G. A. and E. D. Jensen, "Computer Interconnection Structures: Taxonomy, Characteristics and Examples," *Computing Surveys,* 7, 4, December 1975, pp. 197-213.
8. Swan, R. J., S. H. Fuller and D. P. Siewiorek, "The Structure and Architecture of Cm*: A Modular, Multi-Microprocessor," *Computer Science Research Review* 1975-76, Carnegie-Mellon University, Department of Computer Science, Pittsburgh, Pa., December 1976, pp. 25-47.
9. Swan, R. J., L. Raskin, and A. Bechtolsheim, "Deadlock Issues in the Design of the Linc," Internal Memo, March 1976.
10. Denning, P. J., "Virtual Memory," *Computing Surveys,* Vol. 2, No. 3, September 1970, pp. 153-190.
11. Brinch-Hansen, *Per, Operating System Principles,* Chapter 8, "A Case Study: RC-4000," Prentice Hall, 1973.
12. Jefferson, David, "The Hydra Message System," to be published.
13. Ingle, Ashok and D. P. Siewiorek, "Reliability Modeling of Multiprocessor Structures," *Proceedings IEEE CompCon '76,* September 1976.
14. Ingle, Ashok and D. P. Siewiorek, "Reliability Models for Multiprocessor Systems with and without Periodic Maintenance," Computer Science Technical Report, Carnegie-Mellon University, September 1976.
15. Siewiorek, D. P., W. C. Brantley Jr., and G. W. Lieve, "Modeling Multiprocessor Implementations of Passive Sonar Signal Processing," Final Report, Carnegie-Mellon University. Pittsburgh, Pa. 15213, October 1976.
16. Van Zoren, H., "Cm* Host User's Manual," Department of Computer Science, Carnegie-Mellon University, December 1975.
17. Bell, C. G., R. C. Chen, S. H. Fuller, J. Grason, S. Rege, and D. P. Siewiorek, "The Architecture and Applications of Computer Modules: A Set of Components for Digital Design," *IEEE Computer Society International Conference,* CompCon 73, March pp. 177-180.
18. Bell, C. G. and A. Newell, *Computer Structures: Readings and Examples,* McGraw-Hill, New York, New York, 1971.

The implementation of the Cm* multi-microprocessor[†]

by RICHARD J. SWAN, ANDY BECHTOLSHEIM, KWOK-WOON LAI and JOHN K. OUSTERHOUT
Carnegie-Mellon University
Pittsburgh, Pennsylvania

ABSTRACT

The implementation of a hierarchical, packet switched multiprocessor is presented. The lowest level of the structure, a Computer Module, is a processor-memory pair. Computer Modules are grouped to form a cluster; communication within the cluster is via a parallel bus controlled by a centralized address mapping processor. Clusters communicate via intercluster busses. A memory reference by a program may be routed, transparently, to any memory in the system. This paper discusses the hardware used to implement the communication mechanism. The use of special diagnostic hardware and performance models is also discussed.

INTRODUCTION

The companion paper[1] has introduced Cm* as a large, extensible multiprocessor architecture. It has an unusually powerful and complex addressing structure which allows close, protected cooperation between large numbers of inexpensive processors. This paper describes the combination of hardware and firmware which implements the address space sharing and interprocessor communication mechanisms.

Cm* is a multiprocessor system as we define it (rather than a network of independent computers) because the processors share a common address space. All processors have immediate access to all memory. The structure of Cm* is shown in Figure 1. The primary unit is the *Computer Module* or Cm. This consists of a processor, memory and peripherals interfaced to a local memory bus and a "local switch." The local switch, or *Slocal,*[‡] interconnects the processor, its local memory bus and the *Map Bus*. The Map Bus provides communication between up to fourteen Computer Modules within a *cluster,* and is centrally controlled by the *Kmap,* a high performance microprogrammed processor. Each Kmap interfaces to two *Intercluster busses,* by means of which it communicates with the other clusters in the system.

There is a system-wide 28 bit virtual address space. This address space is divided into *segments* with a maximum size of 4096 bytes. Programs refer to segments indirectly via *Capabilities,* which are two-word items containing the global name of a segment and specifying access rights to the segment. The processors have a 16 bit address space which is divided into 16 pages. A mechanism is provided which allows a program to associate any Capability it possesses (and hence any segment to which it is allowed access) with any page in its immediate address space. A full description of the address mapping scheme is given in Reference 1.

To demonstrate the viability of a structure it is necessary to build a pilot system with currently available components. To be a successful demonstration, the pilot system has to be a useful, economical computing resource in its own right. Therefore, in the Cm* network described here, many design tradeoffs were made on the basis of current technology and the resources available. The highly experimental nature of the project encouraged an emphasis on generality and ease of debugging in the hardware components, rather than just minimization of costs. There are many aspects of the detailed design which would have to be re-evaluated if the structure were to be implemented in a different technology or built as a commercial product. In particular the distribution of functions between the processors and the Kmap would be carefully reconsidered. The modular nature of Cm* makes it particularly suitable for implementation in LSI.

The second section of this paper illustrates the mechanism for memory references. The various hardware components of Cm* are described in the following six sections. The third section describes the processor-memory pairs and their interface to the Map Bus. In the fourth section opportunities for parallelism in the address mapping mechanism are considered. Three autonomous functional units of the Kmap are presented in later sections, and describes the

[†] This work was supported in part by the Advanced Research Projects Agency under contract number F44620-73-C-0074, which is monitored by the Air Force Office of Scientific Research, and in part by the National Science Foundation Grant GJ 32758X. The LSI-11's and related equipment were supplied by Digital Equipment Corporation.

[‡] The names used for hardware components of Cm* are derived from PMS notation.[2] The leading, capitalized letter indicates the primary function of the unit, e.g., Computer, Processor, Kontroller, Link, Switch. The subsequent letters, optionally separated with a period, give some attribute of the unit. For example, Slocal is a local switch. Pmap is a mapping processor. The name Cm* derives from (Computer.modular)* where * is the Kleene star.

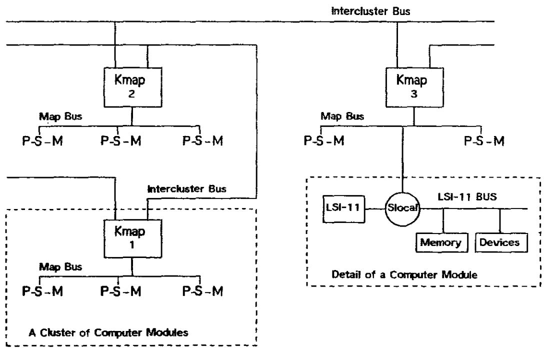


Figure 1—A simple 3 cluster Cm* SYSTEM

support given to hardware diagnosis and microcode development in the Kmap. For an effective implementation it was necessary to find a reasonable performance balance between system components. Some of the performance modelling which guided our judgment is presented in the last section.

THE MECHANISM FOR LOCAL AND NONLOCAL REFERENCES

Addresses generated by processors in a Cm* system may refer to memory anywhere within the system. Mapping of an address and routing to the appropriate memory are performed in a way that is totally transparent to the processor generating the address. If an address is to refer to the memory local to that processor, the memory reference is performed in a completely standard way except that the Slocal relocates the high-order four bits of the address. (See Figure 2.)

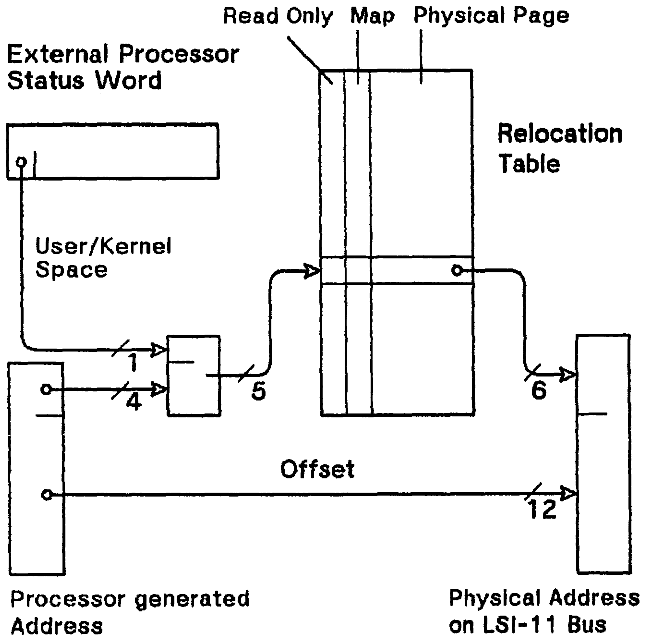


Figure 2—Addressing mechanism for local memory references

When the page being referenced is not local (i.e., the "Map" bit for the referenced page is set in the Slocal) a *service request* is made to the Kmap by the Slocal. Upon receiving the service request the Kmap executes a Map Bus cycle to read in the processor-generated address from the Slocal, as well as the number of the Cm making the request, and two status bits indicating which address space was executing on the processor and whether the reference was a read or a write (see Figure 3). If the segment being referenced is local to the cluster, the Kmap will use information cached in its high-speed buffers to bypass most of the processor-to-virtual-to-physical address mapping. Thus it can quickly translate from the page number referenced by the processor to a physical address consisting of the number of the Cm containing the physical location and an eighteen-bit local address. A second Map Bus transaction is executed to pass this address, and a bit indicating whether a read or a write is to be performed, to the destination Slocal. If the operation is a write, the data may be passed directly from the Cm making the reference to the Cm containing the word to be written. The destination Slocal performs the read or write via a Direct Memory Access. When this is completed it issues a *return request* to the Kmap to acknowledge completion. A third Map Bus cycle is performed to transfer the data back to the processor that made the reference (in the case of a read) and to acknowledge completion of the reference so that the requesting processor may resume activity.

A second alternative when the Kmap receives an address to map is that the physical location being referenced is not local to the cluster. In this case the information cached in the Kmap for the page being referenced will not indicate a physical location directly; instead it will give a sixteen-bit segment name, the number of the cluster containing the physical memory allocated to the segment, and two bits used to extend the read/write bit to a three-bit *op code.*

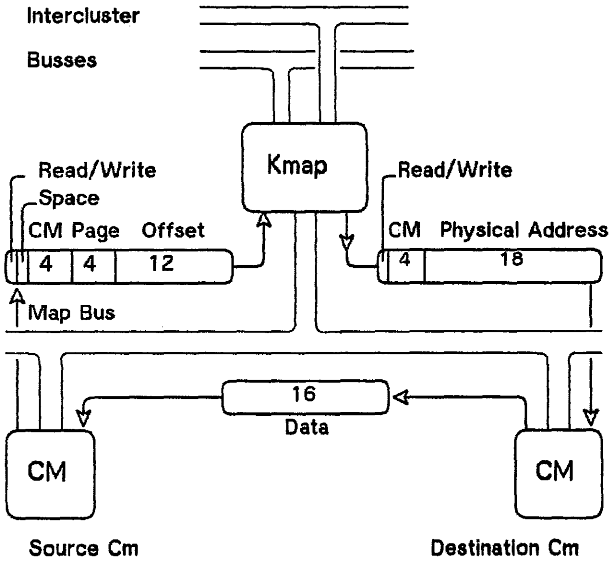


Figure 3—The mechanism for cluster-local references

This information is combined with the twelve low-order bits of the original processor address to form the full virtual address of the object being referenced. (See Figure 4.) The virtual address, along with the processor data (if a write is being performed) is sent via an Intercluster Bus to the Kmap of the cluster containing the segment (if there is no Intercluster Bus directly connecting the two Kmaps the message will be steered from Kmap to Kmap until it reaches the destination cluster). The destination Kmap will then map the virtual address to a physical one within its cluster. Map Bus transactions will be executed to pass the physical address (and data if needed) to an Slocal which in turn performs the operation and returns acknowledgment (and, perhaps, data) back to the destination Kmap. A return message is used to pass back acknowledgment and data to the Kmap of the originating cluster. Finally, this Kmap will relay the data and acknowledgment back to the initiating Cm to complete the reference.

Several points are worth noting with respect to the above schemes. Except at the local memory bus level, where conventional circuit switching is used, all communication is performed by *packet switching*. That is, buses are allocated only for the period required to transfer data. The data is latched at each interface, rather than establishing a continuous circuit from the source to the destination. This approach gives greater bus utilization and avoids deadlock over bus allocation. All transactions are completely interlocked with positive acknowledgment being required to signal completion of an operation (it is possible to allow a processor executing a nonlocal write to proceed as soon as the data for the write has been received by the Kmap or destination Slocal, without waiting for completion of the operation; however in this case the Kmap will expect to receive acknowledgment in place of the processor so that appropriate actions may be taken if none is received). The complete processor-to-virtual-to-physical address mapping is performed only in the case of intercluster references. As the locality of a reference increases the amount of this mapping that may be bypassed (and hence the speed of the reference) increases, with local caches of certain mapping information used to effect the bypass. An important characteristic of the addressing structure is that there is exactly one Kmap that may perform the virtual-to-physical mapping for a given segment. The requirement that all references to a segment occur with the cognizance of a single Kmap greatly simplifies the moving of segments and the implementation of operations requiring mutual exclusion.

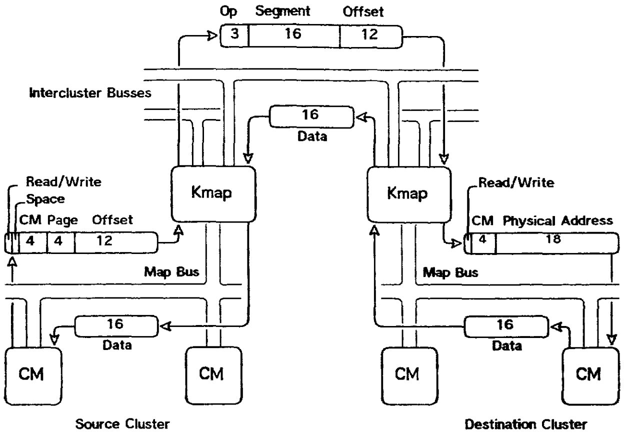


Figure 4—The mechanism for intercluster references

THE COMPUTER MODULE

The first level of the Cm* network hierarchy is the *Computer Module*, or Cm. The Cm's provide both the memory and processing power for the multiprocessor system.

The decision to use a standard, commercially available processor (the DEC LSI-11) has had a considerable impact on the design. Use of a standard instruction set has made a large pool of software and software development aids directly available. The not inconsiderable effort to design and implement a new processor has been avoided.

At the software level, the prime disadvantage of the LSI-11 instruction set is that only 16 bit addresses can be directly manipulated. The companion architecture paper discusses in detail the mechanism used to expand a processor's address space from 16 bits to 28 bits.

The components of a computer module

A Computer Module, Figure 5, can act as a stand alone computer system. The standard commercially available components include the DEC LSI-11 processor and dynamic MOS memory. Any LSI-11 peripheral may be used on the bus, including serial and parallel interfaces, floppy and fixed head disks, etc. The standard 16 bit memory has been extended with byte parity. Memory refresh is normally performed by microcode in the LSI-11; however, the fact that a processor may be suspended indefinitely while awaiting the completion of a complex external reference has made it necessary to augment each Cm with a special bus device to perform refresh.

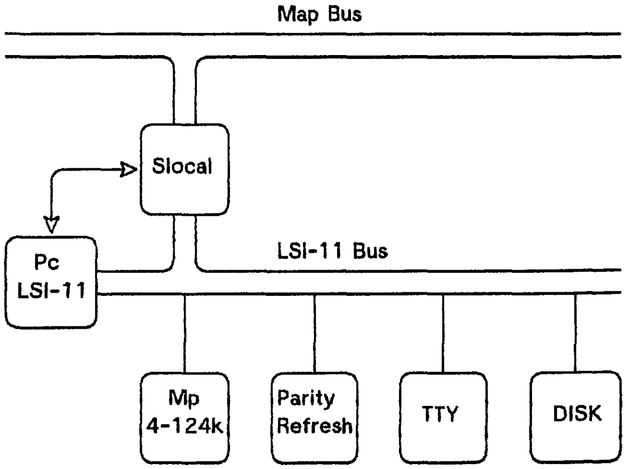


Figure 5—Details of a computer module

The most important component which has been added to each Cm is the Slocal. This provides the interface between the processor, the Map Bus and the LSI-11 Bus. The prime function of the Slocal is to selectively pass references from the processor to either the LSI-11 Bus or the Map Bus and to accept references from the Map Bus to the LSI-11 Bus. The Slocal also provides simple address relocation for references made by its processor to local memory. Figure 2 shows how this relocation is performed; the "Map Bit" in the local relocation table is set for pages which are not in the local memory of the processor.

In addition to the Local Relocation Table the Slocal provides a number of other control registers. All these registers are addressable as memory locations on the LSI-11 bus; however only the Kmap and highly privileged system code will have direct access to them. One of the key registers is the *eXternal Processor Status Word* (XPSW⟨15:8⟩). The LSI-11 implements only the low order byte of the standard PDP-11 *Processor Status Word* (PSW⟨7:0⟩). Logic in the Slocal (with assistance of standard signals from the LSI-11) allows the XPSW to be saved and restored during interrupt, trap and other operations in unison with the internal PSW. The XPSW allows selective enabling of various Slocal functions and controls a simple three level interrupt scheme. On power-up the XPSW is cleared, which disables all special operations by the Slocal including the relocation of local memory references. In this mode the processor acts as a bare, unmodified LSI-11. The Local Relocation Table can be initialized either by console operations, execution of local bootstrap code or remotely by any processor in the network. After initialization, enabling Reloc Mode (XPSW⟨11⟩) will allow local relocation and give access to the rest of the network.

Incorrect use of PDP-11 instructions such as HALT, RESET, Move-To-Processor-Status-word, Return from Interrupt, etc., can cause loss of a processor, garbling of an I/O operation or enable circumvention of the system's protection scheme. The Privileged Instruction Mode bit (XPSW⟨13⟩) enables logic in the Slocal which detects the fetching of any "dangerous" instruction. An immediate error trap is forced if an unprivileged program attempts to execute a privileged instruction.

Several registers in the Slocal are concerned with providing diagnosis and recovery information after a software or hardware error is detected. Almost all errors are reported to the processor by forcing a NXM (Non eXistent Memory) trap. This includes errors detected by the Kmap during remote references. The Kmap signals the error by writing to the "Force NXM" bit in an addressable register in the Slocal. The Local Error Register indicates the nature of the error and whether the erroneous reference was mapped. The "Last Fetch Address" register is updated to hold the address of the first word of an instruction every time the LSI-11 fetches a new instruction. If an error is detected, this register is frozen until the Local Error Register is explicitly cleared. Also frozen in the Local Error Register is a count of the number of memory references performed in the execution of the instruction. In conjunction, these two registers provide sufficient information to restore the state of the LSI-11 for retry of the instruction during which the error was detected.

The Slocal also provides two interrupt request registers. Interrupt enable bits in the external processor status word allow masking of the interrupt requests. Provided reference is permitted by the memory protection scheme, any processor in the network can interrupt any other processor simply by writing to the correct address.

Data paths for nonlocal references

An idealized form of the basic data paths and latches within a Cm* cluster is shown in Figure 6. Depending on the address generated, a reference from the processor is passed either to the local memory bus or to the Map Bus. A local memory reference is performed in a conventional way. For a nonlocal reference, the address (and possibly data) is latched and serviced request is issued to the Kmap. The broken line in Figure 6 shows the path of a read to the memory of another Cm in the cluster. The address from the source processor is read by the Kmap which translates it into a physical address within the memory of a Computer Module. This physical address is placed onto the Map Bus by the Kmap and latched at the target Cm. A conventional Direct Memory Access (DMA) cycle is performed by the destination Slocal, the data read is latched and the Kmap is again requested, this time with a return request. To complete the operation, the Kmap responds by transferring the data over the Map Bus from the target Cm to the requesting Cm (this simply requires the latch at the target Cm to be enabled onto the Map Bus and the latch at the requesting Cm to be strobed). At this point the source processor, which was suspended, is given the data as if a normal memory reference had been performed.

This simplified description of a Computer Module has been presented to emphasize the simplicity of the basic mechanisms required for an intra-cluster reference in Cm*. In the actual implementation using the LSI-11 processor the data paths are rather different than the idealized structure shown in Figure 6. The differences are due primarily to the need to minimize the changes to the LSI-11. Although still simplified, Figure 7 is a more accurate representation of the

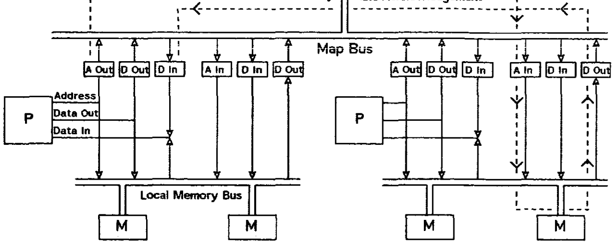


Figure 6—An idealized and simplified representation of the data paths in a cluster

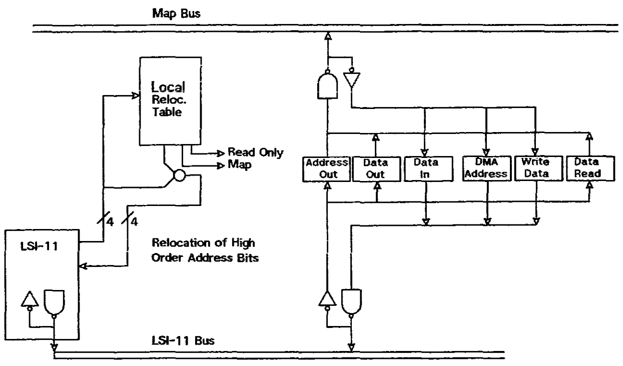


Figure 7—Simplified LSI-11—Slocal data paths

data paths and latches used to interface the LSI-11 and the LSI-11 bus to the Map Bus.

The processor board is modified so that the Local Relocation Table in the Slocal can be inserted in the data path of the four high order address bits. The timing margins in the processor's address path are wide enough to allow insertion of this delay without loss of performance. The LSI-11 Bus is the only data path from the processor for both local and nonlocal references. If the processor were permitted to hold the LSI-11 bus while waiting for completion of a nonlocal reference then references from other processors in the network to memory on the LSI-11 bus would be blocked. This could very easily lead to deadlock situations. To give greater concurrency and to eliminate the deadlock potential, the Slocal is able (using simple microcoded state sequence logic) to force the processor off the LSI-11 bus while it is waiting for completion of nonlocal references. While the processor is forced off the local bus the Slocal takes over DMA bus arbitration for the suspended processor.

CONCURRENCY WITHIN THE MAPPING MECHANISM

Early in the design of Cm* the speeds of the various components in the system began to appear as follows: the time for a "typical" Map Bus transaction was about 0.5 microseconds; the time required in the computational unit of the Kmap for an address mapping was 1-2 microseconds; the time to transfer a message on an Intercluster Bus was 2-4 microseconds; and the time for an Slocal to execute a read or write requested by the Kmap was 3-4 microseconds. In referring to the mechanisms for nonlocal mappings it can be seen that no single component is responsible for a very large fraction of the time required for a nonlocal reference. Thus if each cluster had a mapping concurrency of one (only one nonlocal reference could be processed at a time per cluster) both the utilization of the mapping components and the throughput of the mechanism would be low (the effect of concurrency on system performance is discussed quantitatively in a later section). In addition the possibility of deadlock in intercluster references is introduced.

The solution adopted for Cm* was to separate the four functions whose timings are given above and to allow a concurrency of eight in the mapping mechanism of each cluster. The packet-switched nature of Cm* yields cleanly to this approach, and requires only that queues be implemented to store messages at the interface between the components. Figure 8 depicts this structure, in which the Kmap has been logically sub-divided into three separate units: the *Kbus*, which is master of the Map Bus and controls all transactions on it; the *Pmap*, or mapping processor, which does all the address translation and maintains the cache used to speed up mapping; and the *Linc*, or intercluster link, which presides over the transmission of messages between clusters.

One other notion must be introduced before proceeding to a detailed discussion of the components of the Kmap, namely that of a *context*. Operations requiring mutual

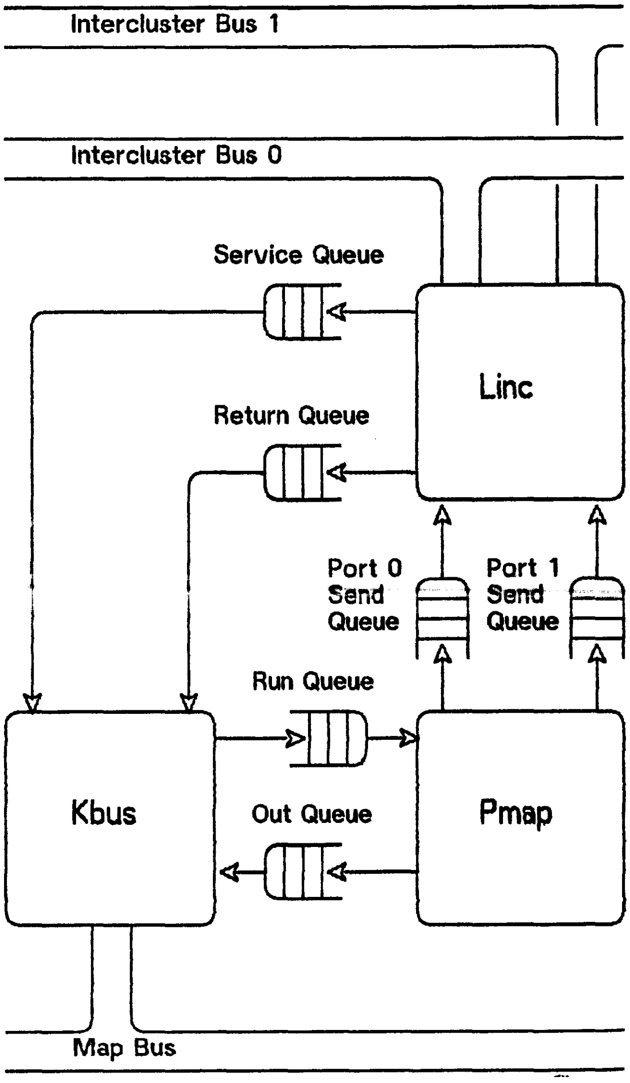


Figure 8—The components of the Kmap

exclusion (for example, changing the virtual-to-physical mapping of the system) will be implemented in Cm* as memory references to "special" segments which will then cause the Kmap to perform the desired operations in a protected way. In general these operations will require several references by the Kmap to main memory. If the Pmap is to be used for other mappings while these main-memory references are being made by the Kbus and Slocals, there must be some means of saving and restoring its state so that processing can be resumed when the memory reference has been completed. The solution adopted is to provide registers in the Kmap to save and restore state for up to eight overlapping operations. A mapped operation in some stage of processing by the Kmap is referred to as a *context.* Each context has allocated to its exclusive use eight general-purpose registers and four subroutine linkage registers (one of which is used to save the microprogram address while awaiting the completion of Map Bus transactions).

The Kbus maintains the status of the eight Pmap contexts and allocates them to new service requests. The context number and other status are then placed in the *Run Queue* to signal the Pmap that the context is runnable. The mapping processor activates the context by removing its number from the Run Queue and starting execution of microcode at an address determined by the status bits. When the new context is activated the processor address is mapped, and a request for a main-memory reference is placed in the *Out Queue* (during this time the Kbus has been free to read in service requests or perform functions requested by the Pmap). A *context swap* is executed in the Pmap to deactivate the current context pending the completion of the memory reference and to activate the next one in the Run Queue. The Kbus transfers address and data to the destination Slocal, then processes other requests while the memory reference is being performed. When the memory reference is completed the Kbus either reads the acknowledgment and/or data back into the Kmap and places the context back in the Run Queue for reactivation, or it sends the acknowledgment back to the processor that originally made the service request (thereby completing the mapping operation) and marks the associated context as "free" for reallocation to a new service request. The fact that a context remains allocated to each nonlocal reference until that reference is completed (regardless of whether or not more Pmap processing is expected to be needed) means that if an error is detected the context can be reactivated and will have enough state information to handle the error in an intelligent fashion.

Communication between the Linc and Pmap is similar to that between the Kbus and Pmap; the Pmap queues a request for an intercluster message to be sent (separate queues are provided for each Intercluster Bus) and suspends the requesting context. When a return message is received for the context the Linc causes the Kbus to reactivate the context in the Run Queue. When an incoming intercluster message is received by one of the Linc's Intercluster Bus Ports, it is queued and a request is issued to the Kbus to allocate a free context to the request and activate it in the Run Queue.

THE KBUS AND THE MAP BUS

Because of the great variety of tasks it must perform and the necessity that it be able to respond to errors in an intelligent way, the Kbus was designed as a microprogrammed processor controlled by 256 40-bit words of read only memory. It has a microcycle time of 100 nanoseconds which is synchronized with the 150 nanosecond clock of the Pmap and Linc at 50 nanosecond intervals. Figure 9 shows the major elements of the bus controller.

The Map Bus contains 38 signals, of which 20 are bidirectional lines used to transmit addresses and data between the Slocals and Kbus of the cluster. The Kbus is master of all transactions on the bus; as such it specifies a source and destination for each cycle as well as status bits indicating the use of the data (address, data, etc.). The bus is synchronous, with the Kbus generating all of the strobes used to transmit data. Each Slocal is provided with private service and return request lines to the Kbus. The arbiter section of the Kbus scans these in a pseudo round robin priority scheme.

The Kbus maintains the queues and registers used for communication with the Pmap. The Run Queue contains eight eight-bit slots (and thus is guaranteed never to overflow), each containing a three-bit context name and five additional bits of activation status. The Out Queue contains four 39-bit entries. The Pmap loads this queue to request Kbus operations and must check its state before loading to insure that it never overflows. Each Out Queue slot contains an op code used to select one of thirty-two Kbus operations, and additional address, data, and context information relevant to the operation. Two registers are loaded by the Kbus on behalf of each Pmap context. They are readable only by the Pmap and writable only by the Kbus. The *Bus Data Register* contains the last data word read in from the Map Bus for the context and the *Bus Condition Register* gives control and status information for the transaction.

The Kbus is responsible for the allocation and deallocation of contexts, and maintains the status of each context

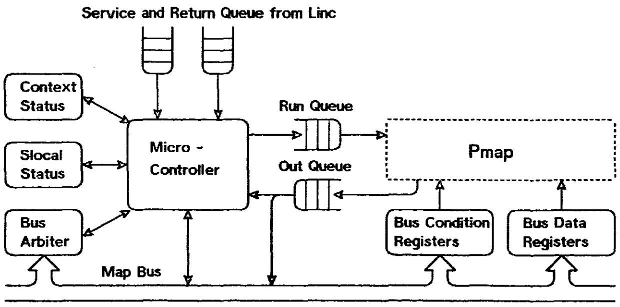


Figure 9—The components of the Kbus

for this purpose. It also keeps two additional bits of status for each context which are used to insure that, when a context suspends itself to await the execution of a main-memory reference or the sending of an intercluster message, an acknowledgment of the completion of the operation is received within a reasonable time (two milliseconds). If a suspended context times out it is forcibly reactivated with status bits indicating the error.

The Kbus also maintains nine bits of status for each Slocal in the cluster indicating whether the Slocal is busy with a Kmap-requested memory reference and, if so, what to do with the information returned at the end of the transaction. This status is set whenever a local memory reference is initiated and is used to insure that two contexts do not simultaneously try to request a memory access through the same Slocal.

THE PMAP, THE ADDRESS MAPPING PROCESSOR

The mapping processor of the Kmap, or Pmap, is a sixteen-bit horizontally microprogrammed processor. It occupies a central position within the Kmap, coordinating the activities of the other components. It is pipelined and has a cycle time of 150 nanoseconds. Microinstructions are 80 bits wide; a 1K*80 bipolar RAM is used as a writable microstore. The Pmap also uses a high-speed 5K*16 RAM to store the active Capabilities and segment descriptors. In addition to performing the basic address translation for the nonlocal references of a cluster, the Pmap must support certain operating system primitives, statistics gathering, and other experimental functions without excessive performance degradation.

Data paths

A register transfer level diagram of the Pmap is given in Figure 10. The main data paths consist of three internal high speed tri-state busses. Two of these, the *A* and *B busses*, take data from various sources and feed them to the inputs of the Arithmetic Logic Unit. The third bus, the *F Bus,* takes the ALU output and distributes it to various parts of the Kmap. The Kbus and Linc are also connected to these busses. Pipeline latches are used to overlap fetch of operands with current data operations.

The *Shift and Mask Unit* provides the ability to perform field-extraction on one of the ALU operands. This capability is important since the Pmap frequently deals with packed information in segment descriptors, intercluster messages, etc. The input to the Shift and Mask Unit is rotated by an arbitrary amount and then masked by one of 32 16-bit standard masks stored in a PROM.

For efficient address mapping, it is crucial that the Kmap have fast access to the information it needs to perform the virtual-to-physical address translation. This information consists largely of the active Capabilities and segment descriptors, of which up to 448 may exist in the cluster at a time (sixteen in each of two address spaces for each of fourteen processors). Although content addressable memory was not used because of the large capacity needed, the careful positioning of tables within the data memory, combined with a hash-coded list structure used for storing descriptors, has produced a cache-like structure.

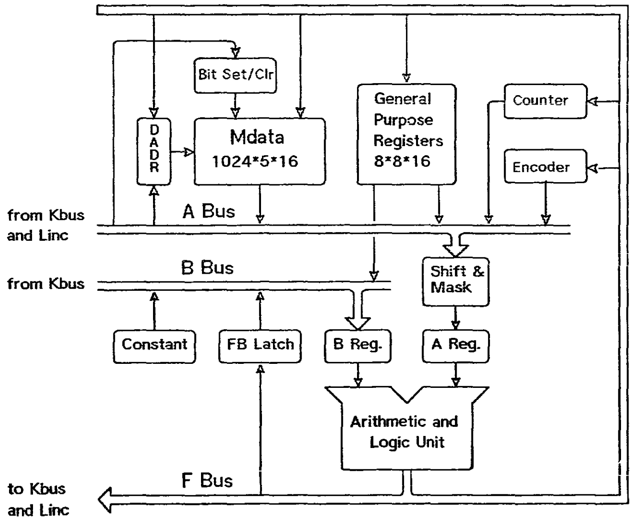


Figure 10—Data paths in the Pmap

The data memory, or Mdata, is divided into 1024 (expandable to 4096) *records,* each record containing five 16-bit words. The record organization was chosen because the segment descriptors, with cacheing information, fit comfortably within this 80-bit space. Each word has associated with it two parity bits, one for each byte. The memory is word addressable, with the record address coming from the *Data Address Register* (DADR) and three-bit word indices from fields in the current microinstruction. Thus once the record address of a descriptor or capability has been computed, the individual subwords may be accessed without expending further cycles to generate data memory addresses.

Data to be written in the Mdata may be taken either from the A Bus or F Bus. Because it is frequently necessary to set and clear status bits in segment descriptors (for example the "dirty" and "use" bits used for demand paging, and the lock bit used for mutual exclusion) bit set and clear logic is provided for data input from the A Bus. It provides for the setting or clearing of either or both of the two high-order bits of the input word. To further increase parallelism, it is possible to simultaneously read and write different words of the same record. It is therefore possible, say, to set the "use bit" in one word of a segment descriptor and at the same time extract the segment limit from another word of the same descriptor.

Microprogram sequencing logic

One characteristic of the Cm* address mapping algorithms is the large number of conditions to be tested. The service of a typical request will require testing of request status, operation type, and segment type and checking of the following conditions: protection violation, descriptor locked, segment localizable, etc. To perform address mapping within a reasonable number of cycles requires the Pmap to have a flexible multi-way branch capability.

A block diagram of the microprogram sequencing logic is given in Figure 11. A *Base Address* is selected from either the *Next Address* field in the current microinstruction or the output of the *Subroutine Linkage Registers.* Two bits in the microinstruction select the mode of branching (two-way, four-way, sixteen-way) and two three-bit fields control six 8-to-1 condition code multiplexers. Multi-way branching was implemented in the conventional way by OR'ing the selected condition codes with the Base Address. The address thus generated is stored in *MADR*, the *Microprogram Address Register,* to fetch the next microinstruction. There is a conditional override mechanism that can prohibit a potential 16-way branch. When the override condition is true, a branch is taken to a seventeenth location regardless of the value of the 16-way branch condition code.

Context considerations

There are a total of 64 general purpose and 32 subroutine linkage registers, allowing each context exclusive use of eight general purpose registers and four subroutine linkage registers. The *Current Context Number,* stored in the *Context Register,* selects the current register bank. Normally this register is loaded from the Run Queue when a context swap is executed. For diagnostic purposes the Pmap may directly load the Context Register, hence if required a microprogram may access the registers of any context. Each context may nest subroutine calls up to four levels deep. By convention, the zeroth linkage register is also used to store the reactivation address of a suspended context. The status bits in the Run Queue indicate whether a context is to be activated at its reactivation address (to continue an ongoing operation) or to be explicitly started at one of the first sixteen locations in the microstore (to begin a new operation, or handle certain error conditions).

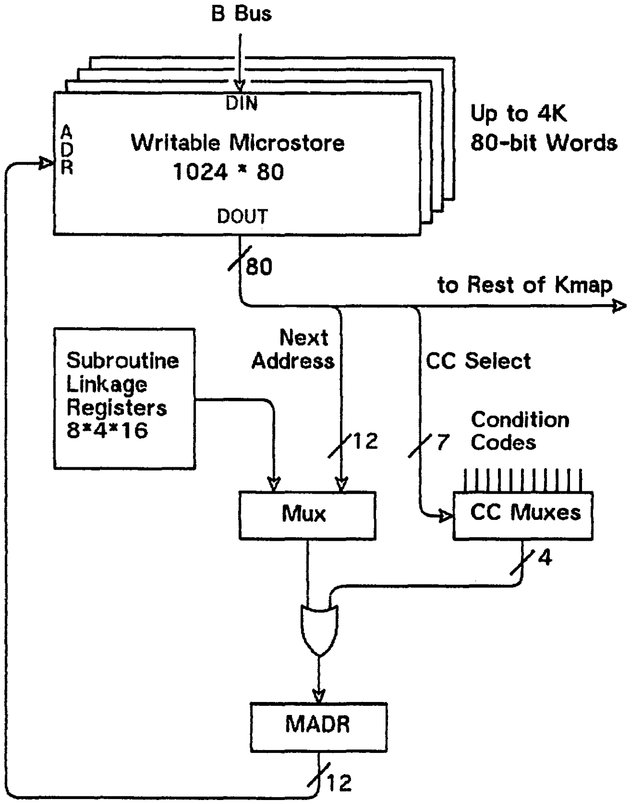


Figure 11—Microinstruction address generation logic

THE LINC AND INTERCLUSTER BUS STRUCTURE

The Linc provides intercluster communication by connecting the Pmap to two *Intercluster busses.* Communication is in the form of short messages passed between Kmaps. Messages are stored in a *Message RAM* which is shared between the Pmap and the two Intercluster Bus *Ports.* Pointers to messages pass through an automatic system of queues. Messages are usually sent directly from source to destination cluster, but they can also be forwarded by intermediate clusters (thus allowing arbitrary network topologies to be constructed). Message routing is controlled by Pmap microcode. The goal in the Linc design was to provide fast, deadlock-free intercluster communication with a minimum of Pmap overhead.

Intercluster bus protocol

The Intercluster busses contain 26 lines: 16 data, 2 parity, and 8 control. They operate in an asynchronous, interlocked fashion at a transfer rate of 450 nanoseconds per word. Mastership is passed cyclicly between requesting ports, effectively implementing a round robin priority scheme. The current bus master arbitrates future mastership in parallel with its current data transfers.

Intercluster messages consist of one to eight 16 bit words. The most common formats are shown in Figure 12. The header word contains a six bit identifier for source and destination cluster, the source context number and the complex bit. A return message has a unique source field of

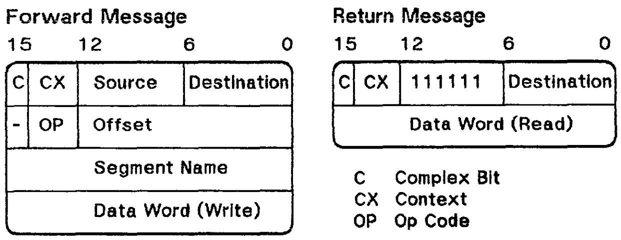


Figure 12—Standard message formats

all ones. The source context number is sent with the message to allow a direct reactivation of the suspended source context. The complex bit provides an escape mechanism to other message formats, e.g., for error messages or block transfers.

Components of the linc (Figure 13)

Buffer space for messages is provided in the central 1K*18 Message RAM, divided into 128 buffers of eight words each. This is sufficient to avoid any possibility of deadlock over buffer allocation except in very large systems.[3] The Pmap has priority for access to the Message RAM, although it is also directly accessible by the Ports. Several contexts may use the Linc in an overlapped fashion without interference since each context has private facilities for addressing message buffers. A context has two ways to address message buffers. It may use its context number to access a *reserved buffer* which is used for the creation of forward messages and to receive return messages. There is also a *Pmap Address Register* for each context to deal with incoming forward messages. Words within a buffer are selected by a Pmap microcode field. Each Port section has an address register and a word count register for accessing the Message RAM.

Five queues are maintained by the Linc. Two *Send Queues,* one for each Port, are used by the Pmap to request transmission of messages. To request that a message be sent on an Intercluster Bus, the Pmap places the address of the message buffer in the appropriate Send Queue. The *Free Queue* keeps the addresses of all the message buffers not currently in use. The *Service Queue* is used by the Linc to notify the Kbus and Linc of the addresses of incoming forward messages, and the *Return Queue* to request that the Kbus reactivate contexts when replies to their forward messages are received. All of the queues are implemented as partitions of a single 1K*11 bipolar RAM.

The Linc uses the same 150 nanosecond clock as the Pmap. For diagnostic purposes the Pmap has access to almost all of the internal state of the Linc and may execute all the internal microcycles executable by the Ports.

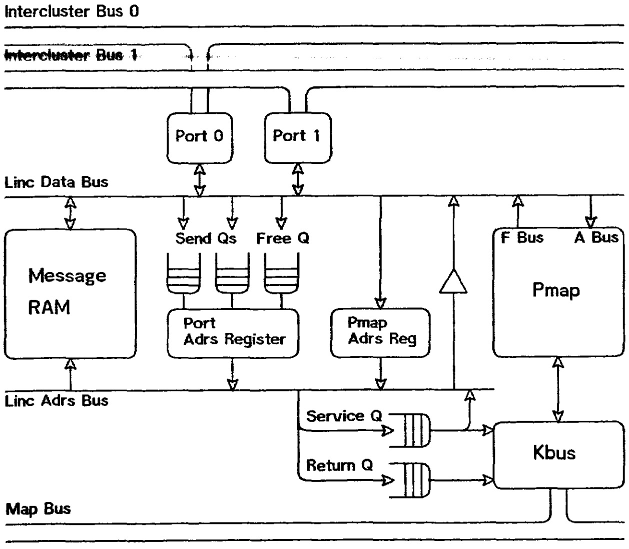


Figure 13—Components of the Linc

An intercluster message transaction

A complete message transfer is shown in Figure 14. The Pmap at the source cluster creates the forward message in a reserved context buffer. Then its pointer is put into the appropriate Send Queue. The Linc pops the pointer off the Send Queue into the *Port Address Register,* acquires mastership of the corresponding bus and transfers the message, one word at a time, from its Message RAM onto the Intercluster Bus and into the Message RAM of the destination Linc.

At the destination side the receiving Port has already obtained a buffer from the Free Queue. If the message is received completely without error, then its pointer is placed into the Service Queue (if not, the message is ignored; a timeout will occur at the source). The Service Queue requests the Kbus to allocate a free Pmap context to service the message. It includes status bits to start up specific microcode. The context will transfer the pointer from the Service Queue into the Pmap Address Register and process the message, making appropriate main-memory references. It then creates a return message in the same buffer, setting the source field to ones to indicate this. On a Read, the data word will be appended. The buffer pointer of the completed return message is queued again in the Send Queue. When the message has been sent, the pointer is released into the Free Queue. At the original source the return message is placed in the reserved buffer for the requesting context. Its context number plus status is passed to the Return Queue and the context is reactivated to send data or an acknowledgment back to the requesting processor.

DEVELOPMENT AND DIAGNOSTIC AIDS

A common strategy used to aid in hardware and/or microcode development is to construct a software simulator

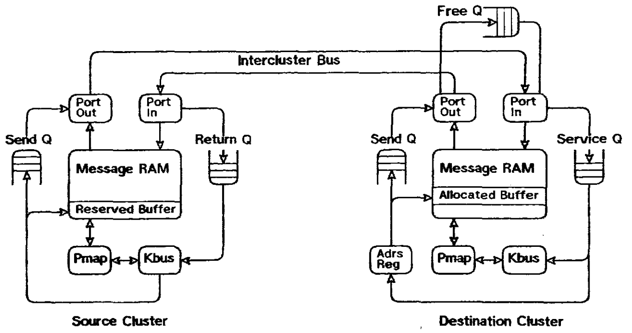


Figure 14—An intercluster message transaction

for the hardware. This allows initial debugging to be performed before the actual hardware is available and can provide a more comfortable environment in which to work. However, simulators are expensive both in terms of development effort and computer time; furthermore they cannot give an exact reflection of the hardware. Thus this approach leaves the final bugs to be found using the real hardware, and is of no aid in diagnosing component failures (rather than design errors). The alternative approach adopted for Cm* was to incorporate special hardware, called *Hooks*, directly into the Kmap for use in hardware and microcode development. The interfacing of the Hooks to a standard LSI-11 allows extensive software support for hardware development and diagnostics while at the same time providing a convenient environment for the debugging of microcode on the real hardware.

The Hooks give to an LSI-11, referred to as the *Hooks Processor,* the ability to intimately examine and change the internal state of the Kmap. They provide the capability for the Hooks Processor to load microcode into the writable control store of the Pmap, read the values on the A and B busses of the Pmap, and to independently start, stop, and single-cycle the Pmap-Linc and Kbus clocks. An interrupt is generated for the Hooks Processor whenever the Pmap clock stops (either due to a microprogram-invoked halt or a memory parity error on the control or data stores). Furthermore, several of the internal registers of the Pmap have "twin registers" associated with them which may only be loaded by the Hooks Processor. These alternate registers may be enabled via the Hooks to override microprogram-controlled values. The presence of the Hooks added approximately ten percent to the cost of the Pmap while enormously reducing system development time.

PERFORMANCE: MEASUREMENTS AND PREDICTIONS

Before discussing the models used to estimate the performance of a Cm* cluster, several simple measurements (made on a cluster containing two processors) will be presented. The average time between memory references (including both code and data) made by a single LSI-11 executing entirely out of local memory varies between 2.5 and 4.0 microseconds, depending on the mix of instructions being executed. For a "typical" code sequence, based on measurements of compiled BLISS-11 programs, the interreference time was 3.0 microseconds. Measurements made on the same "typical" code sequence, except with all references mapped via the Kmap to the other processor in the cluster, yielded an average time between references of 7.7 microseconds. With the latter measurement there was no contention for use of the Map Bus, Kmap, or destination Slocal. Although no actual measurements were available at the time of this writing, it is expected that the time for intercluster references will be between 15 and 20 microseconds.

A simple queueing model was developed to estimate the performance of a cluster.[4] The model assumed an exponential distribution of nonlocal requests, exponential service time in the Pmap, and exponential distribution of the total non-Pmap overhead incurred during a nonlocal reference. It is assumed that the Pmap is the primary cause of contention hence the waiting time for other facilities is ignored. Figure 15 plots the results of this analysis. The relative rate of memory referencing in a cluster is plotted as a function of the number of active processors and their *hit ratio* to local memory.

Because of the inability of the queueing analysis to model contention for all cluster facilities it was feared that the results would prove to be an optimistic estimate of cluster performance. Therefore a series of simulations was performed in order to model more closely the true operation of a cluster.[5] The simulation and queueing results were in close agreement and so the simulation study will not be discussed further.

Figure 15 indicates that system performance is extremely dependent on the local hit ratio. It has been hypothesized that the local hit ratio would lie in the range between 85 and 95 percent, in which case the effect of the nonlocal references would be "reasonably" small. Unfortunately, this implies that code must be entirely local to the processor executing it. Two memory-intensive programs, a quicksort

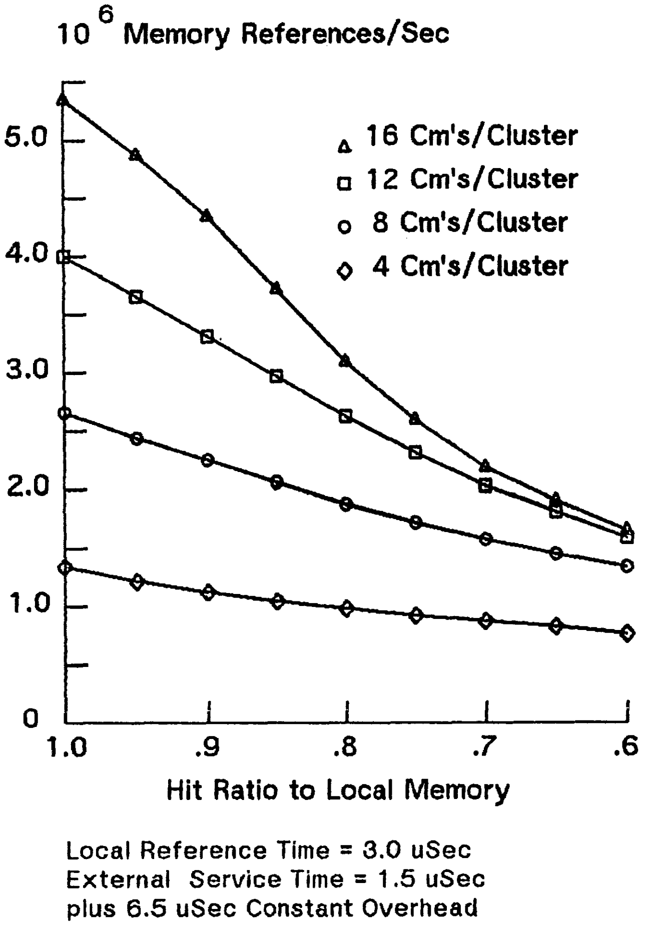


Figure 15—Absolute cluster performance

and a memory diagnostic, have been run on the initial Cm* system (one cluster, two modules). Measurements of the performance degradation when code and local variables are kept local but the area being sorted or diagnosed is moved to the other processor in the cluster indicate that local hit ratios of 90 percent or higher are being obtained in both cases. Expensive operating system functions such as block transfers are expected to lower this figure, but it is also expected that most user programs will make less intensive use of shared databases than the above examples.

The queueing model was used to predict the degradation of cluster performance if either the Pmap were made slower (and thus cheaper) or if the concurrency of the mapping mechanism were eliminated. The results for a cluster containing twelve processors are shown in Figure 16. A slower Pmap was modelled by increasing its service time from 1.5 to 3.0 microseconds. The last model represents a cluster implementation where each external reference is carried to completion before servicing subsequent requests. This would be the situation if only one Pmap Context were provided, i.e., eliminating the concurrency between the Map Bus and the Pmap. Both the slow and non-concurrent clusters show enormous performance losses, especially at the low end of the 85 to 95 percent hit ratio range. The inability of slower or non-concurrent Kmaps to support large numbers of modules implies a need for more Kmaps per Cm* system. It also suggests that more intercluster communication will be required since each module will have fewer immediate neighbors.

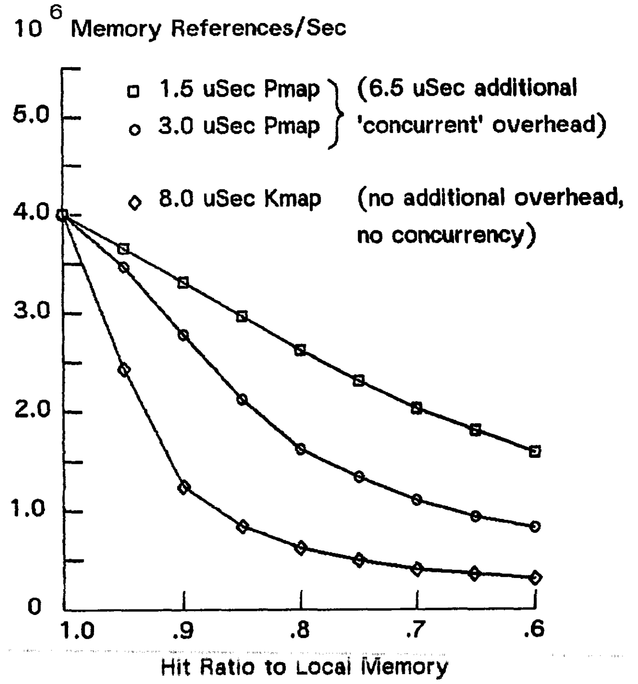


Figure 16—Cluster performance with slower Pmap or without concurrency between Pmap and Map Bus

CONCLUSION

Detailed hardware design of Cm* begain in late July, 1975. The initial goal of a 10 processor, three cluster system is expected to be realized in the first quarter of 1977. Considering the Kmap alone, the time from the beginning of design to a working prototype (excluding the Linc) was less than nine months. It is felt that this relatively short development time is due to extensive use of automated design aids, microprogramming at almost every level and the inclusion of additional hardware to aid in debugging. The Hooks facility in the Kmap has been particularly successful. However it will not be possible to declare the overall system a success until it is regularly and reliably supporting a community of satisfied users.

REFERENCES

1. Swan, R. J., S. H. Fuller, and D. P. Siewiorek, "Cm*: a Modular, Multi-Microprocessor", *AFIPS Conference Proceedings,* Vol. 46, 1977 National Computer Conference.
2. Bell, C. G. and A. Newell, *Computer Structures: Readings and Examples,* McGraw-Hill, New York, New York, 1971.
3. Swan, R. J., L. Raskin and A. Bechtolsheim, "Deadlock Issues in the Design of the Linc," Internal Memo, Computer Science Dept., Carnegie-Mellon University, March 1976.
4. Swan, R. J., S. H. Fuller, and D. P. Siewiorek, "The Structure and Architecture of Cm*: A Modular, Multi-Microprocessor," *The Computer Science Department Research Review 1975-1976,* Carnegie-Mellon University, December 1976.
5. Brown, K. Q., "Simulation of a Cm* Cluster," Internal Memo, Computer Science Dept., Carnegie-Mellon University, May 1976.

Software management of Cm*— A distributed multiprocessor†

by ANITA K. JONES, ROBERT J. CHANSLER, JR., IVOR DURHAM, PETER FEILER and KARSTEN SCHWANS

Carnegie-Mellon University
Pittsburgh, Pennsylvania

ABSTRACT

This paper describes the software system being developed for Cm*, a distributed multi-microprocessor. This software provides for flexible, yet controlled, sharing of code and data via a capability addressed virtual memory, creation and management of groups of processes known as task forces, and efficient interprocess communication. Both the software and hardware are currently under construction at Carnegie-Mellon University.

INTRODUCTION

Semiconductor technology advances are leading toward the inexpensive production of computer modules (i.e., a processor plus memory of a moderate size) on a single chip. Multiple computer modules interconnected to form a multiprocessor or a network offer a large number of processing cycles far more inexpensively than an equally fast uniprocessor. Yet, such a computer module system is useful only if a suitable fraction of the processing cycles can actually be used for applications.

The software designed to manage a computer module ~~system can contribute substantially to making the system~~ a cost effective environment in which to program applications. This paper discusses the software designed to manage a computer module system called Cm* which is currently under construction at Carnegie-Mellon University. We pay particular attention to the philosophy of software construction that influenced many of the design decisions.

For the purposes of this paper, we will only review some attributes of the architecture that are salient to the design of operating system software. Companion papers[1,2] describe and discuss the Cm* architecture in detail.

Cm* is a multiprocessor composed of computer modules, each consisting of a DEC LSI-11, a standard LSI-11 bus, memory and devices. We describe Cm* as a multiprocessor because the system's primary memory forms a single virtual address space; any processor can directly access memory anywhere in the system. To implement such a virtual memory, we introduced into each computer module a local switch, the Slocal‡ which routes locally generated references selectively to local memory or to the Map Bus (when the reference is to memory in another computer module). The Slocal likewise accepts references from distant sources to its local memory..

Connected to a single Map Bus may be up to fourteen computer modules that share a single address mapping and routing processor, called the Kmap. The computer modules, Kmap, and Map Bus together comprise a *cluster*. A Cm* configuration can be grown to arbitrary size by interconnecting clusters via Inter-cluster Busses (see Figure 1). (A cluster need not have a direct bus connection to every other cluster in a configuration.) Collectively, the Kmaps mediate each non-local reference made by a computer module, thus sustaining the appearance of a single virtual address space.

Because processors are numerous, applications of any size will tend not to be designed in the form of a single program executed by a sequential process. Instead we expect users to create *task forces*, i.e., groups of processes cooperating to achieve a goal. Because the number of ~~processes in a task force may vary with the available~~ resources and task parameters, and because processes tend to be small (due to the relatively slow processors or limitations on the amount of local memory), a user will often be unconcerned with individual processes, communicating only with the task force itself.

The Cm* architecture offers to a user the option of employing tightly or loosely coupled processes. Loosely coupled processes communicate rarely, usually in conventional ways via a message transmission mechanism. Tightly coupled processes communicate often, sometimes using the efficient unconstrained paths provided by shared memory. Cm* permits both types of communication since it provides a message transmission facility as well as direct addressing of shared memory. Effectively, a user is free to view Cm* as either a multiprocessor or a computer network.

† This work was supported by the Defense Advance Research Projects Agency under contract F44620-73-C-0074 which is monitored by the Air Force Office of Scientific Research.

‡ In several cases names of Cm* components are derived from the PMS notation described in Reference 3.

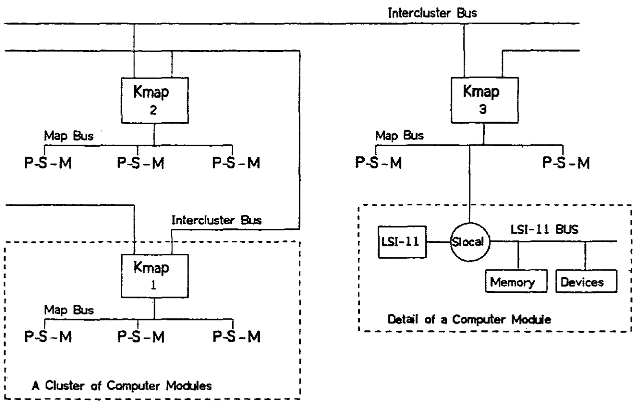


Figure 1—A three cluster Cm* configuration

SOFTWARE DESIGN METHODOLOGY

Cm* is a vehicle for experimentation, particularly in the area of parallel decomposition of algorithms and their efficient implementation on a computer module processing resource. We expect it to be rare that an experimenter (whom we will refer to as a user hereafter) is confident that all his code is debugged, since he will routinely alter parameters and even the code for his task forces in substantial ways. We also expect users to incrementally construct experiments. In addition we expect users to reconfigure modules (of software) combining them to form a new experiment.

Such a view of the user has led us to believe that it is as important for the kernel (or lowest level) software to support the user's software construction activities as it is to provide the primitive runtime facilities required for multiple users to share the computer resources in a disciplined cooperative fashion. Consequently, the software design reflects this concern. We view users as constructing their experiments by incrementally building *modules*.* Each module implements some abstraction useful to other modules that will come to depend upon it. A module then is a "unit of abstraction." It is implemented as

- code and data private to the module,
- a set of externally known functions that can be invoked by other modules making use of the abstractions, and
- a set of references to externally defined modules defining functions used in implementing the abstraction.

The kernel software supports the notion of a module by providing user facilities to create modules and to invoke functions of a module in a protected way. An invoked function is executed in an environment that gives it access to code and data that are part of the module, together with any actual parameters specified by the invoker. Thus the software enforces the boundaries of a module by providing a well defined transition between execution in one module and execution in another. Hopefully this will help contain the influence of errors and expedite debugging.

This notion of module is based on earlier work. In particular it is built on the ideas of modular decomposition discussed in Reference 4 and abstract data types[5] as used in language design.

Module boundaries are used for protection purposes at runtime. Each function is executed with access only to those objects which it requires. In designing the kernel software, we have found that some of its modules implement rather complex abstractions. Yet not all uses of a module require the entire abstraction; some uses rely only on part of the abstraction while others rely on a simplified abstraction. For design purposes a module may be partitioned into a strictly ordered set of *levels* as described in Reference 6. The purpose of dividing a module's design into levels is to permit either incremental introduction of the different parts of one abstraction or increasingly more complex (and powerful) versions of the entire abstraction. The introduction of complexity is postponed until it is truly required. Multiple levels of one module share data structures and even code.

The first level within a multi-level module may define only a subset of the functions of the complete abstraction, but that subset of functions is a useful self-contained, but limited version of the abstraction. Subsequent levels are introduced into the hierarchy as needed. Additional levels of a module may introduce entirely new data structures or extend existing ones. No protection boundaries exist between levels so that higher level code may manipulate data structures introduced in lower levels. Consequently, though module boundaries are translated into runtime protection boundaries, the boundaries between "levels of design" are not detectable in the runtime implementation structures. We will illustrate this difference between modules and levels later when we discuss the Cm* message transmission module.

Levels within a module are strictly ordered. We can define a level A to be "higher" than level B in another module in case A invokes a function defined in B. The set of all levels (of all modules) is partially ordered by dependency. In the design of operating system software, there is not necessarily a cleanly identifiable division of a hierarchy of levels into supervisory and user software. The operating system facilities required by one user differ from those required by another, particularly in an experimental setting. The partially ordered system structure is in a form such that it is readily possible to replace "upper" portions of the dependency hierarchy since level boundries are clear and the dependency relations between levels are known.

CM* SOFTWARE SYSTEM DESIGN

Before describing the kernel software design, we will define two notions that play an important part in that

* This paper always uses the words "computer module" to refer to the hardware structure, and will in the sequel use the (commonly accepted) single word "module" to refer to a programming abstraction. Context should also serve to eliminate any ambiguity.

design: objects and capability addressing of objects. The basic unit which can be named, shared and individually protected, and for which memory is allocated for representation purposes is the *object*. Each object has a unique name and a definitive description used by the software system. Every object has a type that determines the structure of its representation and the operations or accesses which can be performed on it. Current design specifies three types of objects: *data segments,* which are linear arrays of words that may be read and written; *capability lists*, which are structures containing capabilities (to be discussed below); and *mailboxes*, which are structures containing messages.

Objects are named (addressed) using *capabilities*.[7,8] A capability may only be created and manipulated in controlled ways (by kernel provided capability functions). Since users cannot create or forge capabilities, possession of a capability is evidence that the user can reference the object whose unique name appears within the capability. A capability not only identifies a unique object, it records a set of *rights* indicating which of the defined operation (accesses) are permitted to be performed on the object. Controlled use of objects is enforced because an object can be accessed only if a program presents a capability naming that object which contains a right for the desired access. Since possession of a capability endows the possessor with the ability to perform accesses, capabilities also record those rights which a possessor may exercise with respect to the capabilities themselves. (For example, copying a particular capability may not be permitted.)

Based on the above discussion, we next describe the Cm* kernel software. The purpose of the initial levels of software is to provide facilities required for shared usage of resources in an "enforceably cooperative" way. In addition we wish to assist users in programming and executing their experiments by providing convenient structures and functions for creating and executing modules. The operating system software itself is composed of a partially ordered set of levels. In several instances two modules are divided into a pair of levels. For convenient reference, levels are labeled with a tag in the format "module-level." Modules are given alphabetic names; levels are numbered in increasing order as they appear in the system construction hierarchy. The kernel levels to be discussed in this paper are:

CAP-1: Capability referencing
Performs mapping from a capability via a segment descriptor to physical representation of segment (including access control checking)

CAP-2: Capability addressing and memory allocation
Defines an object address space and interpretation of an address; performs memory allocation ensuring that the segments used to represent objects are pairwise exclusive

ME-1: Environments and Modules
Implements the creation and deletion of modules and execution environments

MSG-1: Conditional message transmission
Defines the structures message and mailbox; permits sending and receiving of messages when process suspension is not required

DSP: Dispatching
Defines hardware implemented data structures used to 'load' an environment onto the processor and commence execution

MPX: Multiplexing
Selects the next environment to execute on a processor

ME-2: Environment relations
Records the ancestry by which environments are related; provides for nested and parallel execution of environments

MSG-2: Unconditional message transmission
Provides for sending, receiving, and replying to messages even if environments involved are forced to wait for an indeterminate time to complete message transmission

TI: Trap and interrupt handling
Provides routing of control when either interrupts or traps occur

A diagram indicating the dependency relations among these levels appears as Figure 2. An arrow from level A to level B indicates that a function in level B is invoked in level A. In addition, it is possible that level A invokes functions in any of the levels 'below' B in the dependency graph.

Capability addressing

Module CAP provides capability addressing. Level CAP-1, which is implemented in Kmap microcode, interprets capability references to objects, i.e., it maps a capability to the physical representation of the object named by the capability. Because the state of an object may change and its physical representation may move, the system maintains a single definitive description of each object called a *descriptor* or *segment descriptor*. It records the type of the object, the physical description of its representation (including cluster, module, starting address, and size), state information (e.g., whether the representation is in core, dirty, or locked for Kmap usage), and the (reference) count of the number of outstanding capabilities for the object.

Every existing object has a unique name—the memory address of its descriptor. To perform a mapping from a capability to an object, the identity of the object's descriptor is determined from the capability. It, in turn, is referenced to determine the physical representation of the object. A capability reference fails if the right required to perform the operation desired by the addressing environment originating the reference is not in the capability.

Level CAP-2 extends level CAP-1 to provide for the generation of capability references (we refer to this as

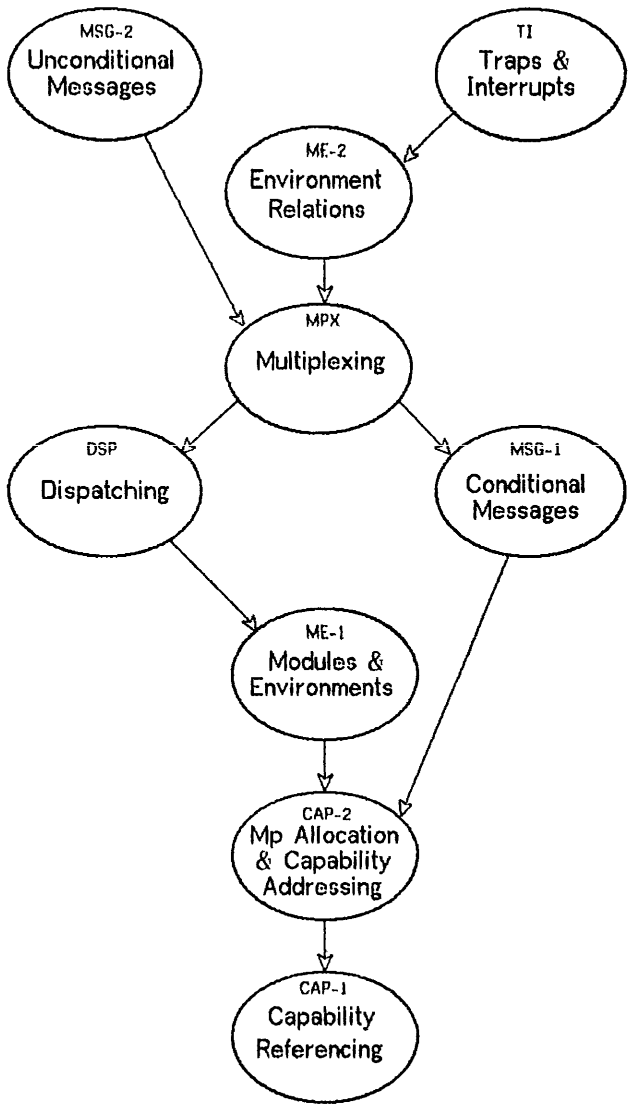


Figure 2—Levels and modules of Cm* software

capability addressing), and for capability manipulation. Capabilities used for addressing purposes are stored in capability array objects called *capability lists*. Given a capability list CL and an index X, one can determine the X-th capability in capability list CL. This may be a capability for an object of arbitrary type, including a capability list object. By repeated application of capability indexing, objects to any depth can be addressed. Because capability list indexing is performed in microcode as well as in software, the architecture restricts indexing to depth 2 in any single operation. This means that in a single addressing operation the path to a target object may "indirect through" at most two capability lists before arriving at the (third) target object. Whenever a processor is executing (i.e., generating capability addresses) one capability list is distinguished as the *primary capability list*. The first index of a capability address is an offset into this primary capability list.

CAP-2 also defines (microcoded) functions for creating, copying, moving, and deleting capabilities as well as for manipulating the rights encoded within a capability.

A Cm* processor (an LSI-11) has a word size of only 16 bits. To permit 16 bit addresses to be mapped to the arbitrarily sized Cm* memory, the notion of a *window* was introduced. It consists of 15 *window registers*, each of which can be thought of as holding a capability. (Actually, in the current design, each window register holds an index to a capability which can be indexed via the current primary capability list.) CAP-2 provides two (microcode implemented) functions *Segload* and *Unload* to associate and de-associate, a window register and a capability. To read or write a data segment, a capability for the segment must be *segloads* into a window register.

A 16 bit machine address is interpreted to select a window register (and thus a capability) and possibly to specify an offset into a segment of memory. For enhanced performance of capability referencing, the descriptors for the objects named in the capabilities associated with the window registers are cached in the Kmap. This mechanism provides virtual addressing and allows for conventional relocation of physical memory. It is sufficiently general to support the definition of Kmap microcoded operations on capability lists and mailboxes.

The last facility introduced in CAP-2 is that of memory allocation. Physical memory is allocated to hold segments so that no two segments overlap.

Modules and environments

Level ME-1 provides for the creation and deletion of modules (as discussed earlier) and for executing invoked functions. A module is implemented by a *module capability list* containing

- capabilities for the code and data segments required to perform the functions defined in this module,
- a data segment containing a vector of *function descriptors* which specify the code to be executed when a particular function is invoked (e.g., the index into the module capability list for the segment containing code for this function), the number of parameters expected and the size of stack required to perform the function,
- a list of other "known" modules containing functions that can be invoked by this module.

ME-1 also defines an *environment*, the structure created as a result of a function invocation. An environment is defined by several objects; one is the primary capability list which is private to a function invocation and acts as the root capability list for all addressing of objects during execution of the function.

The primary capability list contains capabilities for

- the execution stack (private to the environment)
- the module capability list which defines the module containing the invoked function,

- a *state vector* (private to the environment) which contains the processor and addressing state when the environment is not executing on a processor. (The state vector includes processor registers, processor status word, scheduling data, trap and error masks for communicating with the Kmap, and indices of the capabilities *Segload*ed into the window registers during the environments execution.)
- parameter objects specified by the invoker.

The module capability list contains capabilities for those objects shared by all who invoke a function in the module. The primary capability list contains capabilities which are local to a particular invocation of a function.

Level ME-1 provides functions for the creation, initialization and deletion of modules and environments. These, in turn, are used by level ME-2 in providing functions relating the execution of different environments. Functions *Call* and *Return* allow nested execution, i.e., the *Call*ing environment is suspended for the duration of the execution of the newly created *(Call*ed) environment which terminates when the *Called* environment *Returns*. The function *Fork* permits an environment to request that a function be invoked to execute in parallel with its invoker until the function *Join* is performed.

ME-2 initializes a newly created environment to record priority information for scheduling purposes and to record the existence of a newly created environment in the *lineage* (family tree) of its creator. It is this lineage which is used by still higher levels to keep track of a task force, the set of environments which are cooperating to achieve some goal.

Message transmission

The members of a task force need to be able to synchronize their actions and to communicate with one another. To this end module MSG defines an abstraction of a *mailbox* which can contain *messages*. A mailbox is capable of containing some fixed finite number of messages maintained in FIFO order. To permit users to communicate arbitrary objects to one another, rather than data only, messages are pairs of capabilities. (To transmit 16 bits of information, a user can create a *data capability* to contain this user specified information.)

Levels MSG-1 and MSG-2 differ in that MSG-1 provides only the functions *CondSend* and *CondReceive* to transmit messages when these functions can be completed without suspension of the invoker. *CondSend* succeeds in depositing a message into a mailbox only if the mailbox has room for it. *CondReceive* is a function which returns the oldest message in case the mailbox is not empty. Hence *CondReceive* can be used for polling. A received message is placed in the receiving environment's *message-pouch*, a designated pair of positions in the environment's primary capability list. *CondSend* and *CondReceive* will return an error code if the mailbox overflows (is full) or underflows (is empty), respectively.

The second level, MSG-2, extends the set of message transmission functions to provide a synchronization as well as a communication mechanism. MSG-2 relies on the hierarchy above the MPX level where the notion of blocked environments was introduced. MSG-2 provides the unconditional message functions: *Send, Receive,* and *Reply*. *Send* performs the same tasks as *CondSend*: except when the target mailbox is full, *Send* will cause the sending environment to be blocked awaiting an opportunity to deliver its message. Likewise, the *Receive* function causes the environment attempting to *Receive* a message from an empty mailbox to become blocked. *Send*ing a message to an empty mailbox on which an environment is waiting will cause that environment to *Receive* the message and become unblocked. Similarly, if *Receive* causes a full mailbox to no longer be full, it will awaken the oldest environment awaiting to deposit a message.

MSG-2 also defines a *Reply* function for mailboxes. This function differs from *Send* in that after executing the *Reply* function on a mailbox as permitted by a capability for that mailbox, the right to *Reply* to that mailbox is removed from the capability.

The two levels of the message transmission module provide an excellent example of a decomposition of a single module. MSG-1 defines both message and mailbox data structures, but provides functions which are of limited applicability; in some situations the functions fail returning an error code. Conditional functions are used to transmit messages in a well-defined fashion, but do not perform synchronization.

MSG-2 extends the definition of the mailbox data structure so that waiting environments can be recorded when necessary. It also provides new functions extending the usefulness of mailboxes, but not "covering up" or subsuming the conditional functions which are useful when polling is desired. The multiplexing module relies on the conditional message functions of MSG-1 and implements blocking and unblocking on which the second level of MSG depends.

Dispatching and multiplexing

Dispatching (DSP) and Multiplexing (MPX) are both levels and entire modules. DSP defines the hardware implemented state vector and its associated *Envload* function which loads an environment onto a computer module and begins execution. *Envload* is implemented in a combination of Kmap microcode and software. Software portions of *Envload* locate the process register values and the processor status word values in the state vector and load them into the physical processor registers. The software then stores the index of its capability for the environment in a special location which alerts the Kmap that an *Envload* is in progress. The Kmap portion of this function loads appropriate values found in the state vector into the window registers and various Slocal registers.

Functions in DSP are used exclusively by the multiplexing module (MPX) which is responsible for selecting the next environment to be *Envload*ed. Module MPX defines a set of *Runqueues,* each of which is a mailbox. If an environment is eligible for execution, i.e., it is not blocked nor already executing on some processor, then there is a message containing a capability for it in one of the runqueues.

Associated with each processor is an ordered list of at least some of the runqueues. The ordering selects the priority with which that processor services the mailboxes. The same Runqueue may appear in various positions in the ordered list of runqueues of different processors. The *Multiplex* function, invoked by the superior levels ME-2 and TI, cycles down the list of runqueues (private to the processor executing *Multiplex*) performing *CondReceive*s on the runqueues. If the *CondReceive* is successful, then the result is a capability for the next environment to be *Envload*ed on the executing processor.

Trap and interrupt handling

Software traps and interrupts signal exceptional conditions caused by program action and external asynchronous events, respectively. With only a few exceptions (e.g., responding to a clock interrupt or to a high speed device interrupt), hardware traps and interrupts are translated into software traps and interrupts, so that modules can indicate what action is to be taken when they occur.

Defining a new trap (interrupt) means defining a new *trap (interrupt) vector entry* indicating what function in what module is to be invoked if the trap (interrupt) occurs. When a trap occurs, it was caused by the executing environment, so a *Call* is performed to suspend the current environment and cause the function named in the appropriate trap vector entry to be executed.

Interrupts are asynchronous, and are not necessarily related to the current processor execution. TI offers two options. As a result of an interrupt a *Fork* can be performed to the function named in the associated interrupt vector. This will cause the interrupt to be serviced in parallel with execution of other environments. Alternatively, an interrupt vector or trap vector entry may direct that as a result of an interrupt, status information be sent as a message to a specified mailbox. Presumably some environment capable of handling the interrupt will *Receive* or *CondReceive* to get the message. Interrupts would then be processed sequentially by order of occurrence.

Two observations are appropriate here. One is that using the trap and interrupt mechanism, any level above TI can define vector entries so that code from higher levels can respond to exceptional conditions encountered when code from lower levels is executing. This effects "outward calls" so that lower levels can rely on higher levels when exceptional conditions arise. The second observation is that the trap and interrupt module is quite small, relying heavily on ME for *Fork* and *Call*, and on MSG for mailboxes.

The kernel system

The Cm* architecture provides alternative ways to implement functions. A function may be implemented in Kmap microcode, or it may be implemented in software to be executed by one or more of the computer modules. A computer module may execute a function in either of two address spaces (user or kernel space). The decision where to place a particular function of a particular level of a particular module is determined by considerations such as maximizing performance, providing for proper synchronization, and ease of implementation, as well as maintaining protection boundaries between modules. Because of this independence between the design and the physical realization, alternative implementations of a function are possible. This facility is expected to be valuable in a system designed for experimental use because it allows for function substitution and redesign.

The kernel software system described here is implemented in two parts: Kmap microcode and a set of programs which run in the kernel space of the computer module processors. It is intended that in the initial system all of the capability functions and message functions will be performed by Kmap microcode. The remaining functions will be implemented in software to be executed from the kernel space of the computer modules.

The kernel and user spaces have symmetric data structures because both are executing environments. Both the user and the kernel system have a primary capability list which acts as a "root" for capability addressing purposes. Both primary capability lists include a capability for a state vector and for a module capability list. It is the primary capability list and the state vector of the kernel space that maintain information particular to a processor. Shared data and code in the kernel are referenced via capabilities in the kernel's module capability list.

STATUS OF SOFTWARE DEVELOPMENT

As of December 1976, the microcode available provided only for simple relocation of physical addresses with no capability referencing. Development of microcode to support capability operations and the message facility will follow shortly.

Kernel space programs have been coded in BLISS-11,[9] a system implementation language. This set of programs is being tested using a simulator for the Cm* machine[10] which executes on C.mmp, another multiprocessor system developed at Carnegie-Mellon University.[11] The simulator models multiple computer modules as multiple processes, and is able to run at about half the speed of a Cm* processor by exploiting the writable control store features of the C.mmp multiprocessor. Since the kernel code is successfully executing on the simulator, it is expected that the software kernel will be available for use shortly after the completion of the Kmap microcoding.

Future software development

The kernel system modules as described constitute a very primitive system. A number of additional software levels and new modules are in various stages of design. It is expected that most of the levels in these modules will be implemented as programs in the user space. Modules under development include:

Secondary Store Management—Current design proposes adding some disk memory local to some clusters, with large file storage accessible via a high speed link to either the C.mmp or the DEC KL-10.

Linkediting—The creation and management of modules as Cm* modules will be performed by a linkeditor intended to simplify the construction and management of function tables, segments of code, and invocation sequences.

Command Interpreter—This module will provide on-line, interactive access to the Cm* machine. This will allow a programmer to dynamically manage a task force. Currently interactive terminal communication is provided by a PDP-11 connected to each computer module by a serial line unit.[12]

ALGOL 68 Runtime System—The first programming system to be available on the Cm* machine is expected to be ALGOL 68. (Until such a system is available, code will be cross-compiled on another machine). This version of ALGOL 68 will be designed to exploit the multiprocessing facilities of the Cm* machine.

Resource Policy Modules—A task force requires many runtime decisions concerning scheduling and resource allocation. It is the task of a policy module to provide for these decisions based up on the dynamic state of the task force and the Cm* machine as a whole.

SUMMARY

This paper represents a status report on the design of the firmware and software for management of a distributed multiprocessor called Cm* and the software construction philosophy which influenced its design. We have described the lowest levels of the kernel; some of the microcode and all of the software implementing what we have described now exists.

Besides continuing with the design and implementation of further levels of software, we intend to experiment with the placement and execution of kernel code within different Cm* configurations. Parameters of these experiments will include varying the physical location of the kernel code, the number of copies of that code as well as which computer modules can execute different portions of the code.

For example, one experiment is to limit the number of processors that can execute ME-2 code to (say) two processors in a cluster. If user programs executing on processors other than the designated two request ME-2 functions, their requests will be recorded so that the designated two processors can process these requests at some later time. The motivation for such an arrangement is that a processor is much more efficient if it executes code from its local memory.

In addition to such operating system experiments, we plan a number of experiments employing Cm* in the solution of different types of applications problems.

ACKNOWLEDGMENTS

The Cm* software design was strongly influenced by the ideas which emerged from the family of operating system project (reported in Reference 6) and the virtual memory project which predated the family of operating system effort (reported in Reference 4).

In addition we would like to thank three individuals, Richard Swan, Victor Lesser, and Lee Cooprider for their comments and suggestions.

REFERENCES

1. Swan, R. J., S. H. Fuller, and D. P. Siewiorek, "Cm*: a Modular, Multi-Microprocessor," AFIPS Conference Proceedings, Vol. 46, 1977 National Computer Conference.
2. Swan, R. J., A. Bechtolsheim, K. Lai, and J. Ousterhout, "The Implementation of the Cm* Multi-Microprocessor," AFIPS Conference Proceedings, Vol. 46, 1977 National Computer Conference.
3. Bell, C. Gordon and Allen Newell, *Computer Structures: Readings and Examples,* McGraw-Hill, 1971.
4. Parnas, D. L. and W. R. Price, "The Design of the Virtual Memory Aspects of a Virtual Machine," Proceedings ACM SIGARCH-SIGOPS Workshop on Virtual Computer Systems, 1973.
5. Liskov, B. and Steven Zilles, "Programming with Abstract Data types," *SIGPLAN Notices,* Vol. 9, No. 4, April 1974.
6. Habermann, A. N., Lawrence Flon, and Lee Cooprider, "Modularization and Hierarchy in a Family of Operating Systems," *Communications of the ACM*, Vol. 19, No. 4, April 1976.
7. Dennis, J. B. and E. C. Van Horn, "Programming Semantics for Multiprogrammed Computations," *Communications of the ACM,* Vol. 11, No. 3, March 1968.
8. Lampson, B. W., "Dynamic Protection Structures," *Proc. AFIPS 1969 FJCC 35,* AFIPS Press, Montvale, N.J., 1969.
9. Wulf, W., et al., "Bliss: A Language for Systems Programming," *Communications of the ACM,* Vol. 14, No. 12, December 1971.
10. Chansler, R. J., "Cm* Simulator Users' Manual," Department of Computer Science, Carnegie-Mellon University, 1976.
11. Wulf, W., et al., "HYDRA: the Kernel of a Multiprocessor Operating System," *Communications of the ACM,* Vol. 17, No. 6, June 1974.
12. Van Zoeren, H., "Cm* Host User's Manual," Department of Computer Science, Carnegie-Mellon University, December 1975.
13. Parnas, D. L., "On the Criteria to be used in Decomposing Systems into Modules," *Communications of the ACM*, Vol. 15, No. 12, December 1972.

Using assertions to improve language translators

by ARTHUR PYSTER
University of California
Santa Barbara, California

ABSTRACT

New enhancement techniques for language translators based on work in program verification are developed. Assertions are normally added to a program in order to verify the program is correct. Once verified, the assertions are usually ignored. This paper shows that verified assertions often contain information which can improve certain object code characteristics when the program is translated: execution time, storage requirements, and program style. The latter quality is especially important if the object code is itself in a "high-level" language. The techniques developed fall into three categories: early binding, using complementary constructs, and noting restricted cases.

INTRODUCTION

Enhancing translator output is an area of enormous practical concern. Optimizing object code is perhaps the most obvious enhancement a translator can make. Nearly all translators available try to be somewhat clever in reducing the run time and storage requirements of the code they generate. Even with increasing processor speeds and cheapening memory costs, time and space optimization will continue to play an important role in efficient computer utilization. Making full use of the more abstract features the target language offers is another form of translator enhancement. This is particularly true when the target language is itself high-level, therefore having many complex features to apply. For example, in translating into FORTRAN it would improve the style of the object code if DO-loops were generated rather than more primitive "IF,INCREMENT, GOTO" loops. Of course, the ability to translate between high-level languages increases portability when moving from one computer system to another where the languages or language dialects available differ. This paper develops new enhancement techniques based on work in program verification.

The increasing concern over program verification has created new opportunities for translator enhancement, especially from the specification and verification of program assertions.[1–3] Assertions typically state properties of programs such as the range of variables or the relationship between the values of two or more variables. Once verified, these assertions may be treated as an integral part of the program body and hence information drawn from them may be used to enhance translator output. Of course, it is absolutely imperative to the correctness of the enhancement that the assertions made are, in fact, true. Otherwise the enhanced code could exhibit different input/output behavior than the source program! With automated verification systems, there is little likelihood of error except in specifying the input assertions. But an error here quite possibly indicates the programmer misunderstands the problem specifications and hence the program would likely fail independent of the optimization.

The paper itself has six sections. The second section discusses assertions in general. The third, fourth and fifth sections detail three enhancement strategies assertions make possible: (1) early binding, (2) complementary constructs, and (3) restrictive cases. Finally, the last section summarizes the paper's contents and indicates future lines of work.

ASSERTIONS

The notion of program assertions is credited largely to Floyd[4] in a classic paper on proving programs correct (although he called them "verification conditions"). Assertions are conditions on the commands of a program such that each time a command with an assertion is reached, the condition should be true at that point. This is shown in Figure 1. The particular formats used to state assertions in this paper are self-explanatory and do not require detailed introduction.

Verified assertions can add a new level of abstraction to a program: Assertions (a) and (b) might be

(a) x IS A *stack*;
(b) y IS A *tree*;

found in a PL/I list processing program. Since *stack* and *tree* are not primitive PL/I data-types, the PL/I code itself would not contain this information directly about lists x and y. x and y would be described in a far more primitive

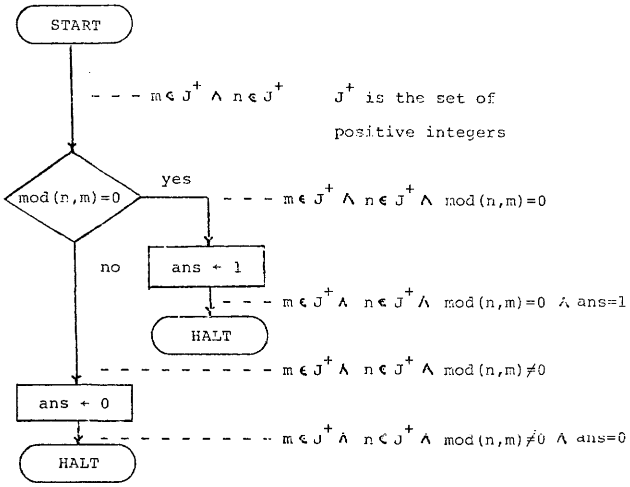

START
$m \in J^+ \wedge n \in J^+$
J^+ is the set of positive integers
mod(n,m)=0
yes
$m \in J^+ \wedge n \in J^+ \wedge \mathrm{mod}(n,m)=0$
no
ans ← 1
$m \in J^+ \wedge n \in J^+ \wedge \mathrm{mod}(n,m)=0 \wedge ans=1$
HALT
$m \in J^+ \wedge n \in J^+ \wedge \mathrm{mod}(n,m)\neq 0$
ans ← 0
$m \in J^+ \wedge n \in J^+ \wedge \mathrm{mod}(n,m)\neq 0 \wedge ans=0$
HALT

Figure 1—Flowchart (with assertions) of program which computes whether integer n is divisible by integer m

manner within the code. A translator might use such "cheaply" obtained information in the assertions to optimize storage allocation or structure processing. Examples of this are given in the fifth section of this paper. The cost of *determining* that *x* and *y* are stacks as part of an optimization step would probably be prohibitive. Moreover, without the "machine-understandable" statement in the text that *x* and *y* are special types of lists, the compiler would not even know which type of list structure *x* and *y* might be!

In a sense, assertion statements are language extensions. Currently there are no standards established for their syntax in the way ANSI has established standards for COBOL and FORTRAN. Within the next few years, a movement for standardization will probably emerge. At the point when assertion formats stabilize, translator writers will have a firm basis for treating assertion statements as an integral part of the host language and will then be able to consider assertion properties in their enhancement strategies.

EARLY BINDING

Binding in programming languages is establishing the mappings between names and data objects, and their descriptions. For example, the PL/I declaration

DECLARE X FIXED DECIMAL (3,1) AUTOMATIC;

binds the name X to a data object which holds three-digit decimal numbers whose values range from −99.9 to 99.9, inclusive. Furthermore, the memory location for storing the value of X is by virtue of the term "AUTOMATIC" in the declaration allocated when the block containing this declaration is entered.

Different languages have radically different policies on *binding time*, the time in the history of program execution when bindings are established. In general, FORTRAN binds as early as possible, while APL binds as late as possible. The common rule of thumb is that early binding is cheaper to implement but less flexible than later binding. If a translator tries to convert code in a language with a late binding time policy to code in a language with early binding time policy, serious difficulties can arise. For example, ALGOL permits a program to determine array dimensions dynamically upon block entry. FORTRAN binds array dimensions at compile time. Therefore, an ALGOL program which relies on dynamic dimensioning cannot be converted in a straightforward manner to FORTRAN. However, if the translator knew the *maximum* value the ALGOL program will, in fact, use to set array bounds, it could allocate a static array in the FORTRAN program with this maximum value for its bounds. Of course storage would sometimes be wasted by the FORTRAN program, but avoiding such waste is one of the main reasons ALGOL uses dynamic memory management. Unfortunately, there is no vehicle in ALGOL for specifying the range of a variable. Hence, no translator can, in general, perform this straightforward translation by relying on the ALGOL text alone. However, it is quite common for an assertion about the range of a variable to appear in a program:

```
ASSERT 1≤N≤100;

INTEGER ARRAY B(N);
```

This assertion, which is not part of the ALGOL language, makes it possible to intelligently bind the array size earlier in the FORTRAN program than in the ALGOL original. Without the assertion, there is no clear strategy for handling the binding time difference problem.

The problem with different binding times crops up again in translating from a *typeless* to a *typed* language, such as APL to PL/I. An APL variable can freely hold a character string at one instant and an integer or real number the next. PL/I requires a programmer to declare the data-type of each variable at compilation time and to maintain that data-type for the entire program run. Therefore, knowing nothing about the range of values an APL variable will assume, a translator cannot easily substitute appearances of APL variables with PL/I counterparts. However, through the use of assertions about the source program, it is conceivable that the range of many APL variables could be determined. In many cases these variables may have values of only one data-type assigned to them. For such variables the translation would proceed quickly. Having the proper assertions present in this case would then greatly simplify the object code.

Assertions can also be used to optimize *compiled* code with respect to binding time. Consider again an ALGOL array with dynamically computed storage bounds:

```
INTEGER ARRAY B(e);
```

where 'e' is a positive integer-valued expression. Every time the containing block is entered, compiled ALGOL code would probably recompute the value of e. There are many circumstances under which the value of e would be constant over a long period once it had been computed at run time. In these cases, it would be more efficient to save the value of e and re-use it, rather than repeatedly recompute it. An assertion to the effect that e remains constant over a specified time period would make this possible. An alternative way to avoid recomputing e would be to keep track of whether or not the values of the component variables of e change between block entries. This is an expensive bookkeeping operation. Furthermore, it will not handle situations in which the values of expression components change, but the overall expression value does not, such as in

```
INTEGER B(I−J);
```

where the difference between I and J could be constant even if the values of I and J individually change.

COMPLEMENTARY CONSTRUCTS

This section is largely founded on the premise that it is better to use "high-level" constructs in the object code whenever possible. This practice not only enhances program readability, which could be important if the object code is itself in a high-level language, it can also lead to improved time and space bounds for the high-level translator output when it is iteself compiled into machine code. The second advantage arises from the fact that special optimization techniques can often be developed to deal with complex but well-structured constructs. For example, FORTRAN DO-loops are often set up using a machine-language looping statement such as "Branch and Count" on IBM 360 hardware. It is more difficult to detect that this same fast construct is applicable if the more primitive "IF,increment,GOTO" form of loop is used instead.

Sometimes there are features in both source and target languages which seem analogous in purpose and often form. One such pair is the ALGOL FOR-loop and the familiar FORTRAN DO-loop. In translating from ALGOL to FORTRAN, however, it is not always possible to substitute a DO-loop for each FOR-loop occurrence in the source program. There are several important differences between DO- and FOR-loops even though they both have basically the same function. FORTRAN DO-loop parameters are restricted to be all integer variables or constants which have positive values. ALGOL FOR-loop parameters can be any arithmetic expressions. Furthermore, DO-loops are executed once even if the loop predicate is initially false. FOR-loops are skipped completely unless the loop predicate is initially true.

As a consequence of the differences between DO-loops and FOR-loops, it is not possible in general to substitute a DO-loop for a FOR-loop in the object code. Substitution is possible only when the translator knows that the FOR-loop parameters all have positive integer values, and that the loop predicate is always initially true. Without these guarantees, the object code must be gerry-rigged to accommodate the behavioral differences. This latter act slows the program, increases program size and hinders readability. With proper assertions inserted into the ALGOL source program, for those cases where the FOR-loop does behave in a manner equivalent to a DO-loop, the translator can generate the simple object code. In some cases it should be possible to translate the assertions automatically as well. Figure 2 illustrates the differences between translation with assertions and translation without.

A second example of complementary constructs is a built-in square-root function in the source and target languages. Suppose the source language has a complex number primitive data-type and the target language does not. It would be wrong to translate square-root function to square-root function unless the function argument is never negative. Computing the range of the argument may be prohibitively expensive or even impossible to compute using the source text alone. However, the proper assertions could make this determination feasible, if not trivial. The unappealing alternative is for the translator to construct its own square-root function whose range includes complex numbers. Since complex numbers are not a primitive data-type of the target language, they would have to be simulated using a 1×2 matrix or some similar vehicle. Such efforts would horribly muddy the object code without reason if, in fact, the function argument were never negative.

The two examples just cited are instances of a general phenomenon which is pictured in Figure 3. The *parameter* or *argument space* of a language construct is the domain of its input parameters or arguments. Figure 3a shows the union U of the parameter spaces of constructs A and B. The intersection of their parameter spaces, $A \cap B$, is the area with diagonal hatches. The circle $A \sqcap B$ within $A \cap B$ encompasses all common data points for which A and B behave identically. The larger $A \sqcap B$ is with respect to $A \cap B$, the greater likelihood that the straightforward translation is possible. Assertions are helpful in determining if a particular data point is in $A \sqcap B$ or $(A \cap B)-(A \sqcap B)$.

Figure 3b shows a situation related to, but distinct from that of Figure 3a. It is possible for constructs A and B to

```
INTEGER I,J,K,L;
   ⋮
FOR I=J STEP K UNTIL L DO
   BEGIN
     ⋮
   END;
```

(a)

```
      INTEGER I,J,K,L
         ⋮
      I=J
  5   IF I .GT. L GOTO 10
         ⋮
      I=I+K
      GOTO 5
 10   CONTINUE
```

(b)

```
INTEGER I,J,K,L;
   ⋮
ASSERT 1≤J≤L  K>0;
FOR I=J STEP K UNTIL L DO
   BEGIN
     ⋮
   END;
```

(c)

```
      INTEGER I,J,K,L
         ⋮
      DO 10 I=J,L,K
         ⋮
 10   CONTINUE
```

(d)

Figure 2—Program (a) without assertions translates to program (b). Program (c) with assertions translates to program (d)

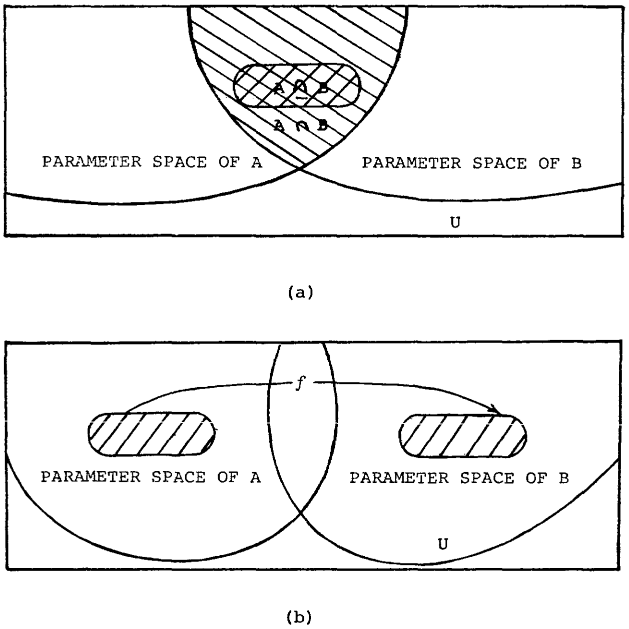


Figure 3—Relationships between parameter spaces of language constructs

behave identically for a significant number of *distinct* data points. For example, construct B may behave the same for *integer* input 3 as construct A does for *real* input 3.0, but B may be undefined on *real* input. In that case, a conversion *f* to the correct data-type in the object code would make it possible to emit the complementary construct B as the generated code. Assertions can identify whether such conversion is always possible; e.g., that construct A never has argument 3.5. Of course, the limiting factor here is the ease with which the conversion can be accomplished. Converting real to integer is fairly trivial, but converting sequential arrays to linked arrays can be costly.

RESTRICTED CASES

Often an operation or data item is restricted in some way which is difficult if not impossible to detect from the program text itself. A translator could often employ knowledge of such a restriction to advantage. Assertions can provide that information at low cost. For example, a program could assert that array X is *diagonal* or *sparse*. Storage could be conserved by allocating only half of the indicated space for the diagonal array and by using linked allocation rather than sequential for the sparse array. Having restricted cases of operations and data is common. Several other places where optimization is possible are:

(1) In ALGOL, SNOBOL or any language which treats all subroutines as potentially recursive, assert a subroutine is *not* recursive and forgo expensive run-time set-up.

(2) A high-level list processing language could have a "*search for node X*" operator. If it is asserted that the graph is acyclic and/or connected, the search is simplified.

(3) For the FORTRAN computed GOTO, skip the test in the compiled code for the index variable not being between 1 and the number of alternative branches if it is asserted it always will be.

(4) In a list-processing language skip the test for stack underflow if it is asserted the stack is never empty.

(5) If the asserted range of a data item is small, then storage allocation could be less than would otherwise be efficiently possible. This is especially important in translating between dialects of a language which are implemented on machines with different word sizes; e.g., going from CDC-6600 FORTRAN with 60-bit words to IBM 360 FORTRAN with 32-bit words.

(6) If it is asserted that string B is always a substring of string C, then when searching for the position in C where B begins, a recovery for a failing pattern-match can be eliminated.

CONCLUSIONS

This paper has demonstrated that assertions can be profitably applied outside their original context of program verification. In particular, several strategies for enhancing the object code generated by language translators, including but not restricted to compilers, have been developed.

Assertions can be viewed as language extensions. As such, they allow the translator to compensate for deficiencies of the host language whose programs are being translated. Thus, possible enhancement in style, speed and storage requirements of the object code are simply a beneficial side-effect of efforts in verifying program correctness. Consequently, the programmer never need concern himself with which assertions would be most profitable. There are many enhancement strategies made possible through effective use of assertions. Only a handful have been mentioned here.

There is a strong similarity between assertion usage here and the notion of a language preprocessor. The key difference is that a preprocessor is written to avoid modifying a compiler, while this work urges compiler changes for the sake of efficiency and style. Typical preprocessor extensions add new *commands* to a language. Assertions add new *descriptors*. There is usually no vehicle for expressing these descriptions in the host language; e.g., variable range in FORTRAN. Hence normal preprocessing techniques are not applicable.

Some of the inflexibilities of standard languages can be circumvented by the language augmentations assertions offer. It should be interesting to see how such features which enter a language through the back-door are applied.

REFERENCES

1. Katz, S. and Z. Manna, "Towards Automatic Debugging of Programs," *Intern. Conf. on Reliable Software*, Los Angeles, 21–23, April 1975.
2. King, J. C., "A Program Verifier," Ph.D. Thesis, Carnegie-Mellon University, Pittsburgh, 1969.
3. Deutsch, L. R., "An Interactive Program Verifier," Ph.D. Thesis, University of California, Berkeley, 1973.
4. Floyd, R. W., "Assigning Meaning to Programs," *Proc. of a Symp. in Appl. Math.*, Vol. 19, J. T. Schwartz (Ed.), A.M.S., 1967, pp. 19–32.

A parser analyzer of empirical design for question-answering*

by ABRAHAM S. BEN DAVID
Xerox Corporation
Rochester, New York

ABSTRACT

Over the last three years, work of an *empirical* nature has been carried out on the design of a natural-language question-answering system. In the present paper, the parser-analyzer of version 3.0 of the QUANSY (*QU*estion *AN*swer *SY*stem) system is discussed. The parser-analyzer is composed of a set of algorithms, each of which is assigned a particular set of tasks. The parser-analyzer operates in a multihypothesis structure. The whole system is programmed in FORTRAN and the sample run times are for a run on a CDC 6400.

INTRODUCTION

Research into natural language systems of all types has been going on for years.[1-3] Even with all this effort, little headway has been made into the design and implementation of an efficient natural-language, question-answering system. The systems that have been implemented[3-6] consume large amounts of computer time and memory and are of limited application. In most cases, previous attempts have utilized various linguistic theories in the design of their parsers. The question that should be posed is, "Is a theory of language explaining to humans how language works appropriate for machine analysis of language?"

The present research represents an empirical study into the question, "Can a model of language analysis specifically for machine analysis be developed?" The model developed for version 3.0 of QUANSY (*QU*estion *AN*swering *SY*stem) has been entirely of empirical design. Any idea, vis-à-vis language analysis, was actually tested by at least partial implementation, incorporated into the model if successful, modified—if possible—until successful, or discarded. Due to this empirical approach, work has progressed slowly, though "wrong paths" have usually been avoided. The results of the present version are detailed enough to demonstrate the potential of the model, though substantial additional work is necessary. After giving a sample QUANSY dialog, the paper will deal with the actual nature of the parser-analyzer of the system, and then with some concluding remarks concerning the import of this work.

SAMPLE SYSTEM OPERATION

The following sample operation is extracted from a larger interaction.[7] All of the QUANSY requests and responses are indented and all capitalized. User replies are nonindented. Two performance measurements will be supplied, PT-parse times in CP seconds (these are times for the process of the parser-analyzer, as reported on in this paper)+TO total time in CP seconds (including physical input of material, dictionary search, memory modification and—for questions—formulation of responses and actual generation).

```
  QUANSY HERE
  EITHER ANSWER MY QUESTIONS
  SEQUENTIALLY, OR IMMEDIATELY SUPPLY A
  MODE VALUE (IF YOU KNOW THE VALUES).
  DO YOU WANT TO ASK A QUESTION.
Yes
  DO YOU WANT TO SUPPLY A TEXT.
Yes
  I AM READY FOR YOUR TEXT,
  END LAST SENTENCE WITH XX. ---
  (I.E. SPACE BETWEEN PERIOD AND HYPHENS)
```

A mosaic is a picture that is made of colored stones. The first mosaics were probably made from slabs of alabaster which is a stone that is soft enough to cut. Parts of pictures were carved on the slabs and painted. These tiles or slabs were then put together to form a large picture. These large pictures were used to decorate the palaces and temple walls of kings in the Near East. The artists used the pictures to tell stories

```
  I AM ANALYZING YOUR LAST 5 SENTENCES.
  I'LL BE RIGHT BACK FOR MORE.
about the kings and what they did. ---)
```

* This work has been supported in part by NSF Grants NSF-GN-40900 and NSF Grant No. SIS75-09282.

```
  (PT—Sentence        1—.014
        "             2—0.34
        "             3—.013
        "             4—.021
        "             5—.028
        "             6—.025
  Average TO         .146       )
  I AM READY FOR YOUR QUESTION
  END IT WITH A PERIOD
Were mosaics used to decorate temples.
  I HAVE CONDUCTED THE BASIC SEARCH
  WITHOUT SUCCESS.
  DO YOU WANT ME TO CONTINUE THE SEARCH
  (YES OR NO).
Yes
  LARGE PICTURES WERE USED TO DECORATE
  THE KINGS TEMPLE WALLS.**
  (PT—question—.014, TO—.184)
  DO YOU WANT TO ASK ANOTHER QUESTION.
Did the mosaics tell stories about kings?
  —I AM ANALYZING YOUR RESPONSE FOR
  CONTENT.—
  I HAVE CONDUCTED. etc.
Yes
  THE PICTURES TELLED† STORIES ON THE
  KINGS.
  (PT—.017, TO—.151)
  DO YOU WANT TO ASK ANOTHER QUESTION
Were the earliest mosaics carved from alabaster.
  —I AM ANALYZING YOUR RESPONSE FOR
  CONTENT.—
  I HAVE CONDUCTED. etc.
Yes
  PICTURE PARTS WERE CARVED ON ALABASTER
  SLABS.
  (PT—.013, TO—.222)
  DO YOU WANT TO ASK ANOTHER QUESTION.
Yes
  I AM READY FOR YOUR QUESTION.
  END IT WITH A PERIOD.
I want to know about mosaics.
  A MOSAIC IS A PICTURE
  THE ONE‡ MOSAICS WAS MADE PROBABLY
  FROM ALABASTER SLABS.
(PT—.007, TO—.146)
  DO YOU WANT TO ASK ANOTHER QUESTION.
What were the earliest mosaics made from?
  —I AM ANALYZING YOUR RESPONSE FOR
  CONTENT.—
  THE ONE MOSAICS WAS MADE PROBABLY FROM
  ALABASTER SLABS.
(PT—.011, TO—.147)
  DO YOU WANT TO ASK ANOTHER QUESTION.
Is a mosaic made of pieces of rock.
  I HAVE CONDUCTED—etc.
Yes
  I REGRET, I CANNOT HELP YOU WITH THIS
  QUESTION.
(PT—.021, TO—.310)
  DO YOU WANT TO ASK ANOTHER QUESTION.
I want to add information.
  —I AM ANALYZING YOUR RESPONSE FOR
  CONTENT.—
  WHAT IS YOUR ADDITIONAL INFORMATION
(END IT WITH XX. ---)
Stones are rocks. ---
  I AM READY FOR YOUR QUESTION
  END IT WITH A PERIOD.
Is a mosaic made of pieces of rock.
  A PICTURE IS MADE ON COLORED STONES.
(PT—.015, TO—.159)
  DO YOU WANT TO ASK ANOTHER QUESTION.
I want to continue.
  —I AM ANALYZING YOUR RESPONSE FOR
  CONTENT.—
  I AM READY FOR YOUR QUESTION.
  END IT WITH A PERIOD.
I would like to stop.
  IT HAS BEEN A PLEASURE WORKING WITH YOU.
```

DISCUSSION OF SAMPLE SYSTEM OPERATION

The emphasis of this paper will be a discussion of the operation of the parser-analyzer (full details of other operations are available in Ben David[7]). In declarative mode, the natural language input is analyzed and formalized to the memory structure of this system. In the interrogatory mode, the natural language request is analyzed by the same procedures, formalized to the memory structure as a satisfy formulation rather than a modify formulation. Depending on the success of the satisfy formulation, either: (a) an answer is produced from the memory structure, or (b) a query of whether the system should continue search (something has been satisfied but more searching is necessary), or (c) a negative response is issued. A positive response to (b) results in generation of several variant satisfy formulations and a reiteration of the search.

PARSER-ANALYZER

The description of the parser-analyzer to be presented in this paper is intended to be complete enough to facilitate an in-depth comprehension of the system's operation and of how it differs from previous efforts. Due to space (and time) limitations (and in some cases unavailability of detailed documentation of the inner workers of other systems' operation) specific points of contrast to other systems cannot be given here (for overviews of other systems, see References 1, 2, 3 and 7).

** "The kings temple walls" is the internal representation of "temple walls of kings."

† "telled" is generated because "tell" is one of the words the system doesn't know at the beginning of operation. The word is picked up as a verb in the infinitive "to tell" and it is assumed to be a regular verb, therefore "telled."

‡ The system can deal with "first" as input but the present version cannot generate "first"—the verb is also misgenerated because of "one"—more discussion in Ben David.[7]

The process about to be detailed has its "roots" in the first version of QUANSY itself and before that in the front-end of the LEADER retrieval system, see Hillman.[8] However, the present process differs substantially from that of the previous versions. The explanation of the parser-analyzer will be presented as if the parser-analyzer were dealing with a particular sentence anywhere in text. It is not possible to simply start the description of the process; rather, there is an existing environment that must be described. At the very beginning of the analysis of a particular sentence (or throughput unit as described in Ben David[7]) there exists a temporary knowledge structure. This structure contains whatever information has been just previously analyzed and is grouped as a cohesive memory unit. This structure is already linked (see Reference 7), but has not yet been entered in the regular memory of the system. This is because it is expected that additional information may be added in the next few input sentences which will directly affect this structure. The decision-process about whether or not the present sentence relates to this knowledge structure is described in Reference 7.

MANGRAM (Manage Grammatical Analysis)

This routine has no linguistic rules; it is strictly the controller of the grammatical analysis. The analysis is a succession of applications of linguistically oriented routines which are called by MANGRAM in the sequence HYPSTRC, SUBDETR, VRBDETR, OBJDETR. At the end of the sequence, MANGRAM checks the status of the analysis. If analysis is complete, it terminates operation; otherwise it performs the sequence again until analysis is complete. However, analysis is not continued indefinitely in this manner. If the number of sequences necessary to analyze a particular sentence becomes too great, MANGRAM will terminate the analysis and issue an error diagnostic.

One potential (and very important) topic of future research is the design of an efficient default mechanism in a "looping" situation. Because of the complexity of natural language, many things can go wrong in an analysis procedure which result in endless "looping" in an unsuccessful effort to find the *best* solution. A default procedure could be designed to settle for any solution that appeared at all reasonable and not hope for some ultimate or best solution. It should be noted that there is a need for many default procedures in a natural language analysis and this one is just the main one.

HYPSTRC (Hypothesize Structure)

The idea for the present overall approach to the first part of the analysis is derived from some of the ideas discussed by Ulric Neisser, in his book *Cognitive Psychology*. Neisser[9] says "We deal with the sentences we hear by reformulating them for ourselves; we grasp their structure with the same apparatus that structures our own utterances." Neisser is here dealing with verbal communication; however, the same process applies to written language, though in a slightly different way. We can say that we understand language by making more and more sophisticated hypotheses about what a piece of language is, continually comparing our hypotheses to the real thing, and, if we find no contradiction, we continue. When our hypothesis matches the reality of the input text, then we have analyzed the input. This procedure is distinctly different from the grammatical procedure used in QUANSY 2.0 (see References 10 and 11 for more details on version 2.0). In that version, there was no way to analyze complex input (except for some embedded sentences). The previous procedure was sufficient for its limited application, but for the present extension of QUANSY's capabilities into complex and compound sentences and for formulation of cohesive memory units, the change in approach (described above) was necessary.

In considering the operation to be described, reference should be made to Figures 1 and 2. Figure 1 shows the overall parser-analyzer and Figure 2 expands the operation of HYPSTRC (the second level of hypothesis) and its interaction with SCAN (the first level of hypothesis), SCANMOD CONJFST and AUXFST.

In the first call to HYPSTRC, the routine SCAN is called. It makes the first hypothesis. HYPSTRC takes this first hypothesis and goes through section by section, mak-

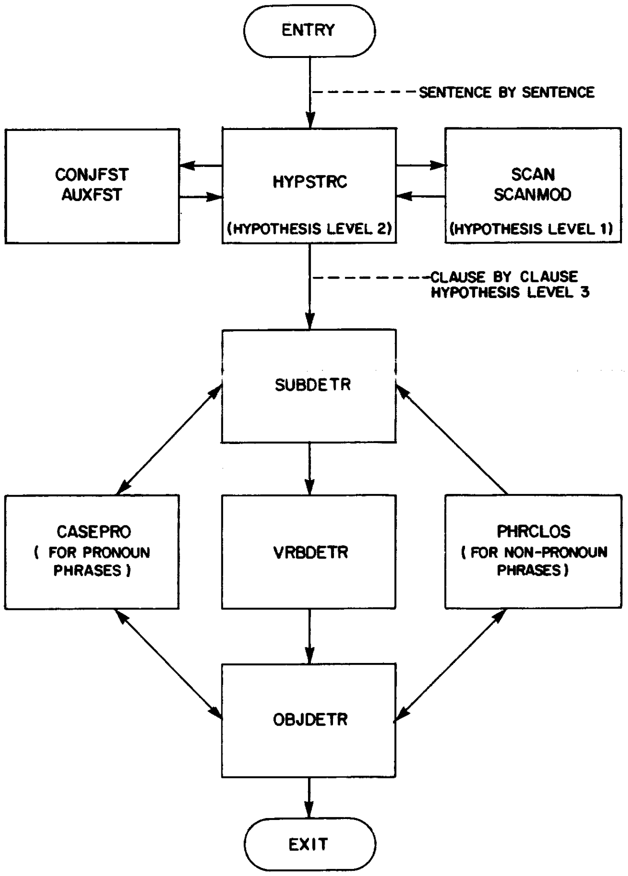


Figure 1—Parser-analyzer

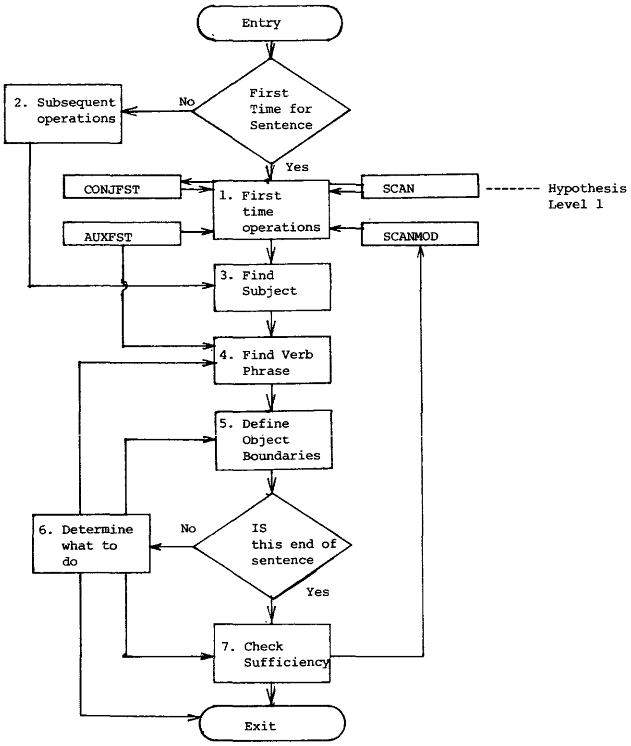


Figure 2—Expansion of hypothesis level 2 of Parser analyzer (each of the named boxes shall be referred to as a section)

ing a more detailed second hypothesis and passing the more detailed second hypothesis back to the system for verification. HYPSTRC makes a more detailed specification of the subject than of the predicate (this so as to make the determination of the beginning of the predicate as certain as possible). The verb phrase is grouped approximately (more or less certainly depending on the amount of ambiguity) and the rest of the present clause is lumped together into the "object"—subject to later processing.

(1) First time operations

The first time through for a particular through-put unit, usually a sentence, various initializations are performed—the subject, object, verb, conjunction, etc., pointers are all zeroed out. The routine SCAN is called to do the initial hypothesis. The initial clause boundaries, according to SCAN's hypothesis, are picked up and the first-time operations are complete (more detail on SCAN's operation is contained in the next few pages). If in the initial boundaries the first word is detected as a conjunction, then the routine CONJFST is called (details following).

(2) Other times operations

On successive times through HYPSTRC for the same throughput unit, there are similar initializations to the first time through, though necessary information must be saved. Consider the sentences:

The boy is eating the cake that the girl baked. (1)

The boy that baked the cake is eating. (2)

On the second time through for sentence (1), the object of the first clause "the cake" should be saved, while for sentence (2) the subject of the first clause "the boy" should be saved. After the decision about what to save is made (based on "start" and "embed" pointers set in the previous run-through), all pointers are initialized accordingly and the next clause boundaries are picked up from SCAN's hypothesis. (One very important initialization is the setting of the embed pointer to negative.)

(3) Find subject

To determine the subject of the clause, HYPSTRC proceeds word by word through the clause (as specified by the boundaries) looking for a verb, preposition, or determiner and ignoring other categories. In the process, it specifies the potential subject and distinguishes potential phrases in the subject. The subject is defined as all those phrases before the verb (including noun phrases, prepositional phrases, infinitive phrases, etc.). All words ignored are added to the present phrase under consideration.

(i) Verb

If a verb is found, is it auxiliary verb (form of to be, to have, to do, modal)? If this is auxiliary, assume that it signifies end of subject and beginning of verb phrase, exit this section. If not, note its location and continue. However, if there has been a previous verb noted thusly, retain the last if both are the same tense, retain the first if it is past tense, and retain the second (i.e., the one presently under consideration) if it is in present tense.

(ii) Preposition

If this is the first word in clause, continue (no operation). If not first word, it is "of"? If so, add it to present phrase (i.e., ignore it). For any other preposition, begin new phrase (but stay in subject—this section).

(iii) Determiner

If this is first word in clause, continue (i.e., ignore it). If the immediately preceding word is a preposition or determiner (i.e., "from the" or "the few"), ignore it. (The decision of what to do with this phrase was made in consideration of the last word.) If none of these circumstances apply, suspect that this determiner might be the beginning of the object. If a potential verb has been detected (non-auxiliary noted in section on verbs, above), accept it as the beginning of verb phrase and exit this section.

It is altogether possible that the present boundary of the clause will be reached before this section is exitted (i.e., either from the verb or determiner operations of the section "Find Subject"). The first question is whether there has been a potential verb (as noted in section on verbs). If there has been, take that verb as the beginning of the verb phrase and go on to the next section. If there hasn't been a potential verb, is this the end of a through-put unit? If this is not the end of the through-put unit, what is the nature of the boundary (usually the boundary will be a conjunction but not always)? If the boundary is a conjunction of subcategory 3 or less,* and the following word (after the conjunction) is not a determiner, ignore this boundary, set clause as up to next boundary, and continue processing from the beginning of this section (examples like "The boys and the girls" or "The red and black ball"). If the following word is a determiner, the boundary signals a new phrase: set pointers appropriately, pick up next boundary and continue processing from the beginning of this section.

If the boundary is a conjunction of subcategory 4 or greater, it signals the end of present phrase and embedded situation. Turn on embedded pointer (pointing to this location). Pick up next boundary and continue processing from the beginning of this section (as in example (2) above).

Not discussed in the description of this section is the handling of locational phrases and gerund phrases.** If the boundary reached is one of these (they are detected and grouped by SCAN—details following), it is entered as the type of phrase indicated, almost transparently with regards to the above section.

In the aforementioned example (1), the processing in this section would have reached "is", determined it as the beginning of the verb phrase and "the boy" as subject and processing goes on to the next section. For example (2), "that" is reached as the boundary, no potential verb has been detected and since the subcategory value of "that" is "4" the embeddation pointer is turned on. The next boundary is picked up, to the end of the sentence, and processing begins again. "Is" is detected as the beginning of the verb phrase and processing goes on to the next section.

(4) Find verb phrase

This section assumes the first word in the verb phrase has been found. It groups all adverbs and verbs into the verb phrase until it reaches a verb participle or a word not categorized as either a verb or adverb (the verb participle is grouped into the verb phrase, non-adverbs or non-verbs are not). If the first verb detected was an auxiliary and that verb was the first in the sentence, the routine AUXFST is called. After checking if the AUXFST was successful, operation is returned to section three, "find subject."

(5) Define object boundaries

The object is defined as everything from the end of the verb phrase to the present boundary. It is very possible that there is nothing between the end of the verb phrase and the boundary for this clause. If the boundary is the end of the throughput unit, then the next operation is the sufficiency check, otherwise the next step is section 6.

(6) Determine what to do

There are two distinct boundaries that can occur, conjunction of subcategory value 3 or less or conjunction of subcategory value 4 or greater.[6]

(i) Conjunction of subcategory value 3 or less.

Now there are two alternatives; either there has been a potential object or there hasn't been. If there has been an object, the portion between the present boundary and the next boundary is scanned for a verb. If one is found, and it is the first word in the portion, a check is made to see if it is the same type as the last entry in the verb phrase (isolated by section 4 above). If it is, it is considered another verb phrase. For instance, in:

Parts of pictures were carved on the slabs and painted. (3)

"Parts of pictures" is the subject, "were carved" is the verb phrase, "on the slabs" is the object (in the sense of object as defined in section 5—Define object boundaries), and "painted" would be detected by the above operation as an additional verb phrase.

If the verb is not the same type as the last entry in the verb phrase, this section tries to modify the category of this "verb" to noun and then ignores this word. If the verb is not the first in this portion of text, then is it an auxiliary? If so, this is the start for the next run-through. Add all words from one before auxiliary to object and terminate. If present clause is not embedded, this is a plain conjunctive situation; do not modify anything and terminate. If this verb is a non-auxiliary, has there been a previous potential verb? If so, process this second one like auxiliary (default situation). If there hasn't been another potential verb, note this as one (both in the above circumstance and the following one, the situation is not clear and in effect, a fuzzy hypothesis is made and OBJDETR will have to deal with the problem).

At the termination of the scan, first check if any potential verbs have occurred; if so default out performing auxiliary operation specified above. If there has been no potential verb, check the next boundary. If it is conjunction of value 3 or less, add this whole portion to

* "Subcategory 3 or less" conjunctions are 'and', 'or', 'nor', comma, 'both', etc., "subcategory 4 or more are 'when', 'that', etc." See Ben David.[7]

** Gerund phrase handling is only of a partial nature at present.

object, change boundaries of clause to include this portion and restart this section. If the conjunction is of value 4 or greater, this is an ambiguous path (the one between the present portion and the clause). For instance in the sentence:

The boy is eating the cake and the pie that had been bought was being eaten by the child. (4)

When reaching section 6, "the boy is eating the cake" has been processed. The portion "the pie" is being considered. Since "that" is the next conjunction, it is not at all clear whether "the pie" goes with the first clause or the second. Therefore, "and" is noted as an ambiguous path and nothing else is done—no modification to the clause. HYPSTRC is terminated.

If there hasn't been an object and the word following conjunction is not a verb or adverb, set this point as start for next pass through and terminate. If the word following conjunction is a verb or adverb, assume conjunctive verb. Reset clause boundaries to include this portion and continue processing from section 4, "find verb phrase."

(ii) Conjunction of subcategory value 4 or more

First turn embed pointer on. Is the conjunction "than"? If so, set "than" pointer and move modifier relating to "than" into verb phrase (i.e., bigger than, smaller than, more than, etc.). Set start pointer and terminate.

(7) Check Sufficiency

The first part of the check involves the value of the embed pointer. If it is less than or equal to zero, there is no problem and operation is terminated (if the embeddation pointer is zero, then this is a simple sentence; if it is negative, then there has been enough information). If the embeddation pointer is set, then the object grouping must be scanned for a potential verb. If a verb is found, is it an auxiliary? If so, modify object boundaries to the verb before auxiliary and return. If it is not an auxiliary, note it and continue scan. If at the end of the scan there has been a potential verb, make modifications, allow OBJDETR to complete determination. If no potential verb has been found, check for ambiguous paths. Has there been one? If not, issue error diagnostic and terminate; if there has been one—remove it and restart operation from first operation (call SCANMOD for removing boundary). For example, in (2) above, the verb phrase would be found as "baked" and the object as "the cake is eating." When this section was called, the embed pointer would be set and the end of the through-put unit reached. In scanning the object, "is" is found, the object is modified to "the cake" (and the start pointer is modified for next time through routine). In the sentence:

The boy is eating the cake and the pie that had been bought. (5)

the first time through, "The boy is eating the cake" would have been set as the first clause. The second time through, "the pie" would be set as subject, "had been bought" as the verb phrase, and then the sufficiency check would determine a lack of sufficiency. Backtracking to the previously noted ambiguity, "and", this path would be ruled out and processing restarted. In modified operation, "the boy is eating the cake and the pie" would be defined as the first clause, "The cake and the pie had been bought" as the second clause. (One important note—the problem of multiple ambiguous paths of this nature is not dealt with in this version except by successively backtracking).

SCAN

SCAN goes through a sentence word by word looking for a few central words which are"

(1) pronouns
(2) prepositions,
(3) conjunctions.

As the first part of the hypothesis analysis procedure, this routing has particular importance and is the most experimental. Its operation and justification for such will be described in some detail.

(1) Pronouns

The system distinguishes two main modes, question mode—when the user is querying the system, and declarative mode—when the system is analyzing declarative text. Pronouns are very important, particularly when the pronoun is first or second person and the system is in question mode. Very often the pronoun will be part of what can be referred to as the "Question-frame" or "Question-sign"—for example "What do you have about —", "I want to know (why)", etc. These occurrences are easily recognized and not passed on to the system except as question marks. When a question like this appears in declarative mode—i.e., as part of a text—for example "Joe asked 'What do you have on ice cream'.", then the "Question-frame" operation is not performed.

If a pronoun is detected, is this question mode or declarative mode? In declarative mode, ignore it. In question mode, set up the "question-frame" boundaries. Check the next three words. Is there a preposition? If so, set boundary to the preposition. If the second word following is also a preposition, set boundary to that preposition. If there isn't a preposition in the first three words following the pronoun, don't move this boundary at all. Set back boundary from previous conjunction, if there was one, or from the beginning of the sentence. Mark these boundaries as portion to be dropped.

(2) Prepositions

There are certain phrases which have well defined forms and which play important roles in a sentence (actually, as will be seen shortly, what is being discussed here is not so much a phrase as a particular type of grouping of words which can include several prepositional phrases plus non-prepositional phrases. However, the unifying aspect is that this grouping of words relates a particular bit of information, of varying detail and exactness, with respect to location of two or more objects with regard to each other). In dealing with such phrases in this version, work has focused on the *locational phrase*. This type of phrase is among the most common of such phrases and appears in many forms; detection and isolation of these phrases has been through the prepositions commonly used with them. Some of the forms of these phrases are:

south
south of Boston
to the south of Boston
miles south of Boston
miles to the south of Boston
one hundred miles south of Boston
one hundred miles to the south of Boston
one hundred and fifty one miles south of the city of Boston

The same basic form appears in phrases like:

right of the house,
to the right of the house,
etc.

In addition to its common appearance, it allows for some interesting inference experimentation and was therefore chosen for this version's work.

In order to detect the locational phrase, SCAN looks for the prepositions "of", "on", "to", and "from". It is certainly possible to generate a locational phrase without one of these prepositions (an example was given above—"south (of Boston)"), however, usually one or more of these prepositions will appear in the locational (in the case where none of these prepositions appear, the locational will be ignored—this is even though that potentially important information may be lost, for example "One hundred miles south is Philadelphia"). If the preposition , "of" is detected first, then we assume that this is the only preposition, otherwise we would have detected one of the others first. (Of course, there are locationals which have additional prepositions in their last part—"south of the City of Boston", "south of the highest point *in* the Rockies," etc.—the first "of" is of prime interest. If the grouping under consideration is considered a locational, then the problem of determining how far it extends is a separate problem from determining if it is a locational). If one of the other prepositions is detected first, then the routine looks for "of," which might not occur. In either case, a direction noun is critical; if one is found, then the question is to determine what the specific boundaries of the locational phrase are. Sometimes this decision is easy because the locational will be marked with commas; other times it is a process of preceding backwards and forwards from the detected prepositions looking for numeric and measure words in one direction and location specifics in the other. (Location specifics include names of places, location words—city, town, etc.) Once the boundaries of the locational are determined, it is treated as a unit. More than one locational can appear at one time—for instance in a compound locational. (For example "New York is one hundred miles north of Philadelphia and ninety miles east of Bethlehem.)

(3) Conjunction

This was the first consideration of SCAN. Originally, SCAN dealt only with conjunctions, and in a very simple manner. It soon became evident that additional processing routines were necessary in SCAN. First of all, several words which act like conjunctions but which are usually adverbs (i.e., "now"), especially when these words occur next to other conjunctions, can be modified immediately to save valuable processing time. (Actually, "now" is usually a modifier; however, it is sometimes used alone and the system must be ready. For instance—"Now (that) they have gone, the plans have arrived."—Usually "that" would appear with now"; however, "that" will be left out sometimes, with "now" assuming a conjunction role.) Second, SCAN attempts some preliminary work with comparatives, focusing on the occurrence of the conjunction "than" in examples like—"bigger than," "smaller than," etc. There has been much work on conjunctions detailed in the literature (particularly in comparatives), however, because of the general emphasis of the present work on a question-answering system, most of the previous work was not judged suitable to the present effort.

SCANMOD† simply takes direction from HYPSTRC with regard to turning off a specific boundary which was set in the first call to SCAN from HYPSTRC. It also sets an index value so that when SCAN is called by HYPSTRC (because processing is reinitiated after a call to SCANMOD and SCAN will be called in first run processing by HYPSTRC) nothing will be done by SCAN.

CONJFST-AUXFST

This routine has the responsibility of undoing permutations due to the interrogatory transformation. This transformation is detected either by the question pronoun as the first word ("what", "which", etc.) or by a lead auxiliary verb ("is", "did", "have", etc.) These cases are combined because they require similar actions. (First the details of the operations of CONJFST will be discussed and then the combined operation of CONJDST and AUXFST.)

(1) Conjunction first word

The various question pronouns are also used as conjunctions. For example, "What is the boy eating?" and "The

† The routine HYPSTRC calls in order to modify a clausal hypothesis.

boy is eating what he can." While their usages are different, their senses are similar. In the system's operation, all of the question pronouns are categorized first as conjunctions (of subcategory 4 or greater) and secondly as question pronouns. When a conjunction is detected as the first word of a sentence (by HYPSTRC), this routine is called and it attempts to modify the category from conjunction to question pronoun. If it cannot modify the conjunction (because this conjunction is not a question pronoun), it does nothing except note the conjunction.

If the conjunction has been converted to a question-pronoun, CONJFST then tries to determine the nature of the question. There are those questions where the only task is noting what the question-pronoun is—"What boy is coming?" For others, more work is necessary—"What is the mileage from Chicago to San Francisco." This is a very specific "what question—"What is the mileage" is the same as "How far is it," "How many miles is it," "How many miles is," etc. As much as possible about a particular question must be determined as early as possible in order to speed analysis and increase the accuracy of the response. The boundaries of the particular "question frame" are determined and subtracted from the input (ignored by the rest of the system).

Even if the first word (or group of words) is determined to be a question-frame, it is not certain that a permutation is also present. For example, "What boys are coming tomorrow?" and "What is eating the cheese?" The test for a permutation involves checking for a split verb phrase. This necessitates some auxiliary verb plus a participle. If the auxiliary does not follow immediately after the question frame, suspect a double permutation, for instance "How much gas does tank five contain?" "Tank five" and "does" must be interchanged and then, "How much gas" must be moved to the end of the question. When a double permutation seems likely, the second permutation section is called.

(2) Auxiliary first word

When an auxiliary occurs as the first word of the sentence, the first part of this routine, CONJFST, is skipped. However, just as in the last part of CONJFST, certain predictions are made vis-à-vis what permutations are expected. In this circumstance, a single permutation is expected.

(3) Permutation operation—one permutation (entry for AUXFST and for auxiliary after question frame from CONJFST).

In the permutation check, AUXFST scans for verbs, determiners, and prepositions.

(i) verb found

Is this verb an auxiliary or a verb participle? If so, does it follow the first auxiliary immediately? If so, no permutation—issue error diagnostic and return. If this verb does not follow first auxiliary immediately, set move unit from word after first auxiliary to word before present word. The entry point (i.e., where this move unit belongs) is before first auxiliary. For example, in "Are the boys eating cereal," "the boys" will be move unit and it would be moved before "Are" resulting in "The boys are eating cereal."

(ii) determiners

Is this first after auxiliary—if so skip it. If the word preceding this determiner was a preposition or another determiner—skip it (i.e., the decision about whether to continue the scan or not was made with the previous word). Otherwise, set move unit from word after first auxiliary to word before this one. The entry point is location before first auxiliary. For example, in "Are the boys the ones?", "The boys" is the move unit, the rearrangement results in "the boys are the ones."

(iii) prepositions

Is this first word after auxiliary—if so skip it. Has there been a preposition already? If not, note this one and continue. If there has been one, is this one "of"? If so skip it, otherwise note it in place of the previous preposition.

If the end of the scan is reached without a "move unit" established, check if there has been a preposition. If so, set move unit from word after first auxiliary to word before "noted preposition" and entry point in front of first auxiliary. For example "Are the boys from New York?" becomes "The boys are from New York." Note that "Are the boys from New York coming" would become "The boys from New York are coming."

(4) Permutation operation—two permutations (this section has only been developed to handle one type of situation—more work will follow).

When this section is called, it has no information of a first auxiliary. It first scans for the first auxiliary, notes it and then scans for a second auxiliary or participle (which can also be an auxiliary). If the auxiliary and the second verb are next to each other, no rearrangement is performed. The first move unit is from the word after the first auxiliary to word before second verb—entry point before first auxiliary, second move unit is from first word after question frame to word before first auxiliary—entry point after second verb. For example—"How much gas does tank five contain?"

SUBDETR—OBJDETR

These routines have certain basic similarities, but while the break-up of the constituent phrases in the subject has been taken care of by HYPSTRC to assist in accurate determination of the beginning of the verb phrase, none of the constituent phrases of the object have been isolated (it is important to remember that in referring to the subject and object, the reference is to all the various phrases before and after the verb phrase, as discussed earlier). There are various reasons for this differentiation; most notable is that until the verb phrase is fully determined— it is not clear where the object begins.

Both of these routines are concerned with filling the array PHRVALS. The following locations exist in this array:

1. Preposition
2. Determiner
3. Number—first location — either cardinal or
4. Number—last location — ordinal values + fractions
5. Modifiers—first location
6. Modifiers—last location
7. Noun—main noun of phrase
8. Conjunction
9. Infinitive
10. Adverbs

When these routines have completed operation of a particular phrase, this array (sometimes in multiples) is passed to PHRCLOS (discussed later). However, if pronouns are detected, the routines CASEPR1 or CASEPR2 are called (1 for subject, 2 for object) and PHRCLOS is not called.

The operation is pretty straightforward setting up the array. The exceptions are as follows

(1) Preposition

Prepositions are handled basically the same, except for the following difference in these routines.

(i) SUBDETR

> SUBDETR expects that every preposition will be the first one in its phrase, except for "of". When "of" is detected, a rearrangement is performed to make the words following "of" modifiers of the previous phrase. For example, the phrase "the slabs of alabaster" is converted into "the alabaster slabs." This conversion is carefully noted in case there is some need for the original form, for example "the slabs of alabaster, which is a stone, were transported." For the primary clause, the system wants "the alabaster slabs", for the relative clause, it wants "alabaster". It is important to note that there is a potential problem of ambiguity in these circumstances—while above it is clear that "alabaster" is the subject of "is a stone," for the sentence "The slabs of trees which are a building material were transported," is "slabs of trees" or "trees" the subject of "are a building material"? To disambiguate this the system needs semantic information. Presently, "trees" would be determined as the subject.

(ii) OBJDETR

> OBJDETR treats "of" basically the same as SUBDETR; however, any other preposition encountered is considered the basis for a phrase break unless it is the first word in the object. If it is "from", check for "to" after "from" (i.e., "from Chicago to New York"). In case of a phrase break, the routine PHRCLOS is called.

(2) Conjunctions

Conjunctions have the same impact in OBJDETR and in SUBDETR, except that (just as with prepositions) in SUBDETR there is less expectation that the conjunction might signify a new phrase, as the calculations on that point would have been done in HYPSTRC. In both routines the location of the conjunction is noted in PHRVALS (8) and the phrase indicator is incremented—new information occurring will be entered in another level of PHRVALS, not interfering with the previous information entered therein.

PHRCLOS

PHRCLOS looks at the structure of each phrase (as sent by SUBDETR and OBJDETR) in the array PHRVALS and completely categorizes the phrase. If there is a determiner, it picks up the determiner code and stores it for the phrase. It determines whether a phrase with a conjunction is two phrases or one phrase with a compound modifier. If there are two phrases, does the same determiner and modification apply to both phrases? Is there a number present? If so evaluate it. Is there a preposition? If so, what kind of prepositional phrase is this (CASEVAL)? Based on the information it gathers, it decides whether the phrase is singular or plural.

CASEVAL

This routine is called to calculate the nature of a prepositional phrase. In roder to deal with the great amount of information present in nouns, this routine employs decision tables. Using the information contained in the preposition and the noun, this routine determines what a particular phrase specifies about a sentence.‡

CASEPR1-CASEPR2

This routine (the different names just represent different entry points) calculates the antecedents of pronouns using positional, number, and type (human, non-human) cues.

The routine has two knowledge structures to consider. One is the one presently being built for this sentence (a through-put unit). This knowledge structure is not linked

‡ For example, in the prepositional phrase "in the house", since "in" is subcategorized as 5 and "house" as 21, this prepositional phrase is classified as a "where" phrase.

and will only be completed when the whole sentence has been processed; however, it is available for checking by this routine. For instance in the sentence, "The artists used the pictures to tell stories about the kings and what they did", from the beginning of the sentence to "and" would be processed as the first clause and "what they did" as the second. This is an ambiguous situation and the routine would settle for "the artists" as the referent of "they" (based on positional cue). The other knowledge structure that this routine has access to is the one this sentence fits into, provided there is one. It is entirely possible that neither of these structures is available (i.e., at the beginning of a particular text). If neither of the knowledge structures contains any information, the routine will issue diagnostics unless the referent of the pronoun could be following (i.e., "It is the boy who is responsible.")

VRBDETR

VRBDETR isolates the verb phrase, determines the tense and transformations, and takes care of adverbial modifications of verb whether negative or positive. This routine makes extensive use of decision tables to facilitate fast operation. It is distinctly possible that the determination of the verb phrase made by HYPSTRC will be inaccurate—usually overlong. VRBDETR will modify the object boundaries by moving any information it doesn't need into the object. This happens particularly when new words are introduced or a word is used without a determiner preceding it (i.e., as a predicate adjective, as in "The boy is sick" where "sick" is treated as a noun with associated subcategories—see previous discussion).

CONCLUSION

The parser-analyzer for QUANSY 3.0, as presented in this paper is of straightforward syntactic and semantic nature (though predominantly syntactic). As it stands right now, it is able to handle levels of English up to a fourth-grade level at relatively high speeds. Most important, the parser-analyzer, as presented here in depth, can be seen to be distinctly different from other approaches to natural-language analysis. With the introduction of additional semantic capabilities, the system should be able to achieve substantially higher levels of performance.

REFERENCES

1. Cuadra, A. C. and A. W. Luke, *"Annual Reviews of Information Science and Technology,"* Vols. 4, 6, 8—ASIS, 1973, Britannic, 1971, 1969.
2. Wilks, U., *"Grammar, Meaning and the Machine Analysis of Language,"* Routledge and Kegan, London, 1972.
3. Winograd, T., *"Understanding Natural Language,"* Academic Press, 1972.
4. Brown, J. S. and P. R. Buston, "Multiple Representations of Knowledge for Tutorial Reasoning," In: *Representation and Understanding,* Eds.: Bobrow, S. and Collins, A., Academic Press, 1975.
5. Lehnert, W., "What makes Sam run?", in *Theoretical Issues in National Language Processing,* Eds.: Schank, R. C., and Nash-Webber, B. C. ACL, 1975.
6. Coles, L. S., "Techniques for Information Retrieval Using an Inferential Question-Answering System with National Language Input," Stanford Research Institute, Menlo Park, California, 1972.
7. Ben David, A. S., "A Question-Answering System (QUANSY) for Information Retrieval," PhD dissertation, Lehigh University, Bethlehem, Pennsylvania, 1976.
8. Hillman, D. J., "The Leader Retrieval System," *AFIPS Conference Proceedings,* Vol. 34, 1969.
9. Neisser, U., *"Cognitive Psychology,"* Appleton-Century Crofts, New York, 1967.
10. Hillman, D. J., "A Study of Information Regeneration for Knowledge Transfer," Final Report to Division of Science Information, NSF, April 1975.
11. Ben David, A. S., "QUANSY, A Natural-Language Question-Answering System," Master's Thesis, Lehigh University, 1975.

Automatic generation of computer programs*

by NOAH S. PRYWES
University of Pennsylvania
Philadelphia, Pennsylvania

ABSTRACT

This is an introduction and summary of research on Automatic Program Generation conducted at the Moore School, University of Pennsylvania. This research culminated in development of a Module Description Language (MODEL) designed for use by management, business, or accounting specialists who are not required to have computer training. MODEL statements describe input, output, and various formulae associated with system specification. No processing or sequencing information is required from the user. A MODEL Processor analyzes the specifications and interacts with the user in resolving inconsistencies, ambiguities and incompleteness. A program for performing the required functions is then generated based on the "complete" specification.

INTRODUCTION

General description of a system for generating computer programs

A major aspiration of computer programming language designers has been to make programming so easy that large ~~classes of educated people who have not been exposed to~~ computer training are able to program. For instance, in 1960 the CODASYL committee designed a programming language named COBOL-COmmon Business Oriented Language. Despite this and many other attempts to reduce the complexity of the programming process, it has continued to require considerable skill and specialization. Currently there are a large number of Application Programmers who handle such tasks. The standard procedure is for a "user," whether manager, accountant or business specialist, to communicate requirements to an Application Programmer who, in turn, composes a program to fit these requirements. The research described in this paper is a continuation of efforts to make feasible the preparation of programs by users (interacting with an automatic program generator) without recourse to middle-men Application Programmers. In view of the historical elusiveness of this goal, it is approached with considerable trepidation.

Figure 1 illustrates the overall concept of interactions between the automatic computer program generator and the user. (The components in the diagram are referred to by the indicated numbers). The *user* (1) is viewed as an individual who is proficient in the immediate field in which the programs are to be applied. Namely, he is viewed as being in a management or a technical capacity, such as in accounting, production control, etc. He must not only have had professional training in this specialized area of activity, but also have had a good mathematical background. The user is not required, however to have had any specialized computer training, but he must understand that when a function is properly specified it can then be executed by a computer.

The user composes statements (3) (via a terminal and a Text-Editor (2)) in a language named MODEL (MOdule DEscription Language). Each statement is considered an integral unit and contains a "chunk" of information. A statement may describe an item of data (*data description*) or an algebraic or logical relation among items of data (*assertion*). References may be made to statements previously entered in a data base (4) by the user, by others who have specified requirements for similar applications, or by others with whom the user wants to share data. The MODEL Processor (5) analyzes the totality of statements transmitted to it and, as appropriate, solicits from the user additions or changes to these statements. It provides a user with listings, cross-references, and requests for additions or changes necessary to resolve incompleteness, ambiguities or inconsistencies (6). Finally, the analysis by the processor leads to certain logical implications which are also communicated to the user to enable him to clarify and self-check his specifications of the requirements. When all outstanding problems have been resolved in this dialogue, the processor produces a program in a computer language. An Optimizing Compiler (7) produces the object code (for the application program) which is loaded into a digital computer (8) for execution.

The system description in this report is based on an operational MODEL II system[1] which incorporates corrections and improvements consisting of new language, sequencing and iteration analysis, over a previously described language and processor.[2] Work is under way to reprogram

* Research supported by the Information Systems Program, Office of Naval Research under contract N00014-67-A-0216-0014, Project NR-049-153.

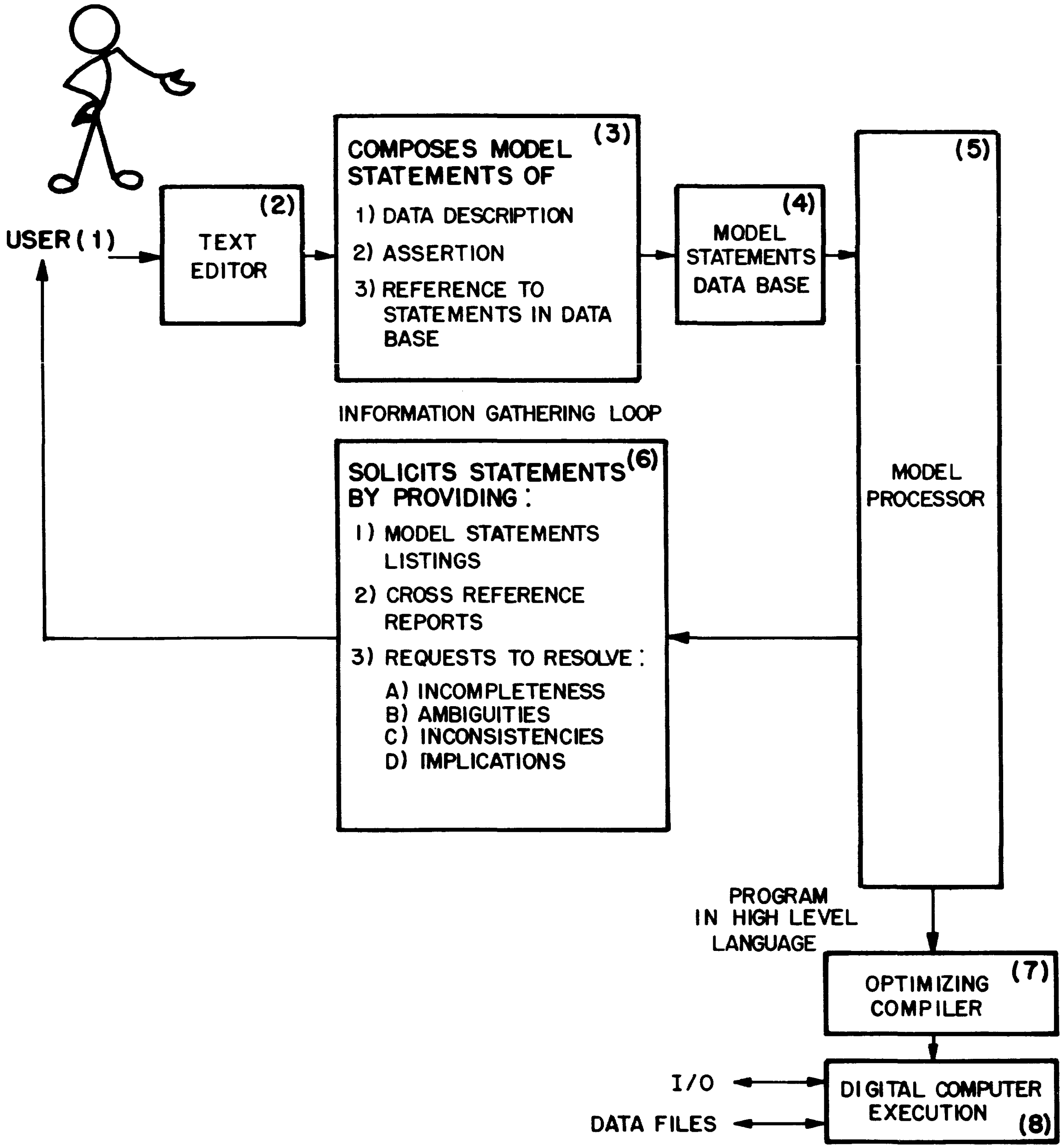

USER (1)
(2) TEXT EDITOR
(3) COMPOSES MODEL STATEMENTS OF
1) DATA DESCRIPTION
2) ASSERTION
3) REFERENCE TO STATEMENTS IN DATA BASE
(4) MODEL STATEMENTS DATA BASE
(5) MODEL PROCESSOR
INFORMATION GATHERING LOOP
SOLICITS STATEMENTS BY PROVIDING: (6)
1) MODEL STATEMENTS LISTINGS
2) CROSS REFERENCE REPORTS
3) REQUESTS TO RESOLVE:
A) INCOMPLETENESS
B) AMBIGUITIES
C) INCONSISTENCIES
D) IMPLICATIONS
PROGRAM IN HIGH LEVEL LANGUAGE
OPTIMIZING COMPILER (7)
I/O
DATA FILES
DIGITAL COMPUTER EXECUTION (8)

Figure 1—Illustration of the concept of automatic program generation

the processor to reflect additional improvements.[3,4] The system is programmed in PL/1 and produces object programs in PL/1.

In the interest of brevity, the paper describes the use of the MODEL language in the second section of this paper, and the analysis leading to solicitation of additional user information in the third section. The automatic program (and flowchart) generation phases of the MODEL system are omitted and described in the referenced reports. A survey of related research can be found in References 4 and 5.

Distinctive characteristics of MODEL

The MODEL language described in this paper incorporates several new characteristics not existing in previous programming languages and which were adopted because of their distinct advantages over past practices. These characteristics are explained below in the context of the system illustrated in Figure 1 where a user communicates with a processor that generates programs.

The new labor saving characteristics that have been incorporated in this approach are as follows:

1. Non-proceduralness—means that the user need not (and cannot) specify any order of evaluation or memory assignments. The "control logic" parts of the ultimately produced program, which are based on such procedural information, are to be deduced by the MODEL Processor. This feature is considered important, not only because it saves programming labor but also because it reduces the necessary computer training of the user. For instance, the user does not need to have such basic concepts as flowcharting or memory.

2. Independence of statements—means that the user can concentrate on composing a single statement at a time. It is neither required nor possible to indicate any relationships among the statements (except implicitly, such as when specifying relationships among variables). A single statement is required for describing each data name, or each formula. Modification or addition of statements can be carried out independently, one statement at a time, in the same manner.

3. Randomness—takes into account that information may originate from a group of users. Also each user's concept of computer requirements is usually not well organized, and a variety of information comes to his mind at different times. The user can describe this information in statements, one at a time, in the order that the information occurs to him. While a certain organization in this approach may be helpful, it is not required.

4. Incrementality—means that once users have provided a certain portion of the totality of the statements, the processor should be able to solicit additions or changes incrementally until a *complete* specification of the computer requirement is obtained. In this manner, it should be possible to avoid the problems of ambiguities or incompleteness that lead to major misunderstandings between the users and Application Programmers, and which require costly corrections and reprogramming.

5. Self Documentation—is attained as the documentation is generated during the dialogue between a user and the Processor. The additional documentation of the corresponding flowchart can be generated by the Processor automatically when generating the program. These latter types of documentation would normally not be of interest to a user, but rather to a Systems Programmer. The documentation of the user's computer requirement and of the associated program to be produced by MODEL is comprised of the collection of the corresponding statements together with the cross references and summary tables and comments produced by the Processor.

6. Maintenance—involves corrections of the programs based on malfunctions discovered during operation, or on modifications of the specifications to meet new needs. In current practice the modifications must also be performed by a middle-man Application Programmer. It is envisaged, instead, that the user would make the changes in the MODEL statements to reflect either the corrections or the modifications, whereupon the Processor would generate a new program automatically.

7. Sharing—of data or computations can be attained by storing the corresponding MODEL statements in the Processor's data base. A user desirous of sharing the know-how of others who have previously stated requirements in similar application areas needs only to reference these statements in order to incorporate them in the specification of his program. Data bases could be physically shared, while computations would be repeated in each user's program. To make changes in the organization of shared data bases, the data description statements must be modified or added, and previously generated programs based on these statements must be automatically regenerated. In this way changes to shared data bases or programs can be carried out without requiring the users to modify their programs individually.

8. Tolerance—The Processor is tolerant of the user's ambiguities and omissions. To fully specify a requirement would necessitate composing many statements which may appear to a user to be self evident and superfluous. The Processor in synthesizing the MODEL statements into a program must recognize the resulting ambiguities and omissions and generate the necessary additional MODEL statements automatically, thus relieving the user of much tiresome detail.

THE LANGUAGE

Example

An example is used in the following pages to illustrate how a user describes a requirement which he wishes to automate.

The example envisages the environment of a department store with many departments, a large number of charge account customers, and an extensive and diverse stock inventory. Point-of-Sale terminals connected to a network of computers are distributed through the several locations of the department store. The user of the MODEL system is envisaged to be a department store analyst who desires to specify the accounting requirements for purchases by cash and charge account customers.

Figure 2 gives an overview of the accounting requirement. The corresponding program module is named *DEPSALE,* and is shown at the center of the figure.

The data for DEPSALE comes from three *sources*. The sales transactions (*SALETRAN*), come from a Point-of-Sale terminal (*POSTERM*) sequentially, one at a time, and contain the information provided by the purchasers. The customer data (*CUSTMAST*) contains records of customers which can be referenced by providing customer numbers. Finally, there is inventory (*INVEN*) data, where information on stock items can be referenced by providing a stock

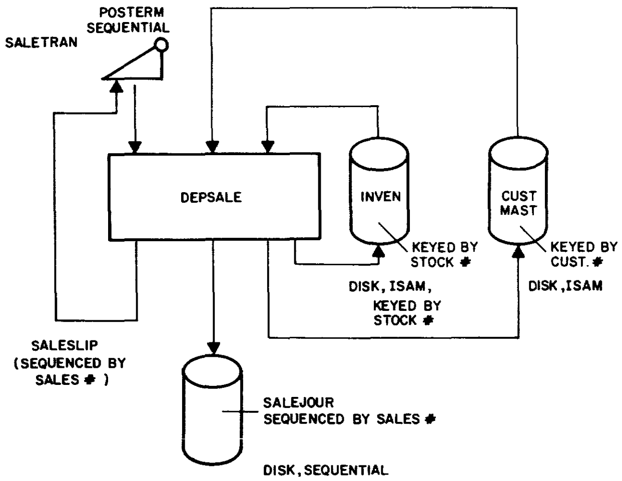


Figure 2—Illustration of a department store sale accounting (depsale requirement)

number. The data that comes in to DEPSALE is referred to as *SOURCE* data.

The *TARGET* data consists of the records in CUST-MAST and INVEN which are affected by the sales transaction and must be updated. Other TARGET data are the entries made in a sales journal (SALEJOUR) which are ordered sequentially by sales numbers (SALES #). Finally, a sales slip (SALESLIP) is produced on the terminal in cases where the sales transaction has been consummated or, alternatively, an exception notice (EXCEPT) is produced when the transaction does not take place.

An outline for preparing requirement descriptions

Figure 3 shows, in outline form, the information that needs to be provided in describing a requirement to be automated. As indicated, the user does not need to follow this outline, but can provide the information in any order. With the aid of a Text Editor, the user can enter statements and organize them into sections, subsections, etc. At the highest level the description is divided into three sections: the header, the data description, and the computation description.

The header contains identification information: module name(1), source(2) and target(3) data names, and references to sections or subsections of computation description statements, (called assertions) that are in a library in the data base(4). The latter may represent standards of data formats and organizations or any previously entered statements. The user is able to specify more complex operations that would be applied to the statement-data-base, to produce new statements to be incorporated in the specifications of a desired program module.

Data description is independent of the computation description, so that the data may be shared by several programs. Data and computation descriptions may be in a library, and called for automatically, to facilitate sharing of data and computations. Data description breaks down into the descriptions of the individual files, inputs, or reports(5) (each forms a subsection). The description of each data source or target is divided into the description of the storage medium, the data, and a set of assertions. The data description assertions are used dynamically to evaluate data dependent structures, such as length of fields, number of repetitions of optional data structures, and to describe intra-record (intra-file or inter-file) references (to be described further).

The computation description consist also of several subsections. First is the description of internal variables, that are not included in the source and target data descriptions(7). Next are assertions that specify subsets of source and target data that will be processed (8 and 9). Finally there are the descriptions of accounting and business decision practices(10).

All descriptions are in the form of statements. There must be one statement for each data name, for each medium used to communicate data and for each assertion.

Data description statements

Data statements

A *data network* concept is employed. Its application to the DEPSALE program is illustrated in Figure 4. First, a user has to name each data structure (to be further explained) and compose a statement for each name used. Each of the source or target files, inputs or reports is organized internally in a hierarchical structure resembling a

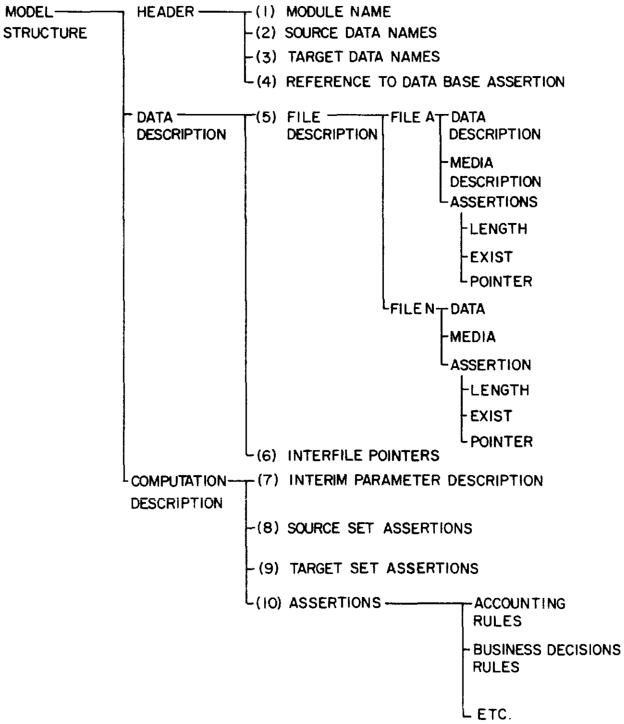


Figure 3—Outline of information provided in describing a requirement

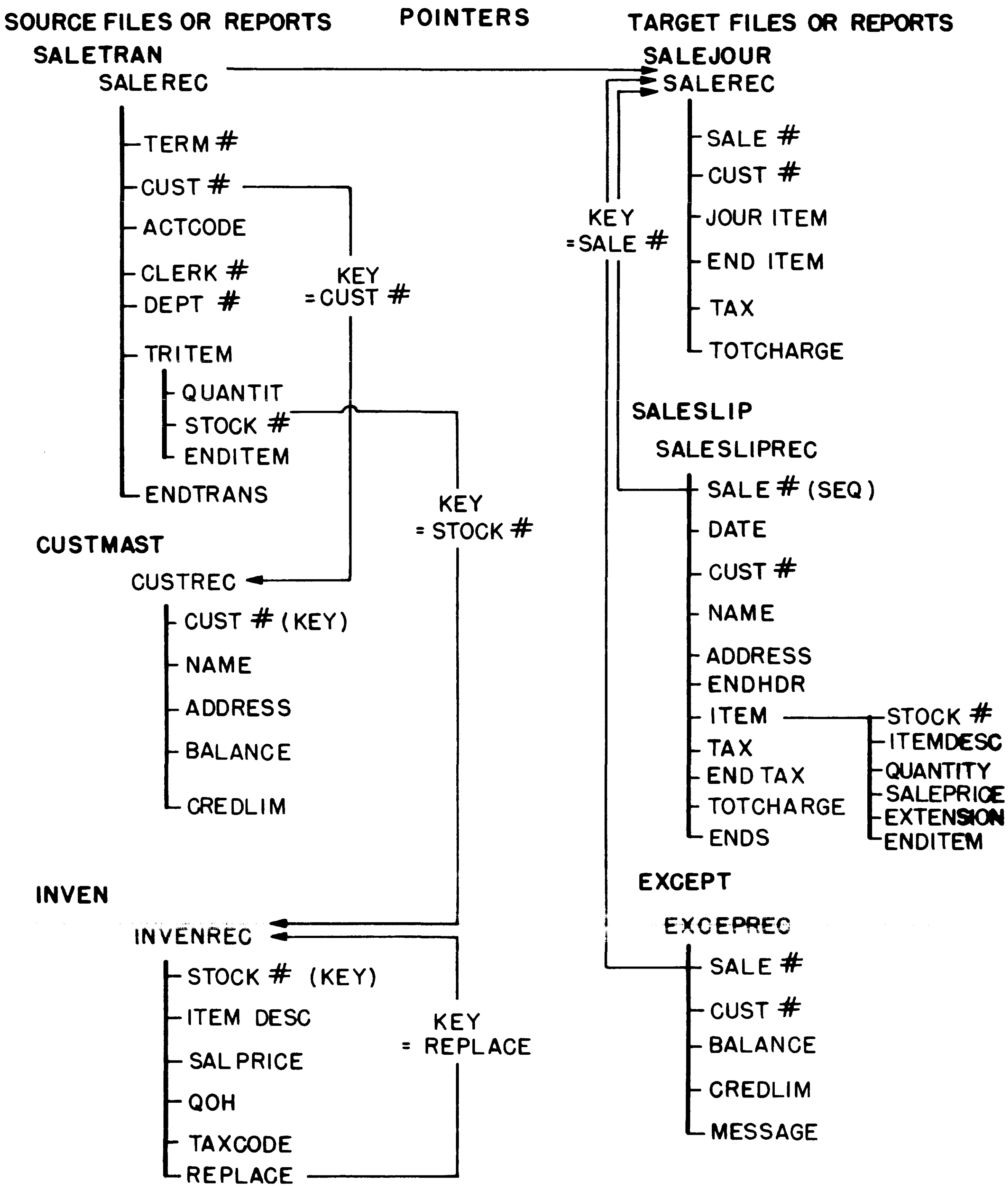


Figure 4—Data network illustration for depsale

tree. Each node of the tree must be given a name. The user must compose a statement corresponding to each data name, in which he provides associated information on the branch connection, data length, number of repetitions and other parameters of source and target media. Network (non-tree) structures are described by use of POINTER type assertions which coordinate instances of repeating data.

For example, consider the sales transaction (SALETRAN) source data at the top left of Figure 4. The STORAGE name is a POINT-of-SALE Terminal—POSTERM, (not shown in Figure 4). The SALETRAN file (an incoming message, referred to here as a FILE) is describable as a tree structure. The name of the FILE is at the apex, emanating from it is the RECORD (SALEREC) node. The branches emanating from the RECORD node lead to GROUP or FIELD nodes. GROUP nodes are not terminal nodes, and the branches emanating from GROUP nodes can again lead to GROUP or FIELD nodes. The terminal nodes are always referred to as FIELDs.

Figure 5 shows the data description statements of SALETRAN. The words, RECORD, STORAGE, GROUP or FIELD are followed by their respective parameters, in parentheses. The STORAGE parameters depend on the devices specified. For POSTERM they are the format, unit number and block size. The parameters of FILE, RECORD or GROUP are the data names of the descendent nodes. Another parameter of FILE is the name of the referencing or sequencing field, if any. Parameters of FIELD are data length and number of repetitions (if more than 1). These parameters can be constants or variables. If they are variables, LENGTH and EXIST type assertions must be provided separately to specify how they can be computed.

Data description assertions

A number of assertion types are needed to describe the data further. They concern length and number of repetitions of data and inter or intra file pointers. Figure 6 illustrates these assertions.

The number of transaction items (ITEM) in a sale transaction is a variable (EXIST type) named EXIST.ITEM.

Assertions specifying the computation of these variables are shown at the top of Figure 6. For instance, number of repetitions can be determined from the position of delimiter characters. The delimiter for end of transaction is the field ENDTRANS. It is assumed to be a Ready symbol, R. The first assertion in Figure 6 specifies the calculation of the number of repetitions of the group TRITEN. It uses the *function* INDEX which evaluates a string of characters to determine the position of the R symbol in relation to the beginning of the strings of the SALEREC record.

```
SALETRAN IS FILE(RECORD IS SALEREC, STORAGE IS POSTERM)
   POSTERM IS TERMINAL(VARIABLE, MAX_RECORD SIZE = 150, UNIT = 2741)
   SALEREC IS RECORD(TERM#, CUST#, ACTCODE, CLERK#, DEP#, TRITEM(1:10
                                                        ENTRANS))
   TERM# IS FIELD (CHAR(5))
   CUST# IS FIELD (CHAR(7))
   ACTCODE IS FIELD (CHAR(1))
   CLERK# IS FIELD (CHAR(2))
   DEP# IS FIELD (CHAR(5))
   TRITEM IS GROUP (QUANTIT, STOCK#, ENDITEM)
      QUANTIT IS FIELD (CHAR(3))
      STOCK# IS FIELD (CHAR(6))
      ENDITEM IS FIELD (CHAR(1))
   ENDTRANS IS FIELD (CHAR(1))
```

Figure 5—Data description statements for SALETRAN

```
EXIST.TRITEM = (INDEX(SALEREC, R)-21) / 10 ;

POINTER.CUSTREC = SALETRAN.CUST# ;

POINTER.INVENREC (FOR_EACH_TRITEM, 1) = SALETRAN.STOCK# (FOR_EACH_ TRITEM);
```

Figure 6—Illustration of assertions associated with data description

Figure 6 also shows the assertions which specify that fields CUST# and STOCK# in SALEREC can be used as pointers to the customer master and inventory files as illustrated in Figure 4. Note that POINTER.INVENREC has two subscripts, the first is FOR_EACH_ITEM, and the second is '1'. As will be seen, the pointer value may be derived also from the REPLACE field of INVENREC.

The other source and target data indicated in Figures 2 and 4 can be described in similar manner. In the interest of brevity the discussion of this data is omitted.

In reproducing a listing of the submitted statements, the Processor names all data description statements and assertions and identifies source and target variables, unless already specified by the user. The names assigned to statements are a derivation of the names of the data elements described or a derivation of the names of the dependent (target) variable in assertions.

Computation description statements

Composition of assertions

Following the outline in Figure 3, the computation description consists first of the description of interim variables in a manner similar to data description, except that these variables are stated to be INTERIM.

In addition to describing interim variables, the description of computation consists primarily of statements with logical or/and arithmetic constructs which are also referred to as *assertions*. Using arithmetic and logical operators, as well as *functions*, the user composes such statements to specify relationships among variables.

In composing an assertion it is necessary to separate the dependent and independent variables. One common convention is to place the single dependent variable on the left of the equal sign (=). Note that the = sign means algebraic equality and not assignment.

In composing assertions, the user specifies relationships using mathematical, non-procedural notations. Many relationships, not directly expressible using arithmetic and logical operators, must then be expressed using *functions* that map the SOURCE data into the TARGET data. These functions are a substitute for established mathematical notation. (an example is the Σ symbol, meaning "summation" which is illustrated further below). Other functions evaluate character strings (such as the INDEX function described above). The use of functions in assertions requires stating the name of the function followed with the specification of the parameters, enclosed in parentheses.

The functions return a value (which may be a single variable, or components of a vector or an array).

SUBSET assertions

Examples of SUBSET assertions are shown in Figure 7. For example, the first assertion specifies that only transactions from terminals SALE2 through SALE5 and from clerks C5 through C7 are to be processed. As shown, it is applied to the SALETRAN source data.

The second assertion in Figure 7 applies to target data EXCEPT. It specifies that entries in this target file be limited to cases where the balance (BALANCE) would exceed the credit limit (CREDLIM).

Illustration of computational assertions

As indicated in Figure 3, assertions can be used to specify relations exemplified by accounting rules or business decisions. Figure 8 gives four examples of such assertions (the total number in DEPSALE is 20) and illustrates several features of assertions:

The first assertion in Figure 8 specifies the evaluation of the EXTENSION field which is the dollar value of purchased stock items of one type. The subscript notation (FOR_EACH_ITEM) after a variable means that it can have several components and by implication, that the process will be repeated a variable number of times corresponding to the number of repetitions of ITEM.

The second assertion in Figure 8 illustrates the use of the SUM function to specify the evaluation of the total charge made on a purchase sales slip (TOTCHARGE). The SUM function has a parameter of the variable to be summed (EXTENSION). (Another assertion not shown in Figure 8 must specify the calculation of TAX).

The third and fourth assertions in Figure 8 illustrate business decision rules. For instance, if an item in the sale transaction is out of stock, namely the Quantity On Hand (QOH) field is smaller than the quantity specified in the sales transaction (QUANTIT) then it is desired that a suitable substitute item, if any, should be sold. The stock number of a suitable substitute item is stored in the inventory record (INVENREC) in the field named REPLACE (see Figure 4).

```
IF TERM # > SALE2 AND TERM # < SALE5 AND

          CLERK # > C5 AND CLERK # < C7  THEN SUBSET.SALETRAN = SELECTED

                                          ELSE SUBSET.SALETRAN = NOT SELECTED;

IF CREDLIM > BALANCE THEN SUBSET.EXCEPT = SELECTED

              ELSE SUBSET.EXCEPT = NOT SELECTED;
```

Figure 7—Subset assertions

```
EXTENSION (FOR EACH ITEM) = SALEPRICE* QUANTIT (FOR EACH ITEM);

TOTCHARGE = SUM (EXTENSION (FOR_EACH_ITEM)) + TAX;

IF QUANTIT <  QOH THEN CHOICE.SUBSTIT = SELECTED

                  ELSE CHOICE.SUBSTIT = NOT-SELECTED;

IF CHOICE.SUBSTIT THEN POINTER.INVENREC (FOR_EACH_TRITEM, 2) = REPLACE

                  ELSE POINTER.INVENREC´(FOR_EACH_TRITEM, 2) = NULL ;
```

Figure 8—Examples of computational assertions for the DEPSALE example

To represent decisions, the user can use a variable with the name of the decision prefaced by the word CHOICE. All such variables can have only two values, SELECTED and NOT-SELECTED. They will be described automatically by the Processor (see a later section). The third assertion in Figure 8 shows the expression that specifies when the substitution is to take place.

The last assertion in Figure 8 specifies the implementation of the decision, namely the stock number in the REPLACE field is treated in the same way as if it were the stock number in the sales transaction. This requires a new value for the POINTER.INVENREC field (see Figure 8). This operation is expressed by use of subscript.

The above assertions are representative of some of the relationships that can be expressed by assertions. The library of functions is open-ended and additions can be made easily to accommodate special needs. However, it is important to restrict the number of functions and to have their operations similar to common mathematical notation in order to assure ease in user familiarization with them.

Reporting formatting assertions

The description of messages or reports in MODEL is similar to that of information stored in computer storage media. The user always views the information as a string of information divisible logically into records, groups, and fields. However, in the specification of messages or reports he has also to consider the continuity and availability of physical space. Additionally, the internal order of data substructures can be specified by the sequence of submission of the corresponding data statements to the Processor.

In describing the format of a report, the user must consider tab, carriage return or new page symbols as if they were data fields. In source data, these formatting symbols would already have the desired values. In target data, the obtaining of the values of these data must be specified by assertions. These values are frequently data dependent. Figure 9 illustrates this by showing two assertions that

```
ENDITEM (FOR_EACH_ITEM) = STRING (CR, 1);

END-TAX = STRING (CR, 12 -EITEM);
```

Figure 9—Example of report formatting assertions

compute the number of carriage returns used in the saleslip (SALESLIPREC, see Figure 4) after printing ITEM groups. Normally, one carriage return symbol after each item line suffices, except after the TAX line when it is desired to advance to the 12th line of the saleslip, where the total charge (TOTCHARG) is printed. The function STRING generates a string, consisting of substrings (CR) specified in the first parameter, which are repeated a number of times specified in the second parameter (12-EXIST-ITEM).

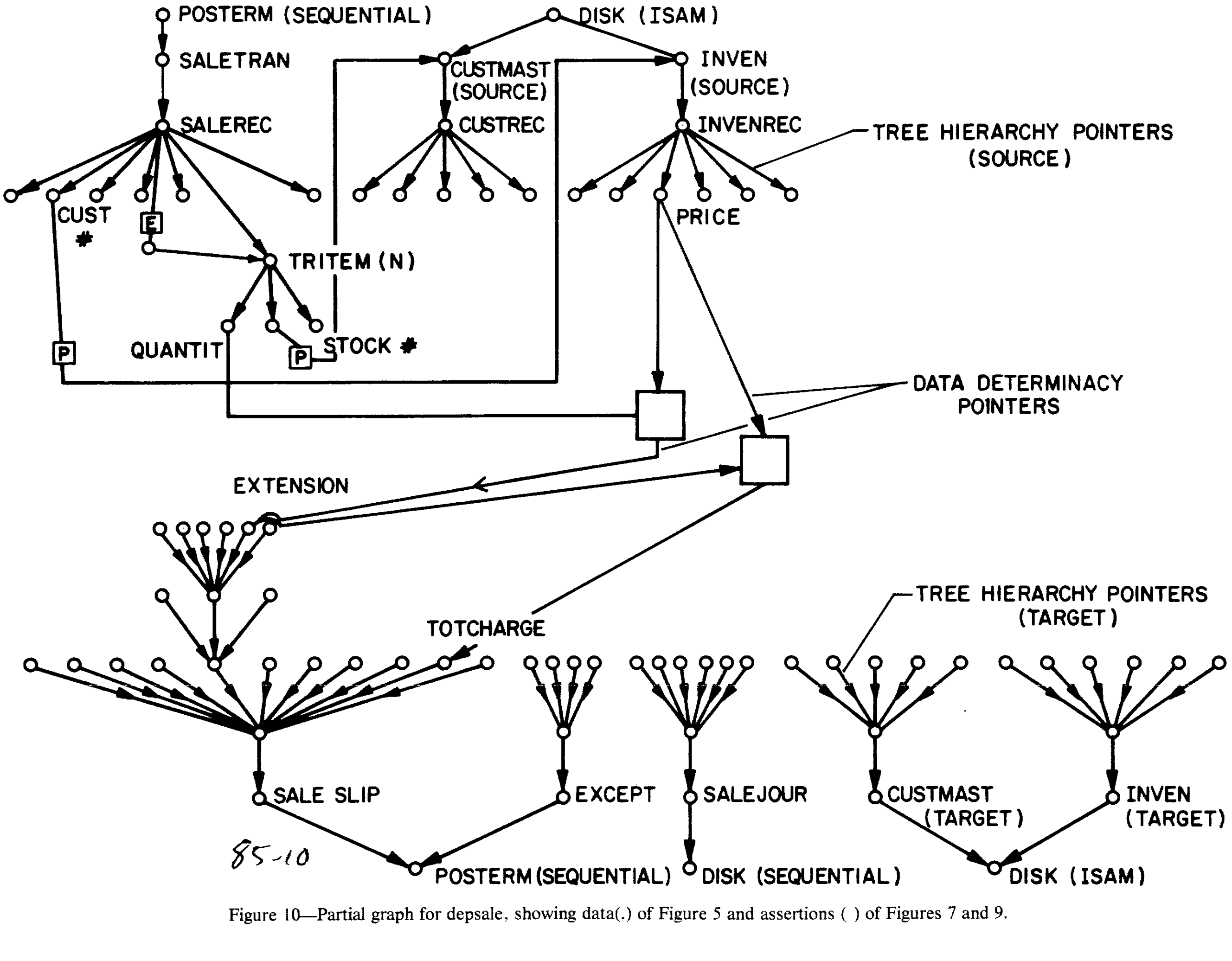
This task of describing a report appears laborious. However, it can become easier by use of picture data types and certain operations (definable in MODEL) which will specify automatically report standards, such as for instance including "end of record," "end of group" and "end of field" fields following the respective data description statements.

GRAPH REPRESENTATION AND COMPLETENESS OF A MODEL SPECIFICATION

Organization of precedence information

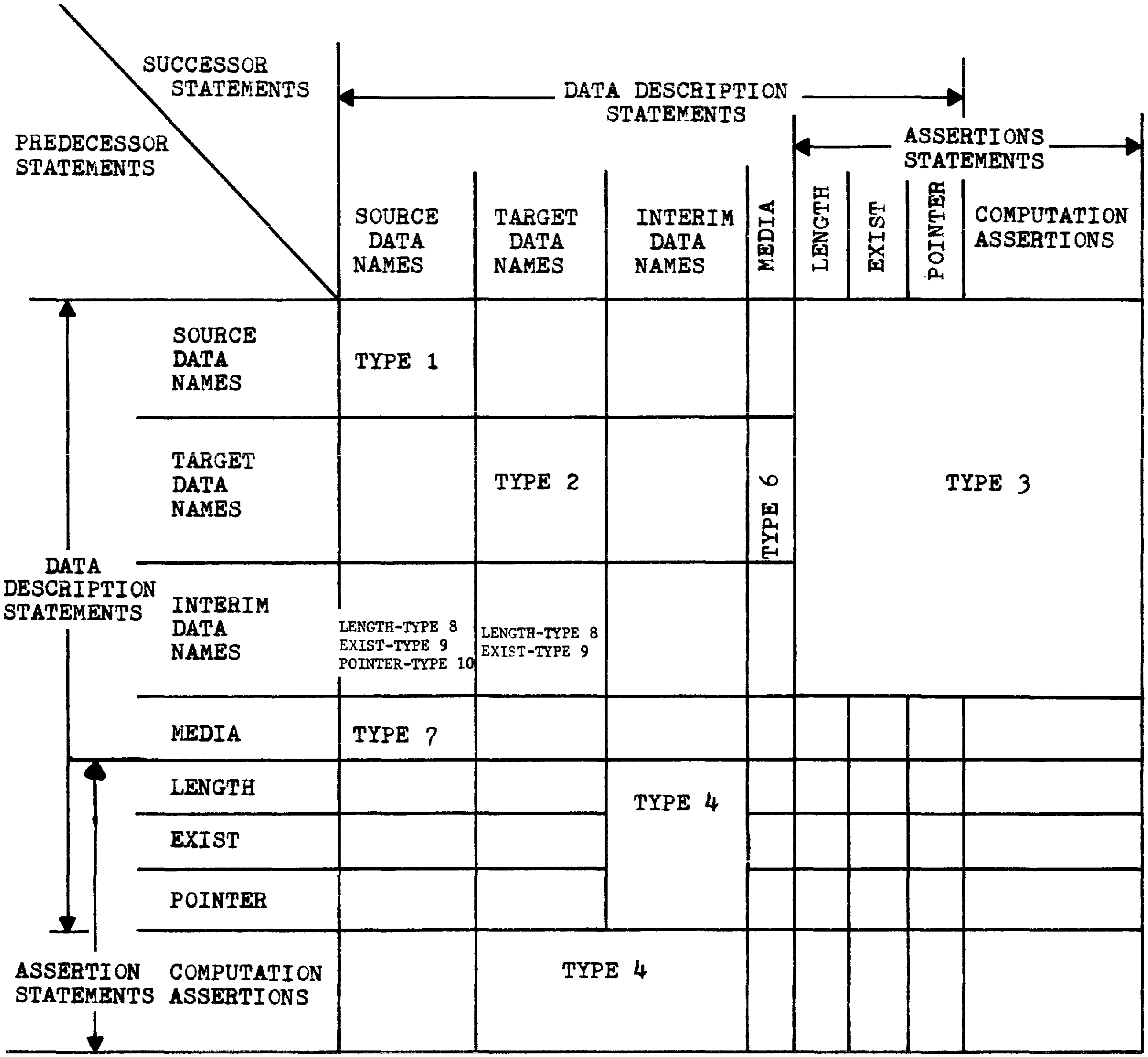
Each statement in MODEL is an integral unit identified by a name. The existence of a *precedence relationship* between two statements indicates that a statement must be evaluated prior to initiating the evaluation of its successor statement. The entire collection of statements is envisaged as a directed graph where the statements are represented by correspondingly labeled nodes and where directed arcs or *pointers* connect the nodes, each representing a precedence relationship between the statements at the pointer origin and termination nodes. Figure 10 illustrates this view, showing the statements (represented as ■) which form nodes of a graph for the above example. Each pointer is labeled, with the corresponding *precedence type*. Thus, each pointer has a direction and a type. The nodes in Figure 10 are shown to have a number of pointers *exits* and *entries*. The exits correspond to precedence pointers emanating from a node and pointing to *descendent* or target data or assertion nodes; the *entries* correspond to pointers originating at *parent* or source data or assertion nodes and terminating at a node.

The pointer finding process examines statements pairwise, using rules for determining the precedence type which will become the *type* of the precedence pointer. The types of the found pointers are entered in a *precedence matrix*, illustrated in Figure 11. Assume that a specification consists



85-10

Figure 10—Partial graph for depsale, showing data(.) of Figure 5 and assertions ( ) of Figures 7 and 9.



PREDECESSOR STATEMENTS	SUCCESSOR STATEMENTS	DATA DESCRIPTION STATEMENTS				ASSERTIONS STATEMENTS			
		SOURCE DATA NAMES	TARGET DATA NAMES	INTERIM DATA NAMES	MEDIA	LENGTH	EXIST	POINTER	COMPUTATION ASSERTIONS
DATA DESCRIPTION STATEMENTS	SOURCE DATA NAMES	TYPE 1				TYPE 3			
	TARGET DATA NAMES		TYPE 2		TYPE 6				
	INTERIM DATA NAMES	LENGTH-TYPE 8 EXIST-TYPE 9 POINTER-TYPE 10	LENGTH-TYPE 8 EXIST-TYPE 9						
	MEDIA	TYPE 7							
ASSERTION STATEMENTS	LENGTH			TYPE 4					
	EXIST								
	POINTER								
	COMPUTATION ASSERTIONS		TYPE 4						

Figure 11—Precedence matrix table, indicating use and existence of types of precedences

of n statements, then there are $n(n-1)$ pairs of statements that need to be considered for finding pointers.

The different precedence types indicate corresponding methods of interpreting the respective statements in the subsequent phase of code generation, not discussed in this report. The pointer type recognition rules are summarized below. These precedence types are extensible. Precedence types can be added provided that they can be stated in terms of pointer selection rules applied to statements, pairwise. The definition of pointer selection rules involves analysis of data and function names in predecessor and successor statements. Once the rules have been applied to pairs of statements and existence of pointers has been determined, these pointers are labeled with the appropriate precedence type. The labels of the respective pointers are then entered in the precedence matrix table, shown in Figure 11, at the intersection of a predecessor statement row and the successor column. Thus, once this process has been completed, a row will contain the types of all the exit pointers of a statement and a column will contain all the entry pointers types.

There are basically two main types of pointers:

(1) *Data Tree Hierarchy:* between data description statements (data names) within a FILE, organized in a tree structure. For source data the node closest to the apex is the predecessor and the node at the end of the branch is the successor, (precedence type 1) and vice versa for target data (precedence type 2).

(2) *Data Determinancy:* between assertions and data descriptions statements. When the data name is the source of an assertion, a data node is the predecessor and an assertion node is the successor (type 3). When a data name is the target of an assertion, an assertion node is the predecessor, and a data node is the successor (type 4).

Additionally, there are several miscellaneous precedence types requiring special interpretations in the code generation, as follows:

Media (storage) statements can have entries from source file statements (type 6) or entries from target files (type 7). LENGTH and EXIST type variables can have pointers to source (type 8) or target data (type 9) respectively. The POINTER type variables have pointers only to source data (type 10).

Figure 11 shows the Precedence Matrix with the rows and columns ordered by respective types of statements. The possible precedence types are indicated at the appropriate intersections. The ordering of statements in Figure 11 is not in fact required, but is purely for illustration and suggests useful reports to the user (as described below).

Analysis of precedence information

As indicated in Figure 1, a most important aspect of the concept of the MODEL system is the MODEL Processor's solicitation of new or modified statements. A number of reports are produced for review by the user. First, the Processor produces a listing of MODEL statements and a report which cross references each data name with the corresponding data and assertion statements. In addition, the Processor requests the user to resolve problems that it encounters. These requests have been divided into four classes: Incompletenesses, Ambiguities, Inconsistencies, and other Implications. Analysis of the information in the precedence matrix (Figure 11) can provide most of this information, as discussed below.

Incompleteness and Inconsistency problems are similar to "errors" and resolution of such problems is prerequisite for completion of the processing. They normally terminate processing. Ambiguity and Implications problems are similar to "warnings," and the Processor continues to complete the subsequent generation of code in the object language. The user may wish to examine these comments of the Processor, and, if necessary, make appropriate modifications, and resubmit them to the Processor. Otherwise these comments should be incorporated in the documentation of the program module.

The messages reporting results of the analysis which are sent to the user must be phrased in a manner that will make it easy for him to make modifications. The messages should therefore preferably address only one statement which needs to be added or modified. The exception to this rule would be where problems arise from Inconsistencies or Implications which are based on more than one statement.

Incompleteness is defined as an instance where the graph is incomplete, where entire statements are missing or where statements are duplicated. Such cases can be recognized by searching the precedence matrix of Figure 11 to verify the following conditions:

(a) Each row and column must have at least one pointer (in a column) and one exit pointer (in a row), with the following exceptions: Source and Target file statements have only an exit pointer or an entry pointer, respectively, while field statements may have no exit pointer (where the field is not used in deriving the target data). Absence of expected pointers, as above, indicates that a statement must be added by the user.

(b) The number of pointers in rows and columns must also be checked. Source data and target data statements can have only one pointer in the corresponding column and row, respectively. Also, Assertion statements can have only one pointer in the corresponding row. The existence of more pointers indicates either an Ambiguity which must be resolved by the user adding qualifying names to the names of similarily named data, or a logical Inconsistency due to duplicate statements.

(c) Each source data file statement which has an index sequential (ISAM) organization, must also have a POINTER type statement as a predecessor.

(d) Pointers in each row and column, should not originate or terminate, respectively, in statements having the same data or assertion name, otherwise an Ambiguity or an Inconsistency is indicated.

(e) The number of pointers in an assertion column must equal the number of source variables of the assertion.

(f) All pointers must conform with the allowable types indicated in Figure 11.

Before reporting the Incompletenesses, an attempt is made to resolve these problems automatically. If such resolution is possible, the suggested additions or modifications of statements are reported. If this process is not successful, appropriate messages with an indication of the missing or inconsistent statements are sent to the user. This supplementing of MODEL statements by the Processor is essential to relieve the user of providing much tiresome detail which may appear evident to the user. Rules for making such judgments may be added or modified based on experience with the Processor. Examples of such automatic additions and modifications to MODEL statements include:

(a) Modifying an assertion statement by preceding the names of an ambiguous data used in assertions with the names of their respective files (or other higher level data names). This would resolve the ambiguity where the same data name is used in a number of data statements.

(b) Naming of statements and identifying the SOURCE (independent) and TARGET (dependent) variables of assertions (where the user omitted this information).

(c) Providing assertions that will indicate equality of similarily named source and target data in the absence of other assertions expressing relationships between such data.

Inconsistencies are conflicts which require the user to conduct a logical analysis of more than one of the submitted statements. Some Inconsistencies are simple to determine. Examples were shown in the discussion of incompletenesses above, where a statement node has more than one exit or entry pointer, but only one is permissible. An Inconsistency message must then be produced which includes the offending statements. A more complex type of Inconsistency arises from the existence of "cycles" which are closed paths in the directed graph, each with a number of nodes and interconnecting pointers. The process for finding cycles is discussed in the references. Cycles in the directed graph denote faulty circular logic. They do not indicate iterations in the resulting program. Iterations in the program originate from a number of other features of the language such as the use of subscripts (FOR_EACH_X) following the name of a variable in an assertion and from the use of repeating data. It is required of the user to "open" the loops found by the processor through a modification of some of the statements corresponding to the nodes of the loops before the Processor can continue with the code generation.

Implications are classes of logical conclusions based on submitted statements that are considered to be potentially of interest to the user. The Processor, while capable of determining such conclusions, cannot further evaluate the implication, either because of limitation of the analysis methods that are employed, or because of lack of information of the area where the program is to be employed. Therefore the cooperation of the user is requested to check and verify the reported conclusions. Implication can be effectively reported in a form similar to "decision tables" which have been widely used in the past. Two such tables can be extracted from the matrix of Figure 11, consisting only of the data name rows and assertion columns where pointers of Type 3 exist and assertion rows and data columns where Type 4 pointers exist. Such tables are considerably smaller than the matrix of Figure 11 and therefore they can include the entire row and column statements.

CONCLUSION

The restrictions on space have limited the scope of this article to the extent that only a small illustrative example was presented with the objective of familiarizing the reader with the general use and operation of the system. The reader is referred to Reference 4 for a more comprehensive survey of the field of automatic generation of programs. References 1, 2, 3 and 4 provide detailed description of syntax and semantics of MODEL, and on the methods of generating programs automatically.

ACKNOWLEDGMENT

This paper is based on MODEL II. The implementation of this system was the responsibility of: S. Shastry and Y. Chang—Syntax analysis, N. A. Rin—precedence matrix and code generation, A. Pnueli—sequence and iteration analysis.

REFERENCES

1. "MODEL II—Automatic Program Generation, Description and User Manual," February 15, 1977 and "Automatic Generation of Computer Programs For Converting Transmitter Data To IRS Tape Standards—Phase I Report," December 15, 1976. Reports submitted to the Internal Revenue Service, Washington, D.C. 20024, Contract TIR-17-62.
2. Rin, N. Adam, "Automatic Generation of Business Data Processing Program's From A Non-Procedural Language," A dissertation in Computer and Information Sciences, University of Pennsylvania, Philadelphia, Pennsylvania 19174.
3. Prywes, N., "Automatic Generation of Computer Programs," Moore School Report #76-02, University of Pennsylvania, Philadelphia, Pennsylvania, 19174, September 1975.
4. Prywes, N., "Automatic Generation of Computer Programs," in *Advances in Computers,* Rubinoff and Yovits ed. Academic Press, in print.
5. Prywes, N., "Automatic Generation of Software Systems," *Data Base,* Summer 1974, pp. 7-17.

Sorting with associative secondary storage devices*

by C. S. LIN
University of Utah
Salt Lake City, Utah

ABSTRACT

A method for sorting large files stored on disks which possess an associative search capability is described. This method, called the *bucket sort algorithm*, uses a sort domain histogram to exploit the associative search capability. We discuss how to establish the sort domain histogram and analyze the performance of the bucket sort algorithm. Compared to the standard merge sort algorithm, this algorithm requires at most the processing time necessary for the initial run generation and the first pass of the merge operation. It also uses no disk storage space to store temporary results. The histogram creation process is analogous to Edelberg and Schissler's gyro sort algorithm which uses special hardware to rearrange data stored in electronic memory loops. The histogram creation process is more efficient than the gyro sort algorithm when each memory loop stores a large number of records and the distribution of sort domain values is not highly irregular.

INTRODUCTION

A recent trend in data base machine research is to design large scale associative memories using head-per-track disks, charge coupled devices and magnetic bubble memories. The design and application of such associative memories have been discussed by Parker,[9] Parhami,[8] Healy,[3] Su,[1,11] Ozkarahan,[7] Shuster,[10] Lin,[5,6] and Edelberg.[2] These associative memories have efficient search capabilities to support retrieval operations. We will study whether these associative memories can support another important data base processing operation—namely, sorting.

We suggested the idea of using an associative search capability to facilitate sorting operations in Reference 5. This idea leads to the so-called *bucket sort algorithm* which uses a *sort domain histogram* to exploit the associative search capability. In that paper, we assumed that the histogram was given. Now we will discuss how to dynamically create a sort domain histogram and then analyze the performance of the overall bucket sort algorithm. We will compare the performance of this algorithm with the standard merge sort algorithm.

The bucket sort algorithm is a software approach to using associative memories as sorting devices. Edelberg and Schissler[2] proposed a hardware intelligent memory which has both an associative search and a sorting capability. To perform the sort operation, the intelligent memory has a "Precession Control" circuit in addition to the circuit for performing the associative search operation. They developed a "Gyro Sort Algorithm" which uses the precession control circuit to sort large files stored on the intelligent memory. We will compare the efficiency of the bucket sort to the gyro sort.

THE BUCKET SORT ALGORITHM

The bucket sort algorithm is designed for sorting large files stored on disks which have an associative search capability. An associative head-per-track disk contains a separate processor module connected to each read/write head. As the disk rotates, these modules compare data passing under the heads with the search key and then transfer selected data to the output channel. If the set of selected records is small, all selected records can be transferred from disks to main memory in one disk revolution time. Otherwise, we can mark the selected records and take as many revolutions as needed to output marked records. The bucket sort algorithm exploits this associative search capability to faciliate sort operations.

Let us define the term *domain, sort domain, interval* and *bucket*. By a *domain* we mean a field in (or an attribute of) a record. Assume we want to sort the records of a file F according to their values in some domain J. We call J the *sort domain* with respect to this sort operation. An *interval* on domain K of some file is the set of K-values which fall between some given upper and lower bounds. When sorting a file F, a *bucket* is the set of records from F whose sort domain values are larger than the lower bound and smaller than or equal to the upper bound of a given interval. In general, buckets are not sorted internally.

Suppose we partition a file F into n buckets and the interval bounds when listed in ascending order are d_0,

* This research was partially supported by the National Science Foundation under grant MCS75-09903.

$d_1, \ldots, d_n$. Assume the CPU can sort each bucket in one revolution time. We can sort the file F in the following way. On the first revolution, we move records with sort domain values less than d_1 into the main memory and simultaneously mark all other records. On the i-th revolution we output marked records with sort domain value less than d_{i+1} and clear the marked records. As each bucket arrives we can sort it in main memory. Meanwhile, we can retrieve the next bucket into main memory. Thus, the file F is sorted bucket by bucket in an ascending order along the intervals of the sort domain. We continue in this way until the n-th revolution, when we simply output all marked records and clear their marks. The last bucket is sorted during the (n+1)-th revolution. If there is no output contention in any bucket retrieval, the file F can be sorted in n+1 revolutions.

Naturally, a problem with this bucket sort algorithm is determining the value of the interval bounds. Our solution is to maintain a *domain histogram* for every potential sort domain. The histogram contains the interval bounds and the size of each bucket. For static permanent files, the histogram can be collected and saved as a part of data statistics. For some attributes such as sex, color, etc., the number of possible sort domain values is usually much less than the number of records in the file. If we know the possible values for these sort domains, we might just as well sort the file by retrieval using the possible values as the key ordering. We don't need a histogram in such cases. However, in general, the domain histogram must be generated dynamically. We propose the following process to create a domain histogram.

HISTOGRAM CREATION PROCESS

For small files, we scan the file sequentially and read sort domain values of all records into main memory. Then we form a sorted list of sort domain values. From this list, we can easily determine the interval bounds. For large files, this method is not useful because it requires too much main memory. We can use the following method to determine the interval bounds for large files.

Suppose we want to partition a file into N buckets of B bytes.* If the sort domain values are uniformly distributed over the range (V_{min}, V_{max}), we can simply set the interval bounds at:

$$d_i = V_{min} + (V_{max} - V_{min}) * i/N \qquad (0 \leq i \leq N)$$

However, the sort domain values usually are not uniformly distributed. The above interval bounds may partition a file into N buckets of different sizes. Some buckets may be either too small or too big. To reduce such non-uniformity, we can partition each of the above N intervals into P intervals with the interval bounds set at:

$$d_j = V_{min} + (V_{max} - V_{min}) * j/(P*N) \qquad (0 \leq j \leq P*N)$$

If P is large enough (say, P=10), each of these P*N intervals should contain a bucket no more than B bytes (The average size is B/P bytes). Assume KNT(j) is the size of each bucket. We can combine several adjacent intervals into one if the sum of their KNT(j) are less than B bytes. Using this method we can determine interval bounds which form a nearly uniform partition of the file.

In most cases, the buckets generated by this process should be equal to or smaller than B bytes. Thus, with B bytes of working space in main memory, each bucket can be sorted internally. In case some buckets of the P*N intervals are larger than the available main memory, these buckets can be sorted by a standard merge sort algorithm. An alternative way to handle these oversized buckets is to partition them again using the same method described above. If we use a large value for P, oversized buckets should not be generated frequently. In the following discussion, we assume that all buckets are smaller than B bytes. Under this assumption, the average bucket size in the worst case is 0.5B bytes. This happens when the sum of every two buckets is slightly greater than B bytes. Thus our interval determination process may partition a file of N*B bytes into 2N buckets. However, there is only a very small probability of this worst case occurring.

PERFORMANCE OF THE BUCKET SORT ALGORITHM

In the bucket sort algorithm, the time required to sort a file is the sum of: (1) histogram creation time, and (2) bucket retrieval and sorting time. When setting up the histogram with P*N intervals, it requires little internal processing because we just update one of the KNT(j)'s as each record is scanned. In most cases, this operation should not be CPU bound. Therefore, if it requires R seconds to sequentially read the file at maximal data transmission rate, then the histogram with P*N intervals can be generated in R seconds. We can ignore the processing time required for interval bound determination process because it is usually much smaller than R seconds. Thus, the histogram creation time is approximately R seconds.

The processing time required for the bucket retrieval and sorting operations is dependent on the CPU's speed, bucket size and the algorithm used for sorting each bucket. As we want eventually to compare the bucket sort algorithm with the standard merge sort algorithm, we will use the replacement selection algorithm[4] to sort each bucket. The replacement selection algorithm is used in the merge sort algorithm to generate initial runs. It uses two input buffers, one output buffer and some working storage in main memory. We will assume the reader is familiar with the details of the replacement selection algorithm.

Suppose the working storage can store W records and the bucket size is smaller than W records. To begin the bucket sort operation, we retrieve the file bucket by bucket to fill up the working storage. Then we build, also in the working storage, a selection tree for the records in the working storage. Through the selection tree, we can select the

* We will discuss how to determine the bucket size in the next section.

record whose sort domain value is the smallest (or largest) among the records stored in the working storage in $O(\log_2 W)$ comparisons. While the remaining records are retrieved bucket by bucket, we perform sorting by repeating the following replacement selection process until the file is sorted. The process is to select, from the working storage, the record whose sort domain value is the smallest (or largest) and replace it with a record from the input buffer. With the bucket retrieval operation, the sort domain value of the selected record is always less (or greater) than the sort domain value of the replacement records. This allows us to sort the file, record by record, in ascending (or descending) order.

Suppose the file has FZ records and it takes S seconds to select and replace a record in working storage. In a CPU bound situation, the bucket retrieval and sorting operations take approximately FZ*S seconds.* To store the sorted file back to disk, it takes an additional R seconds if the read and write operation cannot be overlapped. Notice that the value S is proportional to $\log_2 W$. To reduce the value S, we can choose a small value for W. The optimal value of W, W_{opt}, is the largest bucket size that allows the replacement selection (bucket sort) operation and the bucket retrieval operation to be totally overlapped. If the bucket is smaller than W_{opt}, the replacement selection operation becomes I/O bound. Since our histogram creation process cannot uniformly partition a file into buckets of W_{opt} records, we should set W and the largest bucket size slightly larger than W_{opt} records so that the average bucket size is about W_{opt} records.

If the speed of the CPU is very high, the replacement selection operation could be I/O bound even when all available main memory is used as working storage. In such a case, the minimal processing time required for bucket retrieval and sorting operations is R seconds. As described before, it may take another R seconds to save the sorted file. This minimal bucket processing time can be achieved only if the bucket retrieval operations can transmit selected records from disk to main memory at full speed. Suppose the bandwidth of the disk is t bytes/revolution. The bucket size must be exactly t bytes or multiples of t bytes in order to achieve such an optimum. However, our histogram creation process usually cannot uniformly partition a file into buckets of t bytes or multiples of t bytes. Suppose the average bucket size is 2.4 t bytes. It requires three revolutions to retrieve each bucket. The average transmission rate is 0.8 t bytes/revolution in this case. The bucket retrieval operation will take (1/0.8)*R seconds. In general, it takes LF*R seconds to perform the bucket retrieval and sorting operations where LF is a "loss factor."

The range of LF is between 1 and 2. This range is based on the assumption that the size of the working storage is larger than t bytes. Such an assumption is valid in most systems. The worst case (LF=2) happens when the bucket size is slightly larger than t bytes. Because it takes two revolutions to read a bucket, the average transmission rate is only 0.5 t bytes/revolution. We use only half of the bandwidth in the worst case. Large bucket size will make LF close to 1.

Thus, the total processing time required for the bucket sort algorithm is:

$$R+\begin{cases} LF*R+R & 1\le LF\le 2 \quad \text{(I/O bound)} \\ FZ*S+R & \qquad\qquad \text{(CPU bound)} \end{cases} \tag{1}$$

COMPARISON OF THE BUCKET SORT AND THE MERGE SORT

To show the effectiveness of the bucket sort algorithm, we now compare its performance with the merge sort algorithm which is the standard method for sorting large files stored on disks. For the purpose of comparison, we assume that the file to be sorted by the merge sort algorithm is stored on head-per-track disks. Because the processing time listed in the equation (1) is based on the assumption that read and write operations on disks cannot be overlapped, we will impose the same limitation in the merge sort algorithm.

In the merge sort algorithm, the sort operation is performed in two steps: (1) generate initial runs, (2) merge the initial runs. The initial runs are sorted subfiles generated by the replacement selection algorithm. In the replacement selection operation of the merge sort, we read the file *sequentially*. Given a working space which can store W records, the replacement selection algorithm can produce FZ/2W initial runs* for a randomly ordered file of FZ records. Using an m-way merge, it takes $[\log_m FZ/2W]$ passes to merge these initial runs.

The initial run generation process is the counterpart of the bucket sort operation since they use the same algorithm. Thus, in a CPU bound situation, the initial run generation process takes about FZ*S+R seconds to perform. Notice that, in the merge sort, we usually want to keep the size of working storage, W, as large as possible to reduce the number of merge operations. This is different from the bucket sort algorithm in which we prefer a small W value to reduce the internal processing. As a result, the replacement selection operation in the merge sort may use a larger working storage and it takes more internal processing to select and replace a record in the selection tree. In other words, the S value in the merge sort is usually larger than the S value in the bucket sort. For simplicity of discussion, we assume the S values of the two algorithms are the same.

When the replacement selection operation is I/O bound, the initial run generation process can be completed in 2R seconds rather than LF*R+R seconds. This is because we read the file sequentially in the initial run generation process. We can transmit data from disk to main memory at full speed, i.e. LF=1, in this case.

The merge operation usually should not be CPU bound because it involves little internal processing compared to the replacement selection algorithm. To merge FZ/2W initial

* Assume the time required to establish the selection tree is negligible.

* Equation (3) of section 5.4.6 in Reference 4.

runs, it takes about $2R*[\log_m FZ/2W]$ seconds. Thus, the total processing time required for merge the sort algorithm is:

$$\begin{cases} FZ*S + R & \text{(CPU bound)} \\ 2R & \text{(I/O bound)} \end{cases} + 2R*[\log_m FZ/2W] \qquad (2)$$

To compare the performance of the bucket sort and the merge sort, we subtract (1) from (2). The result is:

$$\text{CPU bound:} \quad R*(2*[\log_m FZ/2W]-1) \qquad (3)$$

$$\text{I/O bound:} \quad R*(2*[\log_m FZ/2W]-LF) \qquad (4)$$

Because there is at least one pass of merge operation, the value of $[\log_m FZ/2W]$ is greater than or equal to 1. Since the value of LF is smaller than 2, both (3) and (4) are always greater than 0. This shows that the bucket sort algorithm is more efficient than the merge sort algorithm. The worst case occurs when LF=2 and $[\log_m FZ/2W]=1$, because the value of $[\log_m FZ/2W]$ indicates how many passes of merge operations are required in the merge sort. The above comparison shows that, in the worst case, the bucket sort algorithm requires the processing time necessary for initial runs and the first pass of merge operations of the merge sort algorithm. For small files, the bucket sort algorithm is slightly faster than the merge sort algorithm. As (3) and (4) show that the difference in the performance of these two sort algorithms is proportional to $[\log_m FZ/2*W]$, the bucket sort algorithm is even more favorable for sorting large files.

There are other advantages of using the bucket sort algorithm to sort files stored on associative disks:

(1) *No temporary space required on the disk*—The bucket sort algorithm requires only some working storage space in main memory. While in the merge sort, we need a temporary storage space on disks to store the initial runs and the temporary results of merge operations. The size of this temporary space is equal to the size of the file to be sorted.

(2) *Shorter response time*—In the bucket sort, we can provide the first portion of the sorted file is available immediately after the first bucket is sorted. While in the merge sort, we must wait until the last pass in the merge operation to get such a result.

(3) *Less cost to interrupt a sort operation*—Sorting is always a time consuming operation. In a multiprogramming system, sometimes it is necessary to interrupt a sort operation to favor a short job. When using a merge sort, we must save all temporary results, whose size is as large as the file itself, unless the interrupt occurs during the last pass of the merge operation. Frequently, this temporary disk space must be released to perform other jobs. That means we have to start all over again later. In the bucket sort, the sorted buckets can be consumed by other processes immediately after they are generated. In this case the only temporary result we have to save is the histogram. Because the size of the histogram is much smaller than the file, it will be cheaper to save.

COMPARISON OF THE BUCKET SORT AND THE GYRO SORT

Electronic cyclic memories such as charge-coupled devices and magnetic bubble memories can be used to implement large scale associative memories. We can consider these memories as "electronic head-per-track disks." Such electronic disks have an important characteristic not available in mechanical disks: that is the data circulation rate in each memory loop (equivalent to a track) can be varied or even dropped to zero under program control. Such a characteristic could be used to design a sorting capability.

Edelberg and Schissler[2] designed an intelligent memory which has both an associative search and a sorting capability. The processor module of each memory loop has a "precession control" circuit to control data circulation within that loop. For each loop, there is a sub-loop which holds a record from that loop. Associated with every two sub-loops of adjacent loops, there is a processor module which can exchange the records in the two sub-loops according to their sort domain values. Thus, each processor module can sort a pair of records stored in two adjacent sub-loops. With n processor modules performing sort operations in parallel, we can sort n records stored in n adjacent sub-loops in (n+1)/2 sub-loop rotation times.

To sort a large file stored on intelligent memory, Edelberg and Schissler designed a *gyro sort algorithm* which uses the precession control circuit and the sorting capability in the sub-loops to rearrange the records. They consider each file stored on the intelligent memory as an array. Suppose a file is stored in L *adjacent* loops (rows) and each loop contains K records. The gyro sort algorithm sorts this L×K array by performing the following process K times:

(1) Sort columns—Move each column into the sub-loops and sort it with the sorting capability described above.

(2) Precess all rows—Rotate the elements in the i-th row by (i−1)mod K positions ($1 \leq i \leq K$).

This algorithm rearranges the elements in the array in such a way that the array is sorted across the rows but not within each row. We can consider each memory loop now contains a bucket. To sort elements in each row, i.e., each memory loop, we must rely on other means, such as sorting each row in the main memory.

Since the gyro sort algorithm is used for forming a bucket in each memory loop, it is analogous to our histogram creation process in the bucket sort algorithm. However, the histogram creation process does not physically form buckets in the memory. With the associative search capability, the sort domain histogram is sufficient for making bucket retrieval operations. For a file of L×K records, it takes approximately K*L/2 memory loop circulation times (equivalent to disk revolution times) to perform a gyro sort. Suppose we pack 100 records (K=100) in each memory loop, it will take 50L loop circulation times to perform a gyro sort. On the other hand, our histogram creation process requires only L loop circulation times (assuming it is not a CPU bound operation) regardless of how many records are contained into each loop.

As far as the operation of sorting each row and the operation of bucket retrieval and bucket sort are concerned, there is little difference in performance since both operations must be performed by the CPU. Thus, when each memory loop contains a large number of records, the bucket algorithm is more efficient even though the gyro sort uses more hardware.

The main advantage of the gyro sort algorithm is that its performance is independent of the distribution of sort domain values. When sort domain values are concentrated within certain ranges, the histogram creation process may have to scan the file several times to determine the proper interval bounds. Such cases usually occur when the number of possible sort domain values is small. For example, it occurs for sort domains such as color and sex. As we mentioned earlier, the file may as well be sorted by retrieval using the possible values as the key ordering in these cases.

When the number of records per loop is small, the gyro sort algorithm could be more efficient. For example, if K=4, it takes only 2L loop circulation times to perform a gyro sort. Since each loop contains only 4 records, it takes very little internal processing to sort each row of the array. In the bucket sort, we should not set the bucket size as small as 4 records even if the optimal bucket size is 4 records. This is because it would require too much main memory space to store the corresponding histogram. It is necessary to trade the processing efficiency for main memory space requirement in this case. A large bucket size will increase the time for internal processing and may make the bucket sort less efficient in such a case. However, the cost-per-bit of memories with a small number of records per loop is high. To reduce the cost, usually it is necessary to pack a large number of records into each loop in large scale associative memories. Thus, we can conclude that the gyro sort is useful for small scale, high speed associative memories. The bucket sort algorithm is suitable for large scale, lower speed associative memories.

SUMMARY

We have described an algorithm for sorting files stored on memories which have an associative search capability. This algorithm, called the bucket sort algorithm, consists of two operations: (1) establish a histogram in main memory, and (2) use the histogram to perform bucket retrieval and sorting operations. Compared to the standard merge sort algorithm, the bucket sort algorithm requires at most the processing time necessary for initial runs and the first pass of the merge operation, the bucket sort algorithm is also more efficient than the gyro sort algorithm if each memory loop stores a large number of records and the distribution of sort domain values is not highly irregular.

ACKNOWLEDGMENT

I am grateful to Professor John M. Smith for his interest, suggestions and guidance in conducting this research.

REFERENCES

1. Copeland, G. P., G. J. Lipovski, and S. Y. W. Su, "The Architecture of CASSM: A Cellular System for Non-Numeric Processing," *Proc. of the First Annual Symposium on Computer Architecture*, December 1973, pp. 121-128.
2. Edelberg, M. and L. R. Schissler, "Intelligent Memory," *AFIPS Conference Proc.*, Vol. 45, 1976, pp. 393-400.
3. Healy, L. D., K. L. Doty, and G. J. Lipovski, "The Architecture of a Context Addressed Segment Sequential Storage," *AFIPS Conference Proc.* Part I, Vol. 41, 1972, pp. 691-701.
4. Knuth, D., "The Art of Computer Programming- Sorting and Searching," Vol. 3, Addison Wesley, 1973
5. Lin, C. S., D. C. P. Smith, and J. M. Smith, "The Design of a Rotating Associative Array Memory for Relational Data Base Applications," *ACM Transactions on Database Systems*, March, 1976.
6. Lin, C. S., "The Design of a Rotating Associative Relational Store," M.S. Thesis, University of Utah, June 1976.
7. Ozkarahan, E. A., S. A. Schuster, and K. C. P. Smith, "RAP-An Associative Processor for Data Base Management," *AFIPS Conference Proc.*, Vol. 44, 1975, pp. 379-387.
8. Parhami, B., "A Highly Parallel Computer System for Information Retrieval," *AFIPS Conference Proc.*, Vol. 41, Part I, 1972, pp. 229-241.
9. Parker, J. L., "A Logic per Track Retrieval System," *Proc. IFIP Congress*, 1971, TA4-146-TA4-150.
10. Schuster, S. A., E. A. Ozkarahan, and K. C. Smith, "A Virtual Memory System for a Relational Associative Processor," *AFIPS Conference Proc.*, Vol. 45, 1976, pp. 855-862.
11. Su, S. Y. W. and G. J. Lipovski, "CASSM: A Cellular System for Very Large Data Bases," *Proc. of Very Large Data Base Conference*, September 1975, pp. 456-472.

A specialized architecture for textual information retrieval*

by L. A. HOLLAAR
University of Illinois at Urbana-Champaign
Urbana, Illinois

and

W. H. STELLHORN
U. S. Army Construction Engineering Research Laboratory
Champaign, Illinois

ABSTRACT

Retrieval of information from the complete text of large document collections cannot be performed efficiently or rapidly by current general purpose digital computers or by most special purpose rotating memory associative processors frequently proposed for efficient processing of relational databases. Characteristics which distinguish text retrieval from retrieval of formatted files are discussed, and a computer configuration employing for special purpose processors is described.

INTRODUCTION

A major use of digital computers is to manage, correlate, and retrieve large collections of data, either in the form of formatted files or text with minimal formatting. Most database information retrieval systems are concerned with the former, which was formalized in a paper on relational ~~databases by Codd.[1] In this formalization, each element of~~ the database can be regarded as an ordered n-tuple, with one or more of the fields forming a unique key field.

For example, assume that the database is used for an inventory control system. The fields in each element are the inventory number (which, because it is unique for all elements, is the primary key), a description of the item, the supplier, the date purchased, and the amount paid. Queries on the database can be for all items of a certain description (single key exact match), all items purchased between two dates (single key range), all items purchased between two dates from a certain supplier (multiple key match), or a number of other forms.

In general, the operations performed on a relational database consist of matching, either exactly or within a specified range, one of the key fields; marking elements based on the success or failure of a match operation and selecting from among these marked elements using Boolean operators; arithmetic functions performed on fields within the elements; and the addition and deletion of elements from the database.

The nature of the database allows elements to be stored in any order without affecting the result of a query (although in a particular implementation they may be sorted on one of the key fields to improve the retrieval speed). The data may be encoded to minimize the storage requirements or improve the efficiency of the matching by allowing the use of either ranges or bit masks. Large databases of this type would generally contain 10 to 100 million characters.

TEXTUAL DATABASES

The second form of data organization is that used by textual information retrieval systems. The files consist of ~~collections of documents, with some scheme to delimit and~~ access the individual documents within the file. While formatted databases are concerned with fields and key values of known position and format, textual databases primarily operate on contexts (sentences, paragraphs, documents, etc.) and retrieval keys consisting of arbitrarily chosen words or portions of words. The contents of textual databases are order dependent and allow little encoding of the data. Very little formatting is necessary, although for efficiency special flags may replace normal typography to indicate the start or end of a context (such as replacing a number of leading blanks on a line with a mark to indicate the start of a paragraph or replacing a period with an end of sentence mark so that decimal points are not misinterpreted).

The normal operations on a textual database consist of forming progressively smaller subsets of the database until the number of documents is small enough to be examined by the user. This is done by the specification of search

* This work was supported in part by the National Science Foundation under Grant US NSF-MCS73-07980 A03.

patterns consisting of co-occurrences, alternatives, and exclusions. Searches for co-occurrences locate two or more terms within a specified context ("FIND 'BEOWULF' AND 'GRENDEL' IN SENTENCE"). The co-occurrences can be either unordered, as in the example given above, or ordered. The specification of an ordered co-occurrence can either require that the terms be contiguous or be separated by not more than a specified number of words. Searches for alternatives locate contents which contain at least one of a group of selected terms. The alternative terms can be specified explicitly, generated from a thesaurus based on a key term, or produced automatically by either suffixing or prefixing. In the latter case, the user specifies that any words either starting or ending with a specified string are to be used in the match. Exclusion searches locate contexts which contains one term but not another.

Many textual databases contain a large number of documents and grow fairly rapidly. One suggested database consists of 200 million characters, with documents being deleted as new documents are added to keep the files from exceeding this amount.[2] A file containing the statutes of all states would take approximately 1 billion characters, while one containing all court decisions (which are of more interest to practicing lawyers) would take around 25 billion characters. This corresponds to 250 disk spindles, using 3330-type technology.

As storage technology improves, the storage of these large databases becomes feasible. Furthermore, as more typesetting is done using computers, the effort necessary to produce the database is substantially reduced—in many cases the same tape used as input to the photocomposition program can be reformatted to form the database. However, while the production and storage of large textual databases is possible, the rapid retrieval of data from these databases using conventional digital computers is not practical. For example, if it takes 1 microsecond to examine each character (which is quite fast), it would take about 7 hours to completely scan 25 billion characters! Of course most searches have no relevance to a given query, so that much of the time is spent on unnecessary scanning.

INVERTED FILES

The common approach to searching such collections rapidly is to use inverted files, i.e., to generate an index consisting of each important word or phrase and a list of all the documents in the database in which the word is found.[3] Figure 1 illustrates one possible scheme for implementing an inverted file structure. Although it is convenient to view this structure as having two parts, the inversion and the document files, increased efficiency can be obtained if the inversion file is divided into two separate files, index and postings. The index file contains each unique term along with a pointer to the postings and a count of the number of entries in the postings list. The postings file contains the actual lists of documents in which the term is contained. The use of two levels simplifies the searching of the index required by prefix and suffix operations, and provides information to optimize the order of merging the lists for searches which involve multiple terms.

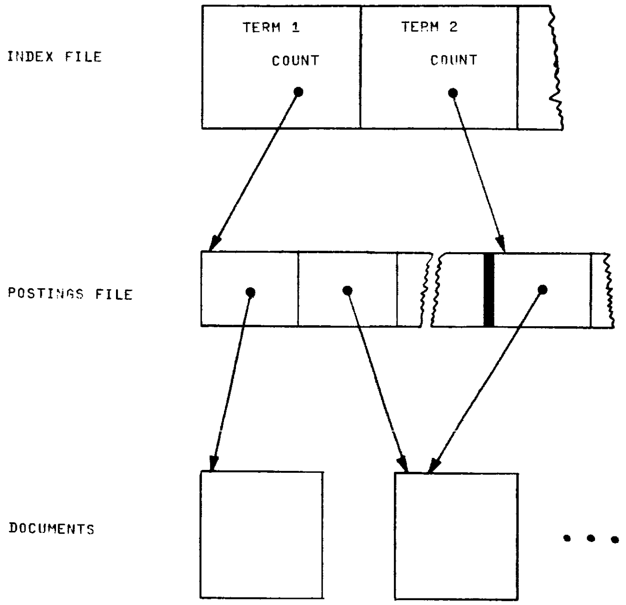


Figure 1—Inverted file structure

The inverted file structure allows the user to combine (using operators such as AND, OR, and AND NOT) lists of entries until a small subset of the database is formed, which can then be searched. However, this approach increases the amount of storage required to hold the database by 7 to 120 percent, depending on the level of the inversion.[4] If the file is inverted to the document level, very little overhead is required and the time necessary to merge lists will be small, but a large number of documents will require full text searching if the query asks for the co-occurrence of two terms in the same sentence or paragraph. If inversion is to the word level, no full text searching will be required, but the inverted file will probably be larger than the original data file and the time required to merge the lists will be high.

When the programs necessary for full text searching or list merging are examined, it becomes apparent that these tasks do not lend themselves nicely to conventional digital computers, which were originally designed for numeric operations. Less than 10 percent of the time within the inner loops are spent fetching or storing the actual data—the rest of the time is spent fetching instructions, controlling program flow, or aligning data so that it can be operated upon. It is clear that if extremely large textual databases are to be utilized, some form of specialized computer architecture must replace the general purpose computer.

A number of papers[5-10] have suggested using a rotating memory with logic on each head for the parallel execution of non-numeric functions. The memory can be either a disk or drum (generally with a head per track) or some form of shift register or delay line (bubble memory, CCD, etc.), which circulates the data in a serial fashion. As the data passes by the head of the rotating memory, logic associated with each head examines the data to check it against a specified key. If a proper match is found, a method is used to mark the data so that it can be referenced by later operations. Properly configured and programmed, the operation of one of these logic per track processors is identical to that of an associative memory.[11]

However, the operations performed by these rotating memory associative processors are tailored for formatted databases, such as relational databases. They do not perform efficiently, if at all, on the context searches required by a textual database information retrieval system. In addition, most are not capable of operating on data which spans more than one track, starting on one and continuing on one or more additional tracks. This is a particular problem on ordered co-occurence searches where the success of the pattern match cannot be determined by a single processor.

A SPECIALIZED ARCHITECTURE

The system to be described is based on performance evaluations and simulations done as part of the EUREKA project at the University of Illinois,[12] discussions with operators of large scale text retrieval systems, and projections of system loading as the size of data and index files are expanded. Many portions of it have been simulated and some have been constructed as prototypes. Figure 2 illustrates the general configuration of the system, with the flow of information indicated by the arrows. The user interfaces directly with a front end processor, which reformats his requests and passes them to a resource scheduling processor. The resource scheduling processor determines the proper hardware subsystems necessary to service the request, and sequences their operation. This configuration would be used on an extremely large system, with smaller systems having some of the separate processor subsystems consolidated. For example, the front end processor and the resource scheduling processor may actually be tasks implemented on the same general purpose computer.

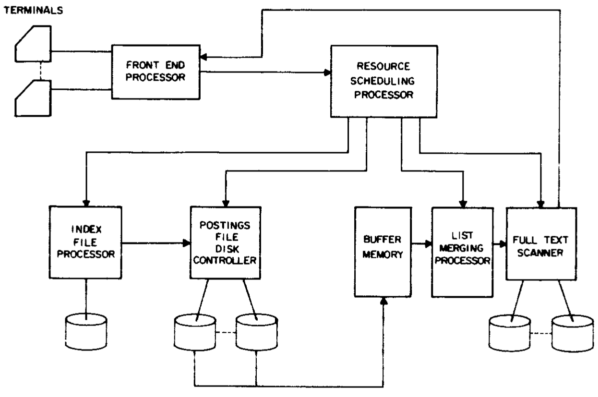


Figure 2—System configuration

FRONT END PROCESSOR

This is a general purpose digital computer, possibly a minicomputer, which is the primary interface between the user at his terminal and the rest of the system. It is responsible for the control of the terminals and their associated lines and modems, as well as other devices such as high speed line printers. Commands given by the user are parsed by this processor, and diagnotics are given to the user if any errors are detected. System output is formatted based on the user's requests. In addition, a number of utility tasks such as accounting, interactive HELP facilities, etc., are handled by this processor.

RESOURCE SCHEDULING PROCESSOR

This subsystem, again probably implemented on a minicomputer, controls the other hardware subsystems the way an operating system would control a variety of software tasks. Its primary job is to queue requests entered from the front end processor or produced as a result of the operation of one of the hardware subsystems, and to dispatch them in a manner which maximizes the system thruput. It tries to reduce both the positioning and rotational latency of the various disk systems by reordering the requests in the queue. The user specified requests are reordered to reduce the length of the intermediate results (and therefore the disk transfers) during postings list merges according to an optimizing strategy.[13] Finally, it monitors the operation of the various hardware subsystems, providing diagnostic control and error recovery if a problem occurs.

INDEX FILE PROCESSOR

The first level of the inverted file search is handled by this subsystem. It finds the terms on the disk, and forwards a list of pointers to the postings file disk controller to be retrieved. The usage counts are returned to the resource scheduling processor so the merge operation can be optimized. Ordinarily, this operation could be handled by the resource scheduling processor, since it involves only a simple disk lookup on a sorted file. However, the use of prefixes and suffixes complicates the matter. In this case, all entries which either end or start (or both) with a given substring need to be retrieved from the disk. This is not difficult for suffixing (except that the user may unintentionally request a large number of entries), since the file is sorted and all entries which match the beginning substring are located in one group. However, the processing of prefixes is not as simple. In this case, only the ending of a word is known, while the file is organized by word beginnings. The conventional solution is to search the file sequentially for all terms which match the specified substring.

If system performance necessitates faster operation than would be possible with conventional searching by a minicomputer, an associative processor similar to those discussed above can be utilized. Unlike the document file and its operations, the index file is ideally suited for an associative processor, since it consists of a series of 3-tuples. However, most of the logic per head units proposed would have to be redesigned to handle a problem which occurs in pattern matching. Consider searching for the substring ISSIPPI in MISSISSIPPI. In normal operations, the character matching circuits would skip the initial M, match the following ISSI, and then expect to match a P. Instead, the match unit encounters an S, and determines that the match operation failed. If it then tries to restart the matching, without backtracking to the second I of the word (most systems are incapable of this backtrack operation), it will not find the desired substring in the word, which is clearly an error. While this may seem a contrived situation, experience dictates that such situations do occur.

Therefore, the search processor must be capable of backtracking or must provide some alternative mechanism to handle this situation. Backtracking can be achieved by the use of a special buffer equal in length to the largest substring allowed in the implementation, and logic to recognize the possibility that backtracking may yield a productive result.

POSTINGS FILE DISK CONTROLLER

This can be simply a standard disk memory system controlled by the resource scheduling processor based on the results from the index file processor. Data, in the form of document postings lists are transferred from the disk to the buffer memory used by the list merging processor. However, because of the high bandwidth capabilities of the list merging processor, it may be necessary to increase the data transfer capacity substantially. Prototypes have been constructed for a modification to a 2314- or 3330-type moving arm disk to allow the reading of a number of the tracks (generally 8 or 16) in parallel. The remaining tracks can be used for format and control purposes. This requires the construction of special deskew and format control logic to compensate for magnetic data migration or misalignment of the heads, especially if the disk was written on a drive other than the one used for reading.

LIST MERGING PROCESSOR

Conventional digital computers cannot efficiently merge two or more sorted files—typically fewer than 10 percent of the memory cycles in the inner loop of the merge operation are devoted to fetching input list items or storing the results of the merge.[14] The remaining cycles are used for the fetching of instructions, alignment of data, and control of program execution. While this is no problem if the lists are not long or if time is not critical, on a large scale inverted file information retrieval system, it can become a substantial bottleneck. It is not unusual to find that up to two-thirds of the time on a large CPU is devoted to reading and merging postings lists.

Because the operations necessary to merge two or more lists are both simple and well defined, this is an ideal area for special purpose hardware to replace general purpose computers. In its simplest form, the required processor consists of a comparator, a data selector, logic to access a memory, and a special sequencer to control these components. Data is fetched from memory and placed in one of two holding registers, corresponding to the two input lists. The values are compared and the lower of the two is transferred to an output register. If the desired operation was to form the union of the two lists ("OR"), the output register's contents are transferred to memory; if the intersection ("AND") was specified, the transfer occurs if the two input values are equal. In either case, the register which held the lower value (or both if they were equal) is loaded with the next list entry from memory.

If fields other than the document number are present, such as a count to indicate how often a term occurs in the document or context flags indicating a term's location, additional logic can be added to produce the desired results in parallel with the normal merge processing.

The starting memory addresses and lengths for the input lists and the operation desired are furnished by the resource scheduling processor. After the appropriate registers have been loaded, the merge processor is started, and operates autonomously until it is finished or an error occurs. When this happens, the resource scheduling processor is interrupted and a new operation is started.

Since the list merging processor does not have the overhead of fetching and sequencing instructions, and can be constructed to eliminate the data alignment problem, speed increases of 10 to 20 over a general purpose processor are possible. However, if a substantial increase beyond this is necessary, some form of parallelism must be introduced. This can be done either by handling many list entries at the same time or by merging more than two lists during an operation.

Figure 3 shows the hardware configuration for a merge

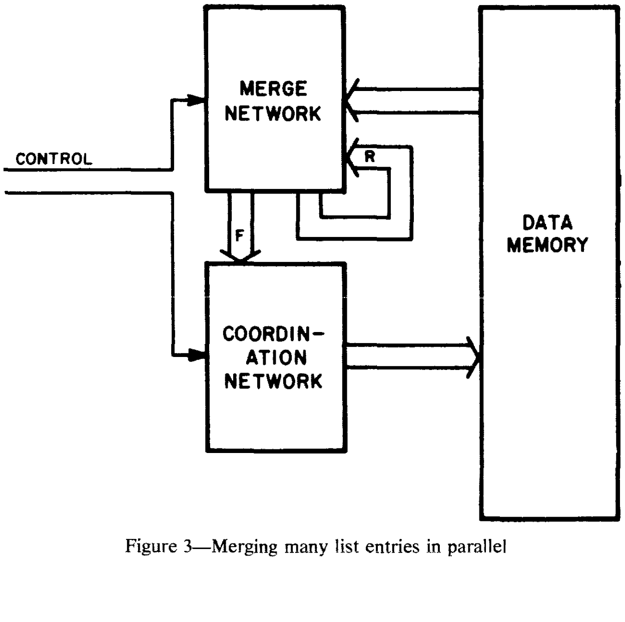


Figure 3—Merging many list entries in parallel

processor capable of combining more than one element from a list at a time.[15] It is based on a Batcher-type even-odd merge network[16] and a term coordination network capable of removing unwanted results. The merge network is capable of combining two sorted n-item lists into a single 2n-item sorted list. Processing is bit serial, with the most significant bit first, and operates in a pipelined fashion. The coordination network removes duplicate entries for OR operations and duplicate and single entries for an AND.

Lists of more than n items can be accommodated by modifying the connections so that only the first half of the merge network results are sent to the coordination network, while the second half are returned to the merge network. Since only one input to the merge network remains, the blocks from the two input lists must be interspersed. Selection is based on the value of the first item within a block, with the list having the lower value selected for input to the merge network on the next cycle. At the beginning of processing, the feedback path is initialized to a zero condition, while at the end a final block of data consisting of "infinite" values is sent to the merge network.

If a number of simple merge processors are connected together to form a network, then more than two lists can be combined at the same time.[17] Since each simple merge processor, or element, has two inputs, the network takes the form of a binary tree (Figure 4). A special memory controller distributes input data from memory to the network, where it flows through in a pipelined fashion. Unlike the previously discussed unit, where a postprocessor was necessary to remove unwanted items, this network removes them as they are being processed. The network operation is synchronized to the buffer memory, with an item either stored or fetched on each major network cycle.

Programming of the network consists of supplying the memory addresses for the input and output lists, and the operation to be performed by each element. If the desired expression does not need the entire network, a PASS operation can be programmed to logically remove an element from the network. Loading can be done either directly from the resource scheduling processor to the individual elements (using an addressable configuration register for each element) or by sending command information through the network with a special flag. The latter is more practical for LSI implementations, as it reduces the number of pins required for each element.

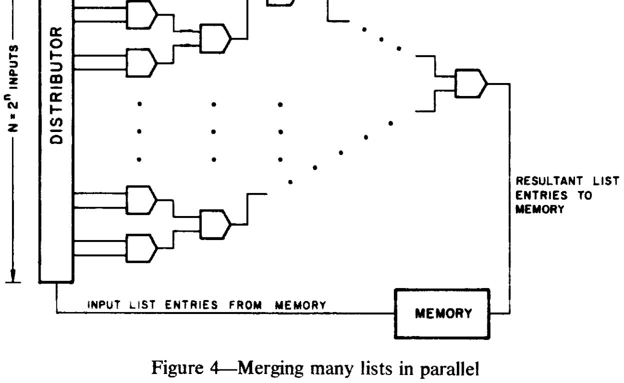

NUMBER OF ELEMENTS = $\frac{N}{2}+\frac{N}{4}+\cdots+1 = N-1$
LOG$_2$ N STAGES
N = 2^n INPUTS
DISTRIBUTOR
RESULTANT LIST ENTRIES TO MEMORY
INPUT LIST ENTRIES FROM MEMORY
MEMORY

Figure 4—Merging many lists in parallel

Because all intermediate results are contained in the network and processed immediately by the next stage, the memory cycles previously used to store and later refetch intermediate results are unnecessary. This can result in a substantial increase in the effective bandwidth of the buffer memory. For example, if the OR of 256 lists (not uncommon due to prefixing, suffixing, and thesauruses) is desired, up to 88 percent of the memory cycles previously required are unnecessary. If operations are limited by the transfer of data between the buffer memory and slower disk files because of space limitations, this savings can produce substantial speed increases.

FULL TEXT SCANNER

The full text scanner is a logic per head rotating memory processor, similar to the rotating memory associative processors and the index file processor previously discussed.[18] As was shown earlier, a standard rotating memory associative processor is unable to efficiently perform the necessary operations on data whose only formatting is context delimiters replacing standard typography.

The scanner is sequentially positioned at each document specified by the final result of the inverted file merge. For each search, the comparison substrings or terms are specified, and the desired context delimiters loaded into start and stop registers. An analysis circuit is configured based on the specified operators and ordering to determine if a search has succeeded in a particular context. A number of independent units process multiple tracks in parallel, with scanning of data beginning independently in each processor whenever a start of context flag is encountered. Whenever an end of context flag is sensed, the results of the comparisons are passed to the analysis unit to determine if the specified pattern has been matched. If so, it is added to a list of matching contexts for later reference by the user.

If the pattern is too large to be contained entirely within the logic unit, it can be divided into a number of smaller patterns and the results of these pattern matches merged similarly to postings lists. A more difficult problem exists if the context spans more than one track. In this case, which can be easily recognized by the lack of a context start or end flag, data must be transmitted between the logic units for two or more tracks to determine if the pattern has been matched. Since the track logic which is scanning the end of the context will complete its operation before the one scanning the start, when the end of context flag is sensed, the pertinent data is passed forward to the unit scanning the start. When the unit scanning the start reaches the end of the track, it can combine the information from its scan with the data passed to it to determine if the pattern was matched.

When all the scanning has been completed, the resultant list is returned to the user through the resource scheduling and front end processors. Since he will wish to display some of the documents identified, commands can be sent by the resource scheduling processor to the full text scanner to transfer a specified context to the front end processor for display on the user's terminal.

SUMMARY

This paper has described the characteristics of a textual database and its information retrieval system, and indicated that it has fundamental differences from a relational database. It was noted that the storage technology exists, or will shortly, to accommodate extremely large collections of documents, some on the order of 25 billion characters. However, current general purpose digital computers are unable to efficiently and rapidly process quantities of text of this magnitude. Special purpose processors, generally rotating memory associative processors, suggested for efficient processing of relational databases also are unable to efficiently handle textual data.

The architecture for an efficient inverted file information system was presented, with specialized processors handling the simple but repetitious tasks of merging lists of document pointers or scanning text for patterns.

REFERENCES

1. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," *Comm. ACM*, June 1970.
2. Roberts, D. C., "A Specialized Computer Architecture for High-Speed Text Searching," presented at the Second Workshop on Computer Architecture for Non-Numeric Processing, January 1976.
3. Roberts, D. C., "Survey of File Organizations," in *Advances in Computers 12,* M. Rubinoff, ed., Academic Press, 1972.
4. Rinewalt, J. R., "Evaluation of Selected Features of the EUREKA Full-Text Information Retrieval System," Report 823, Department of Computer Science, University of Illinois, September 1976.
5. Slotnick, D. L., "Logic per Track Devices," in *Advances in Computers 10,* Franz Alt, ed., Academic Press, 1970.
6. Parker, J. L., "A Logic per Track Retrieval System," *Proc. IFIP Congress,* 1971.
7. Parhami, B., "A Highly Parallel Computer System for Information Retrieval," *Proc. AFIPS FJCC,* 1972.
8. Healy, L. D., K. L. Doty and G. J. Lipovski, "The Architecture of a Content Addressed Segment Sequential Storage," *Proc. AFIPS FJCC,* 1972.
9. Ozkarahan, E. A., S. A. Schuster and K. C. Smith, "RAP—An Associative Processor for Data Base Management," *Proc. AFIPS NCC,* 1975.
10. Lin, C. S., D. C. P. Smith and J. M. Smith, "The Design of a Rotating Associative Memory for Relational Database Applications," *ACM Trans. on Database Systems,* March 1976.
11. Davis, E. W., "STARAN Parallel Processor System Software," *Proc. AFIPS NCC,* 1974.
12. Hollaar, L. A., et al., "The Design of System Architectures for Information Retrieval," *Proc. ACM National Conf.,* 1976.
13. Liu, J. W. S., "Algorithms for Parsing Search Queries in Inverted File Document Retrieval System," Report 742, Department of Computer Science, University of Illinois, April 1975.
14. Hollaar, L. A., "A List Merging Processor for Information Retrieval Systems," presented at the First Workshop on Computer Architecture for Non-Numeric Processing, October 1974.
15. Stellhorn, W. H., "A Specialized Computer for Information Retrieval," Report 637, Department of Computer Science, University of Illinois, October 1974.
16. Batcher, K. E., "Sorting Networks at their Applications," *Proc. AFIPS SJCC,* 1968.
17. Hollaar, L. A., "A List Merging Processor for Inverted File Information Retrieval Systems," Report 762, Department of Computer Science, University of Illinois, October 1975.
18. Stellhorn, W. H., "A Processor for Direct Scanning of Text," presented at the First Workshop on Computer Architecture for Non-Numeric Processing, October 1974.

Fault-tolerant modularized arithmetic logic units

by T. R. N. RAO
Southern Methodist University
Dallas, Texas

and

H. J. REINHEIMER
IBM Corporation
Gaithersburg, Maryland

ABSTRACT

Use of both parities and residue checks, called a "combination code", can provide a cost-effective error detection and correction and modularized design of arithmetic and logic units (ALU). As codes they compare favorably with the byte-error correcting codes of Neumann and Rao in their information rate, and indeed are better suited for a modularized (or byte-sliced) design of ALU.

INTRODUCTION

Error correcting codes and techniques are becoming increasingly important in the design of digital systems for improving reliability and maintainability and providing fault-tolerance. Even in very early machines, such as, IBM 7030,[1] coding techniques have been employed. The error ~~correcting can be broadly divided into two classes, namely,~~ parity-based codes and residue codes.[2] A very commonly used parity code in digital systems is the single-error correcting and double-error detecting (SEC-DED) Hamming code.[3,4] Parity codes are proven to be very efficient and cost-effective for memory and data transfer operations. The residue codes, on the other hand, appear very attractive for checking arithmetic operations, such as, ADD, Complement, cycle, etc.[5-10]

In some computer systems, such as IBM stretch (7030), both parity and residue techniques have been employed to check for errors in different logic blocks of the system.[1] These techniques, however, have not been used together to complement each other to provide improved error detection or correction capability. It is the specific purpose of the paper to combine the two coding techniques into one called "combination code" which can be applied to provide efficient error correction for all operations of an arithmetic logic unit (ALU).

Residue codes

A class of residue codes which finds a low-cost[5-8] application to arithmetic operations is the $[N, |N|_b]$ code. In this code, the information N is an integer and $|N|_b$ is the residue check on N modulo, b (b is called the check base) $|N|_b$ is the non-negative remainder obtained by dividing N by b. Another class of residue codes called "AN codes" has been extensively studied[4,11,13] for correction of errors in arithmetic operations. In this class, the information N is represented as the product AN for a suitable constant A (A here is called the generator and sometimes the check base of the code).

In an AN code if $A=(2^c-1)(2^k-1)$ and gcd(c, k)=1, the code is shown to be capable of correcting any type of error in one byte of the operand.[15] Here the byte length is c bits and the operand word length is n=kc bits (or k bytes). This code can be implemented as a biresidue code[7] with two checkers (check bases 2^c-1 and 2^k-1) for arithmetic operands of length kc bits.

~~Although the checkers are of the low-cost form[5,6] and are~~ well suited for implementation, they differ considerably from the byte processor modules of the ALU. The information rate (IR) of these codes is given by

$$IR=kc/(kc+k+c).$$

For a 36-bit and 56-bit arithmetic logic units, the information rates are 36/(36+9+4)=.735 and 56/(56+7+8)=.789 respectively. The somewhat lower information rates of these codes compared with some other byte-error correcting codes (of Reference 15) can certainly be justified by the low-cost implementation they provide. However, for some n, such as n=32 or 64, we cannot find k and c which are relatively prime and kc=N.

As a contrast to the above codes, we consider "combination codes" i.e., codes with both parity and residue checks which alleviates the difficulty (i.e., k and c need not be relatively prime) and are also advantageous for practical implementation as will be shown later.

Combination codes

The concept of combination codes has a basis on the earlier results of England[16] Langdon and Tang[17] and others[5-10] on the use of parities and residue checks for error correction. Their results can be put together to obtain an effective error correction for byte-organized processors. A combination code (or mixed code) is a block code (see Figure 1) of length n, where the information is followed by a residue check and a group of parity bits as shown. For information X, the corresponding code word is $[X, P, |X|_b]$ where b is the check base preferably of the form 2^c-1 and P is a group of parities calculated over bytes of X.

England[16] presented an example of a combination code as follows (see Figure 1). The information X is at most 7 bytes each byte of 3 bits. Three parity check bits P, and a residue check $|X|_7$ of three bits are computed over X as follows: Let $X=(x_{21}, x_{20}, \ldots, x_1)$ be divided into bytes $B_7, B_6, \ldots, B_0$. Then the parity check portion $P=P_3P_2P_1$ are computed from various bytes of X as below:

$$\begin{aligned} P_1 &= \sum{}^{\oplus} B_1, B_3, B_5, B_7 \\ &= \sum{}^{\oplus} X_1, X_2, X_3, X_7, X_8, X_9, X_{13}, \\ &\quad X_{14}, X_{15}, X_{19}, X_{20}, X_{21} \\ P_2 &= \sum{}^{\oplus} B_2, B_3, B_6, B_7 \\ &= \sum{}^{\oplus} X_4, X_5, X_6, X_7, X_8, X_9, X_{16}, \\ &\quad X_{17}, X_{18}, X_{19}, X_{20}, X_{21} \\ P_3 &= \sum{}^{\oplus} B_4, B_5, B_6, B_7 \\ &= \sum{}^{\oplus} X_{10}, X_{11}, X_{12}, X_{13}, X_{14}, X_{15}, \\ &\quad X_{16}, X_{17}, X_{18}, X_{19}, X_{20}, X_{21} \end{aligned} \tag{1}$$

The residue check, $|X|_7$, is obtained by adding all the information bytes $B_1, B_2, \ldots, B_7$ using modulo 7 adders. (See, for instance, Reference 13 pages 70-71.)

England did not provide a formal proof of the error correction capability of the code but he illustrated by examples the single error detection and correction capability of the code as follows. Assume the erroneous word is X′ and the erroneous bit is x_{17}'. Also assume that the correct value of $x_{17}=0$, erroneous value $x_{17}'=1$). Then by computing the parity check equations over the erroneous word, we get the parity syndrome S_p (or check number) of 110 pointing to an error in B_6 of X′. To obtain the actual error position we compute the residue syndrome

$$S_r = |X'-X|_7 = \Big|\, |X'|_7 - |X|_7 \,\Big|_7.$$

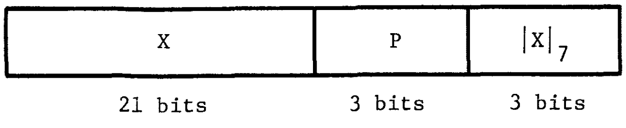


Figure 1—The combination codeword

Clearly, the above will yield the error magnitude of $|2^{17-1}|_7=2$. This determines the error position as the second bit in the byte pointed by the parity syndrome. On the other hand, if $x_{17}=1$, $x_{17}'=0$ (1→0 type error) the residue check yields

$$|X'-X|_7 = |-2^{16}|_7 = 5 = |-2|_7.$$

This once again points to the second bit but to 1→0 type error.

It is to be noted that there is a unique parity syndrome for each one of the seven bytes and a unique residue syndrome for each bit and type error in the byte. Encoding and decoding is very much like a single error correcting Hamming code except that all three bits of a byte are combined into one. Also to be noted is that when the error is in the information part, both parity check and residue check detect this error and together they enable error correction. On the other hand, if error is in P or in $|X|_7$ only one of the two syndromes is non-zero. Therefore, we derive the following error location strategy (2) and the error correction strategy of (3).

$$\begin{aligned} S_p=0,\ S_r=0 &: \text{ No error} \\ S_p=0,\ S_r\neq 0 &: \text{ error in } |X|_7 \\ S_p\neq 0,\ S_r=0 &: \text{ error in P} \\ S_p\neq 0,\ S_r\neq 0 &: \text{ error in X} \end{aligned} \tag{2}$$

$S_p=i\neq 0$, $S_r=j\neq 0 \Rightarrow B_i$ is in error $(i=1, 2, \ldots, 7)$

$$\begin{aligned} j=1 &\quad \text{first bit, } 0\to 1 \text{ type error} \\ j=6 &\quad \text{first bit, } 1\to 0 \text{ type error} \\ j=2 &\quad \text{second bit, } 0\to 1 \text{ type error} \\ j=5 &\quad \text{second bit, } 1\to 0 \text{ type error} \\ j=4 &\quad \text{third bit, } 0\to 1 \text{ type error} \\ j=3 &\quad \text{third bit, } 1\to 0 \text{ type error} \end{aligned} \tag{3}$$

From the above, any single bit error (1→0 or 0→1) in X can be detected and corrected. Whenever only one of S_p and S_r is non-zero, the parity or residue check can be recomputed from X and therefore it is established that any multiple errors in P or $|X|_7$ can be detected and corrected.

COMBINATION CODE FOR ARITHMETIC LOGIC UNITS

The Combination Codes of the preceding section could be applied for arithmetic logic units (ALU's) but the implementation logic would be extremely complex and the cost of implementation prohibitive. Therefore we resort to one important modification which would make the code attractive from implementation point of view. One important modification is to replace the parity byte P, by the parity bits, one parity bit for each of the k information bytes of X. It is fairly simple to note that this modification does not alter the error detection/correction properties of the code. However, by this modification, there will be more parity bits (k bits instead of log k) and lower information rate for the code, but the implementation would be rather straightforward. We could also integrate the parity and associated logic with the information byte and the byte processor logic. The modified combination code is illustrated in Figure 2.

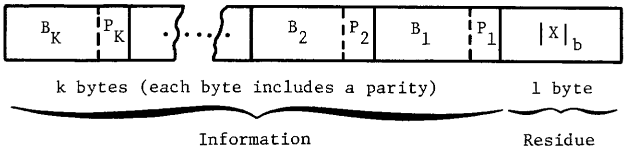


Figure 2—Modified combination codeword

We refer to this "modified combination code" as simply combination code hereafter. We illustrate below a schematic for its implementation. For an operand X of n bits (n=kc) we have a corresponding combination codeword [X, P_X, $|X|_b$] where P_X is k parity bits (one parity bit for each byte of X) and $|X|_b$ is the residue of X modulo b. ($b=2^c-1$).

First let us consider a simple model of an ALU as given in Figure 3:

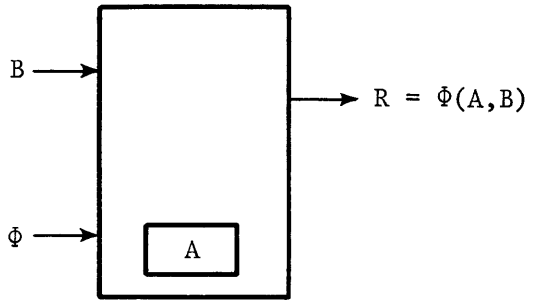


Figure 3—A simple model of an ALU

A is an internal operand, B an input (external) operand and ϕ is the op code, all specified at time t. We denote the output R as the result of the operation ϕ on A and B and is made available at a unit time later. For this model, therefore, we have

$$R(t+1)=\phi(A(t), B(t)) \tag{4}$$

To apply combination code to the above model we resort to a slightly modified ALU (see Figure 4) along with a "residue unit" and a "decoder unit." The operands A and B of Figure 3 appear as codewords [A, P_A, $|A|_b$] and [B, P_B, $|B|_b$] in Figure 4. The operation ϕ in Figure 3 is replaced in Figure 4 as combined operation (ϕ, ϕ_b). For each operation ϕ on A and B, parity logic is designed in ALU so that (R, P_R) are generated as outputs. P_R consists of k parity bits called "predicted parity" bits. Also for each ϕ, a corresponding parallel operation ϕ_b is designed for the residue unit to operate on the residues $|A|_b$ and $|B|_b$. The corresponding result is denoted by

$$R_b=\phi(|A|_b, |B|_b) \tag{5}$$

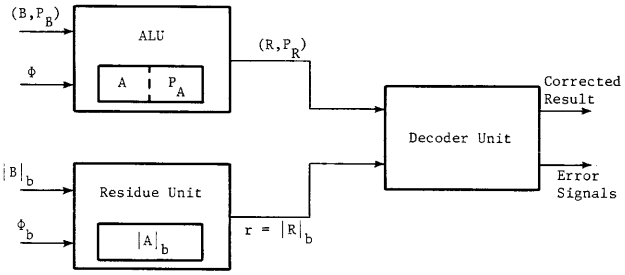


Figure 4—A combinational code schematic for ALU operations

By this approach we generate a combined output [R, P_R, $|R|_b$], a codeword, if there are no errors. However, due to logic faults there could be errors in this combined result. The decoder unit then generates the syndromes (S_p, S_r) and utilizes them to locate and correct those errors. When the errors detected by the decoder unit are not correctable appropriate error signals are generated and maintenance/repair is requested.

The above discussion is a general schematic of the combination code application to ALU's. A number of details are omitted in order to illustrate the concept. In the next sections, we present more details on the parity logic used in ALU modules (the byte processors) and the residue and decoder units.

CHECKING ARITHMETIC AND LOGICAL OPERATIONS

Before we describe the details of implementation of combination codes for ALU operations, we need to derive the parity logic and residue logic equations. Therefore we present in the next section, the parity logic equations for checking ALU operations. For illustration and simplicity, we assume 4-bit ALU modules. In a later section, residue logic equations are derived and tabulated.

Parity checking schemes for arithmetic and logical operations

Sellers et al.[12] described a number of schematics of parity checking for arithmetic and logical operations. Based on their work, Langdon and Tang[16] made an important contribution by evaluating the effectiveness and cost of parity-checked, group carry look-ahead adders. Langdon and Tang compared parity checking scheme with residue (modulo 3) check scheme. They took into account a number of practical considerations such as group carry look-ahead, group length, the fault-coverage and logic circuit complexities (gate counts) etc. Their conclusions point out that parity checking of adders is very effective from both cost and fault-coverage points of view. Reinheimer,[9] in a patent disclosures describe detailed logic diagrams and design for parity and residue checking combination to provide error detection and correction of ALU operations, and his work forms a basis for this paper. Garcia and Rao[14] discussed using multiple parities (or a Hamming distance 3 code[3] to detect and correct single errors in logical operations AND, OR, and EXOR).

Here we approach the problem of parity-checking of a simple 4-bit ALU operation. We treat this subject by modeling the ALU as a generalized sequential machine and deriving the check equations.

The result of an ALU operation ϕ on the operands A and B yields a result $R=\phi$ (A, B) which appears not only on an output but replaces the contents of Register A and serves as one of the two operands for the next operation that follows. For example, if ϕ is ADD operation then $R=|A+B|_m$ where m is 2^4 for 2's complement addition (and $m=2^4-1$ for 1's complement case). Similarly ϕ is defined for all operations of the ALU.

Checking the ALU with parity

Full-Sum Check

The full sum check equation is given by:

$$e_1=P_S\oplus P_A\oplus P_B\oplus P_C\overset{?}{=}0 \qquad (6)$$

Here P_A and P_B are the parity check digits of the input operands while P_C denotes the *parity* of the carries *within the adder*. Equation (6) will be referred to as the full-sum parity check equation. The adder here, is a 4-bit Group Look-ahead Adder that is implemented from the following equations given for the first stage module:

$$C_{-1}=C_{in}$$

$$C_0=G_0+T_0C_{in}$$

$$C_1=G_1+T_1G_0+T_1T_0C_{in}$$

$$\ldots \qquad (7)$$

$$C_3=\text{carry into next group}$$

$$=G_3+T_3T_2T_1T_0C_{in}$$

$$S_i=a_i\oplus b_i\oplus C_{i-1}$$

where:

A and B represent the two operands and S stands for the resultant sum; a_i and b_i are the ith stage inputs, s_i the ith stage sum digit and C_{i-1} is the carry from the previous stage.

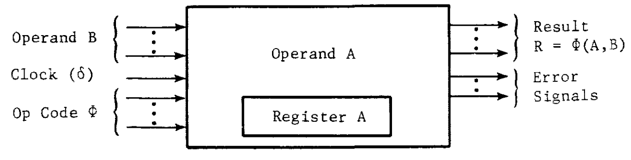


Figure 5—Model for a self-checked ALU module

H_i is the half-sum function for the ith stage $H_i=a_i\oplus b_i$
G_i is the generate function for the ith stage $G_i=a_ib_i$
T_i is the transmit function for the ith stage $T_i=a_i+b_i$

It can be shown that

$$P_C=P_G\oplus P_{TC}\oplus Pc_{in} \qquad (8)$$

where

$$P_G=G_0\oplus G_1\oplus G_2,$$

$$P_{TC}=\overline{G}_0H_1\overline{H}_2+G_1H_2 \qquad (9)$$

$$PC_{in}=C_{in}(\overline{T}_0+T_1\overline{T}_2)$$

Implementing P_C with equation (8) does not require that the carry circuitry be duplicated. Also, the P_C can be formed before the sum is formed, thus supporting the check to be made without degradation of adder performance.

Half-Sum Check

The half-sum check equation is given by:

$$e_2=P_A\oplus P_B\oplus H_0\oplus H_1\oplus H_2\oplus H_3\overset{?}{=}0 \qquad (10)$$

This is implemented to check that there is no error in P_A, P_B, G_i or T_i since each H_i is formed as G_i+T_i.

Carry From One Stage to the Next—Check

To check that the Group-Carry from one stage to the next is not in error it is necessary to duplicate the logic to generate the carry out and compare it with the Carry being sent. Thus, the equation for the check for the Group Look-ahead Carry out from the first group is:

$$e_3=C_3\oplus C_{3dup} \qquad (11)$$

The error signal, E_s, for the ALU module is generated by (12) as follows:

$$E_s=e_1+e_2+e_3. \qquad (12)$$

One of the C_3 is generated independently from the rest. The operation of this Adder as an ALU is as follows:

> For all ops, the ALU operates as an adder and the above adder checks are done for each op. This assures that the hardware is checked for every op against single-failures.

The selected op, ADD, AND, CYL will determine the result which is gated to the outputs. For example: For ADD, S is put onto the outbus; *if* AND, the G is outputted; a CYL is done by making both inputs to the adder the same and generating a Hot carry-in if there is a carry from the high order position.

Residue checking of ALU operations

The residue code, namely, [N, $|N|_b$] code is considered here. For illustration, we select N to be of 16 bits and the

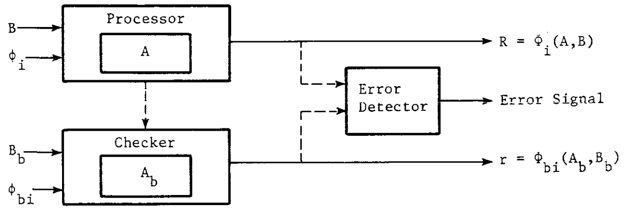


Figure 6—Generalized model for separate processor and checker

ALU uses '1's complement addition. The check base, b=15 is employed. We illustrate a simple model of the ALU and its residue checker as in Figure 6.

Thus, if N represents the integer value of the accumulator contents at any instant, the check $|N|_b$ is stored in a residue register, such that the contents of the accumulator, together with the contents of the residue register form a codeword at the end of an operation cycle.

Let N_1 denote the operand stored in the accumulator at the beginning of an operation cycle. Let N_2 denote the integer value of the input on the parallel data lines from memory. An operation denoted by ϕ_i can be defined as a function of either N_1, N_2 or both N_1 and N_2. Thus, ϕ_i can be a unary or a binary function. The result of an operation is denoted by R, and is given by

$$R=\phi_i\,(N_1, N_2). \qquad (13)$$

The result R replaces the contents of the accumulator by the end of the operation cycle, and is also generated as output.

Let N_{1b} denote the residue of N_1 modulo the check base b. We assume that N_{1b} is maintained in the residue register. Let N_{2b} denote the residue of N_2 modulo b. Our aim is to find an operator ϕ_b, such that $\phi_b\,(N_{1b}, N_{2b})=r$, and r is the residue of R modulo b, thus preserving the relation of congruence modulo b between the contents of the accumulator and of the residue register. In symbolic language. we wish to preserve the relation $|\phi(N_1 . N_2)|_b=\phi_b(N_{1b} . N_{2b})$. If we can find such a ϕ_b for every ϕ, error checking is simply reduced to checking the relation of congruence between the accumulator contents and the check register contents periodically. Any violation of this congruence can be set up as an error signal.

In the following discussion we consider the accumulator to be of length n=16. In order to have simplified checking logic, 1's complement arithmetic will be used (which is equivalent to considering all operations modulo 2^n-1), and b will be of the form 2^c-1, where c divides n.

Error Coverage: The mod b residue check detects all failures for which the difference between the correct answer and the incorrect answer is not a multiple of b. If b is odd, all single errors of the form $\pm 2^j$ are detected. This is because for all j, $|\pm 2^j|_b \neq 0$. All bursts of length less than c (where $2^c-1=b$) and most bursts of length c or greater are detected.

Checking arithmetic operations

The checking of arithmetic operations has been discussed.[13] For each operation that is carried out in the processor, there is a parallel operation that goes on in the residue unit, such that at the end of the operation cycle, the checker contains the residue of the accumulator contents modulo b (if there has been no error in the computation).

The first part of Table I is a listing of the formulas for the results of various processor arithmetic and arithmetic type operations, and their corresponding checker operations. These are easily derived (See Reference 13).

Checking logical operations

In the checking of logical operations, we immediately run into a problem. The residue code is closed under operations such as ADD, SUB, CYCLE, etc., but is not closed under logical operations such as AND, OR, and exclusive-OR. However, some simple techniques can be employed to enable the residue unit to generate the predicted residue for all logical operations with a little additional hardware. One such technique is due to Monteiro and Rao.[8] The residue equations for logical operations appear at the bottom of Table I.

MODULARIZED FAULT-TOLERANT ALU

In previous sections, we covered separately the parity checking and residue checking schemes for arithmetic and logical operations. Here we present a modularized fault-tolerant ALU organization which uses a combination of parity and residue checks. First, by fault-tolerance we

TABLE I—Characterization of Arithmetic and Logical Operations of a Processor

Operation ϕ_i	Result $R=\phi_i(N_1, N_2)$ $m=2^n-1$	Operation ϕ_{bi}	$Q=\phi_{bi}(N_{1b}, N_{2b})$ $b=2^k-1$
	A. ARITHMETIC OPERATIONS		
ϕ_1(ADD)	$\|N_1+N_2\|_m$	ϕ_{c1}	$\|N_{1b}+N_{2b}\|_b$
ϕ_2(SUB)	$\|N_1-N_2\|_m$	ϕ_{c2}	$\|N_{1b}-N_{2b}\|_b$
ϕ_3(SM)	$\|N_2-N_1\|_m$	ϕ_{c3}	$\|N_{2b}-N_{1b}\|_b$
ϕ_4(COMP)	$\|-N_1\|_m$	ϕ_{c4}	$\|-N_{1b}\|_m$
ϕ_5(SHL)	$\|2N_1-A_{n-1}(t)\|_m$	ϕ_{c5}	$\|2N_{1b}-A_{n-1}(t)\|_b$
ϕ_6(SHR)	$\|{}^1/_2(N_1-A_0(t))\|_m$	ϕ_{c6}	$\|{}^1/_2(N_{1b}-A_0(t))\|_b$
ϕ_7(CYR)	$\|{}^1/_2 N_1\|_m$	ϕ_{c7}	$\|{}^1/_2 N_{1b}\|_b$
ϕ_8(CYL)	$\|2N_1\|_m$	ϕ_{c8}	$\|2N_{1b}\|_b$
ϕ_9(CLA)	N_2	ϕ_{c9}	N_{2b}
ϕ_{10}(CLZ)	0	ϕ_{c10}	0
ϕ_{11}(SET)	0	ϕ_{c11}	0
	B. LOGICAL OPERATORS		
ϕ_{12}(AND)	$\|\delta_l(A(t)\cdot B(t))\|_m$	ϕ_{12b}	$\|\delta_l(A(t)\cdot B(t))\|_b$
ϕ_{13}(OR)	$\|\delta_l(A(t)\vee B(t))\|_m$	ϕ_{13b}	$\|N_{1b}+N_{2b}-\|\delta_l(A(t)\cdot B(t))\|_b\|_b$
ϕ_{14}(EXOR)	$\|\delta_l(A(t)\oplus B(t))\|_m$	ϕ_{14b}	$\|N_{1b}+N_{2b}-2\|\delta_l(A(t)\cdot B(t))\|_b\|_b$

mean specifically here that the proposed ALU is not only capable of detecting all errors due to single logic faults*, but also correcting those errors, that is, the ALU is self-correcting for all single faults.

This organization is a further development, and an extension of the one discussed by Reinheimer.[9] We start with an ALU module which is self-checking for all single faults. It is parity-checked for all its operations and all errors due to single faults are detected. When errors are detected, they are indicated by two output, error signals, namely, the carry-out error signal and the sum error signal. As stated earlier, at most one of the error signals will be a 1, since the ALU and parity-checked logic are so designed that for any single logic fault only carry-out (i.e., group carry-out) is in error or the sum is in error but never both.

We consider the fault-tolerant ALU as a whole is of n-bits where n=kc and therefore we organize it as k ALU modules and each module is of c bits. For illustrations we use n=16 and we use four 4-bit ALU modules (k=4, c=4). The residue check base, b, used for the combination code is given by

$$b=2^c-1.$$

Therefore in our illustrations below we use the residue check base b=15.

The organization of a 16-bit ALU is illustrated in Figure 7. All ALU modules shown are exactly identical to the model shown in Figure 6. The inputs to each module are 4-bit data, OP code and timing control. The outputs correspondingly are a 4-bit sum a carry-out to the next module (with an end-around carry from module 3 to module 0; that is $c_{15}=c_{-1}$) and error signals, namely, carry-out error, and sum error (see Figure 7).

The outputs from the ALU modules and the residue checker are applied to the combination code decoder unit called here Error Decoder. The Error Decoder has two units, namely, the Fault Location Unit and Error Corrector Unit whose functions are discussed below.

Residue unit (RU)

The residue checker logic functions have been described for all arithmetic and logical operation earlier. Here we outline briefly the inputs, outputs and the logic blocks of the residue checker. The data register (16 lines) the op code and timing control are the inputs. These inputs are used to generate and maintain the residue modulo 15 of the results of the 16-bit ALU. This part of the logic are called residue predictor block in Reference 8. The RC also receives the 16-bit result (the sum S from ALU) which is used to generate the residue of S modulo 15 independently. This block is called residue calculator. Finally, error magnitude calculator block generates the difference between the actual (calculated) residue and the predicted residue. This difference is the error magnitude. Whenever the error magnitude so calculated is non-zero, then a residue error signal (RES=1) is generated.

Fault locator unit

The fault locator unit receives the error signals from the four modules and the residue checker. In all there will be 9 bits, 2 each from the ALU modules and 1 from the Residue Checker. Fault location unit then generates signals to the Error Corrector unit whenever an error is correctable. It also generates two other signals, namely, no faults signal and detected but uncorrectable errors signal as the case may be.

The fault location is based on the following cases.

(1) If RES=0, all $E_S=0$ ⇒ no error
(2) If RES=0, any $E_S=1$ ⇒ Fault in parity logic uncorrectable error
(3) If RES=1, more than one $E_S=1$ ⇒ Uncorrectable multiple faults
(4) If RES=1, all $E_S=0$ ⇒ Residue checker is faulty
(5) If RES=1, exactly one $E_S=1$ ⇒ Single fault correctable error

Error corrector unit

The error correction unit can be implemented in any number of ways. The correctable errors are assumed only when exactly one E_S equals 1 (and the rest equal 0) and the RES=1. A sum error of a faulty ALU module is corrected by subtracting the error magnitude from the sum output of that module. Other byte outputs will not be affected. On the other hand if a carry-out error is to be corrected, an appropriate number of bytes starting with the next stage may have to be corrected. The error magnitude in the latter case will be ±1. The details of the error corrector implementation are not considered here.

Additional considerations

The correction algorithm depends on the fact that there are carry faults possible that yield one kind of Result Error and Sum faults that yield another kind of Result Error. It is believed that details of the particular correction algorithm can be studied and generalizations made which will help a designer know how to design other devices where errors due to single faults are correctable and errors due to multiple faults are detectable through use of the parity/residue combination code.

The organization of the 16-bit ALU, here, makes use of the self-checking modularized 4-bit ALU module. These modules support other types of fault-tolerant organizations and it is not the purpose in this report to limit the application of these modules. For instance; one could use

* A "logic fault" or hereafter "fault" is a node assuming a different truth value from its correct value. Generally, a logic fault is classified as stuck-at-1, stuck-at-0, or inverted. A single fault could produce multiple errors in a circuit but it still is called "single fault".

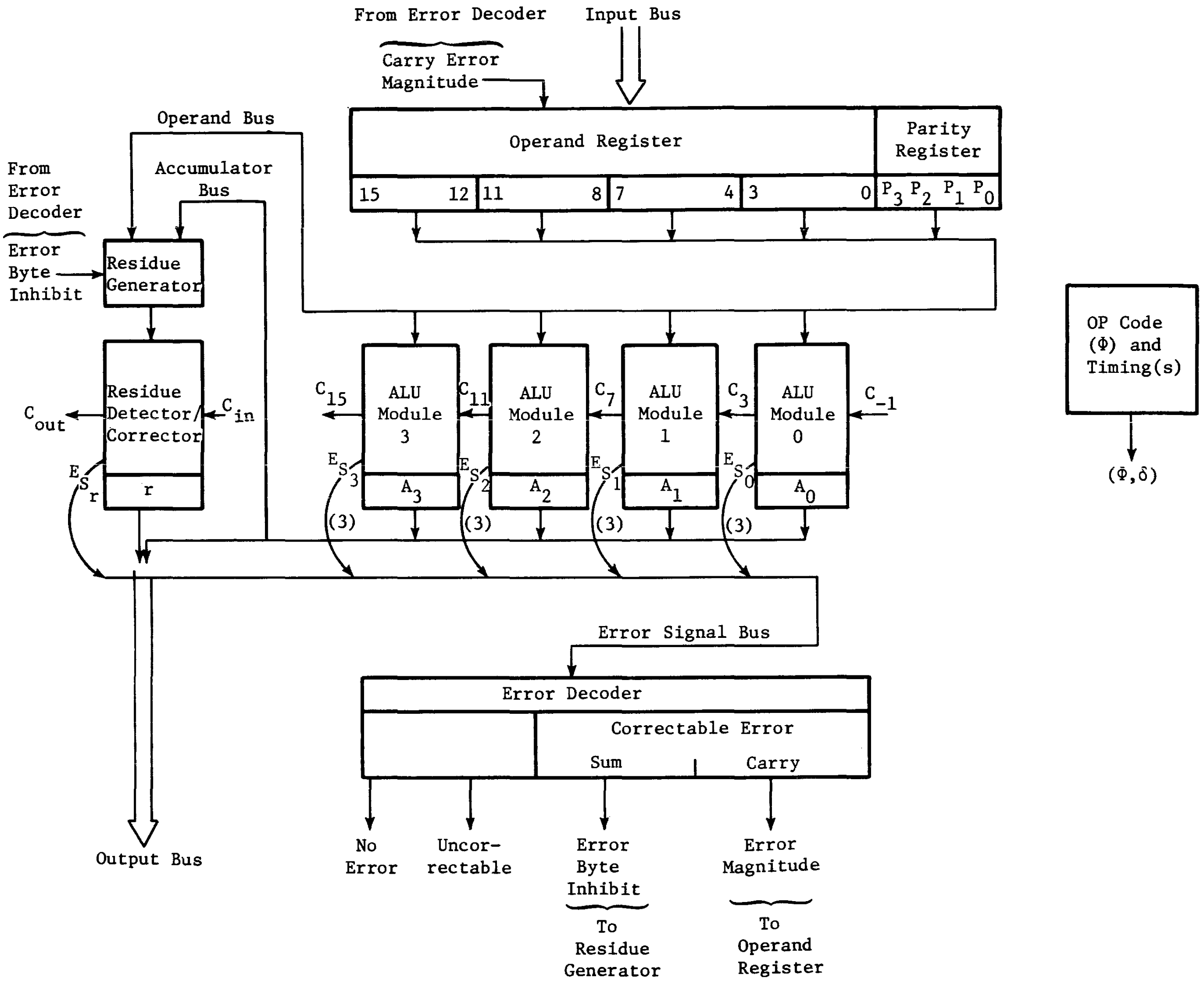


Figure 7—Fault-tolerant modularized 16-bit ALU organization

five of these self-checked 4-bit ALUs for a 16-bit microprocessor with one of these modules as a spare.

To assess the value of an ALU with error correction due to single faults and detection of errors due to multiple faults, such as the one herein, the reliability improvement due to the fault-tolerance needs to be evaluated.

A thorough analysis is necessary to determine the cost of the fault-tolerance. A suggested cost breakdown is as follows: (1) a cost of the full feature; (2) cost of the parity checking; (3) cost of the residue checking; and (4) cost of the ERROR DECODER.

CONCLUSIONS

The combined use of parities and residues as a combination code can be used to provide an ALU organization capable of correcting all errors due to single faults and detection of most errors due to multiple faults.

The code can also be applied to different length ALU's and regardless of the byte size and the number of bytes. This is a definite advantage over arithmetic coding techniques such as biresidue codes where the size n=kc must satisfy (k, c)=1.

As pointed out by Langdon and Tang[3] an ALU that makes use of a parity-checked adder is relatively inexpensive. Further the checking of the logical operations, namely AND, OR, EXOR of the ALU is an automatic by-product of this adder checking scheme. For instance, half-sum check performed for ADD operation is an automatic check of EXOR operation. Checking of other ALU operations, such as, SHIFT, ROTATE can be provided relatively easily by both parity and residue operations.

We have broadly outlined in this report the checking hardware required of the ALU to make it self-checking and self-correcting. The details of implementation are presently under investigation. Based on a preliminary investigation, it appears that the combination code approach to ALU fault-tolerance is effective from both cost and coverage point of views.

ACKNOWLEDGMENTS

The authors acknowledge with gratitude many helpful suggestions from Drs. Se June Hong and W. F. Mikhail and Mr. C. H. Wolff, all of IBM, Poughkeepsie, New York. Part of the research of T. R. N. Rao was supported by National Science Foundation grant number ENG76-11237.

REFERENCES

1. Bloch, Erich, "The Engineering Design of the Stretch Computer," *Proc. EJCC*, 1959, pp. 48-59.
2. Garner, H. L., "Generalized Parity Checking," *IRE Trans. on Elec. Computers,* Vol. EC 7, No. 3, Sept. 1958, pp. 207-213.
3. Hamming, R. W., "Error Detecting and Correcting Codes," *Bell System Tech. Journal.* Vol. 29, 1950, pp. 147-160.
4. Peterson, W. W. and E. J. Weldon, Jr., *Error Correcting Codes,* M.I.T. Press, Cambridge, Mass. 1970.
5. Avizienis, A., "Concurrent Diagnosis of Arithmetic Processors," *Digest IEEE 1st Annual Computer Conf.*, Sept. 1967, pp. 34-37. See also references to earlier work contained therein, notably to 1964-66 JPL reports.
6. Avizienis, A., "Digital Fault Diagnosis by Low Cost Arithmetical Coding Techniques," *Proc.*, Vol. 1, Lafayette Ind., Purdue Univ. Eng. Exp. Sta., April 28-30, 1969, pp. 81-91.
7. Rao, T. R. N., "Biresidue Error Correcting Codes for Computer Arithmetic," *IEEE Trans. Comp.* C-19, May 1970, pp. 398-402.
8. Monteiro, P. and T. R. N. Rao, "A Residue Checker for Arithmetic and Logic Operations," Digest of 1972 International Symposium on Fault-Tolerant Computing, Newton, Mass., June 1972.
9. Reinheimer, H. J., "Error Detecting and Correcting System and Method," IBM Corporation, Gaithersburg, MD. US Patent 3,699,323, October 17, 1972.
10. Payne, A. and H. J. Reinheimer, IBM Corporation, Gaithersburg, MD., U. S. Patent #3,659,089, April 25, 1972.
11. Massey, J. L. and O. N. Garcia,
12. Sellers, F. F., Jr., M. Y. Hsiao and L. W. Bearnson, *Error Detecting Logic for Digital Computers*, McGraw-Hill, NY 1968.
13. Rao, T. R. N., *Error Coding for Arithmetic Processors,* Academic Press, New York, 1974.
14. Garcia, O. N. and T. R. N. Rao, "On the Methods of Checking Logical Operations," *Proc. Second Annual Conference on Information Sciences and Systems,* March 1968, pp. 89-95.
15. Neumann, P. G. and T. R. N. Rao, "Byte Error Correction in Arithmetic Processors," *IEEE Trans. on Comp.,* March 1975.
16. England, W. A., "Improving Reliability by the Application of Selected Redundant Techniques," *Proceedings of Workshop on Reliability Techniques,* UCLA, Los Angeles, April 1966.
17. Langdon, G. G. and C. K. Tang, "Concurrent Error Detection for Group Look-Ahead Binary Adders," *IBM J. Res. Dev.,* Sept. 1970.

The design of self-checking multi-output combinational circuits*

by D. C. KO
Burroughs Corporation
Mission Viejo, California

and

M. A. BREUER
University of Southern California
Los Angeles, California

ABSTRACT

In this paper we present a technique, called Extended-Parity Checking, for the design of error-detecting circuits for combinational logic networks. Its concept is derived from the conventional parity checking technique, which is applicable only for odd number of errors, yet it can detect errors of any degree. A structural model, called the fanout-graph, is introduced which contains a minimum number of nodes sufficient to determine the fundamental causes of multiple errors in a circuit. Output functions are then expressed in a special form, called the Fanout-Observed Output Function (FOOF), which facilitate the analysis of errors. Based on this information and certain circuit parameters, a set of design methods are presented for producing self-checking circuits. Among them, one deals with the addition of external leads by augmenting some of the fanout nodes in the original circuit, while others involve duplicating or checking independently parts of the logic.

INTRODUCTION

The implementation of a self checking system requires appropriate error detecting circuitry. This circuit should generate an *error signal* whenever an output error occurs. This signal can be used to stop computation, signal manual repair work, or initiate a reconfiguration process.

Shown in Figure 1 is a general model of a self-checking system. It consists of two circuits, namely C and D. C is the operating circuit being checked, having input X and output F, both vector-valued. D is a single-output circuit, called the error detecting circuit (or logic), whose output, denoted by ϵ, is the required error signal (subject to timing control). Y is a set of internal signals of C which, depending on circuit constraints, may or may not be available to D. Under our present investigation both C and D are assumed to be acyclic combinational circuits.

The simplest form for a self-checking system is complete duplication in which D properly contains C. In this case, a redundancy ratio of more than 2:1 is expected. Depending on the particular function which C implements, and its structure, some other techniques exist which may sometimes yield a smaller redundancy ratio.[7]

This paper deals with the design of checking circuits. Our goal is to try to achieve a redundancy ratio of less than or equal to 2:1. The technique we are going to investigate is called Extended-Parity Checking (EPC). Its concept is derived from the conventional parity checking scheme. It is well-known that parity checking will fail in case of an even number of errors occurring on the circuit outputs. The EPC on the other hand, will *not* have this deficiency.

FAULT ANALYSIS AND ERROR DETECTABILITY

Let $f:\{0, 1\}^n \rightarrow \{0, 1\}$ be a single-output Boolean switching function over the set of variables $X=\{x_1, x_2, \ldots, x_n\}$. A multi-output Boolean switching function is denoted by $F:\{0, 1\}^n \rightarrow \{0, 1\}^m$ and consists of m single-output functions, i.e., $F=(f_1, f_2, \ldots, f_m)$ where $f_i=f_i(x_1, \ldots, x_n)$ for $i=1, 2, \ldots, m$. Let C be a combinational circuit which realizes F. The set of input lines $\{x_1, x_2, \ldots, x_n\}$ are called *primary inputs* (PI) and the set of outputs $\{f_1, f_2, \ldots, f_n\}$ are called *primary outputs* (PO). We denote an input vector to C by $X_i=(x_1, x_2, \ldots, x_n)$ and the corresponding output vector by $F_j=(f_1, f_2, \ldots, f_m)$.

Let X_i represent the binary input vector whose decimal value is i, e.g. $X_3=(00\cdots011)$ and set $\chi=\{X_i | i=0, 1, \ldots, 2^n-1\}$. By $F(X_k)$ we mean the value of F for input $X=X_k$.

We assume circuits are made up of single-output gate elements such as AND, NAND, OR, NOR, etc. Below are some basic definitions concerning circuit topology.

* This work was supported in part by the National Science Foundation under Grant GK-23886 and in part by the Office of Naval Research under Grant N00014-67-A-0269-0019 (NR 048-299).

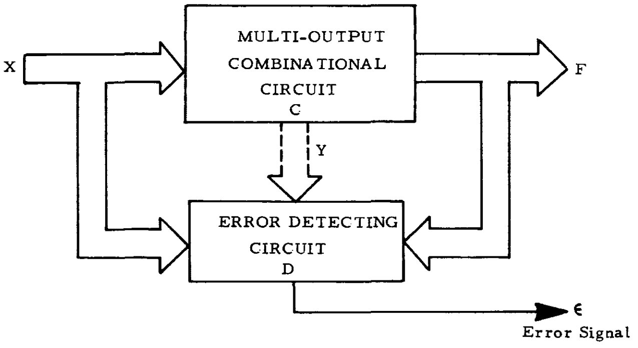


Figure 1—General model of a self-checking system

1. Every PI, PO and gate is a node, called a *signal node* (SN). (We do not differentiate between a gate and its output.)
2. A node is an *internal node* (IN) if it is neither a PI nor a PO nor a node directly connected to a PO.
3. The *fanout value* τ_α of a node α is equal to the number of nodes to which it fans out.
4. A node α is called a *fanout node* (FN), if $\tau_\alpha \geqq 2$.
5. A *signal path* is a sequence of nodes of the form $\alpha_1\alpha_2 \ldots \alpha_n$, where $n \geqq 2$ and α_{i-1} is an input to α_i. A path is *simple* if $\tau_{\alpha_i}=1$ for $i=2, 3, \ldots, n-1$.
6. A *reconvergent node* (RN) is a node having some pair of inputs which are the terminal nodes of paths having a common source node (a FN).
7. A *limited fanout-free* (LFF) circuit is a circuit in which the only FN's are PI's.

In order to simplify the presentation of our results we assume that all circuits being dealt with contain no redundancy. The general case which includes redundancy is dealt with in Reference 8.

In our work we will assume a single permanent stuck-at fault model.

Let C be an irredundant combinational circuit realizing the switching function F, and let $\Delta=\{\delta_1, \delta_2, \ldots, \delta_p\}$ be a set of faults associated with C. Then we denote the circuit C containing fault δ_j by C^j, where $C^0(=C)$ represents the fault-free circuit. C^j realizes the function $F^j(X)=(f_1{}^j(X), f_2{}^j(X), \ldots, f_n{}^j(X))$. If $F^j(X_k)\neq F^0(X_k)$, then fault δ_j is *detectable* by input X_k.

Let $H=\Delta\times\chi$ be the set of all fault-input pairs. Then the *error indicators* $\Delta^j f_i(X_k)$ and $\Delta^j F(X_k)$ are defined as follows. For each $h_{jk}=(\delta_j, X_k)\in H$ we have

$$\Delta^j f_i(X_k)=f_i{}^j(X_k)\oplus f_i{}^0(X_k),$$

for $i=1, 2, \ldots, m$, and

$$\Delta^j F(X_k)=F^j(X_k)\oplus F^0(X_k)$$
$$=(\Delta^j f_1(X_k), \Delta^j f_2(X_k), \ldots, \Delta^j f_m(X_k)).$$

If $\Delta^j F(X_k)=(0, 0, \ldots, 0)=\mathit{0}$ then δ_j is not detected by X_k, otherwise it is. The norm $|\Delta^j F(X_k)|$ is said to be the *Hamming weight* of the vector $\Delta^j F(X_k)$, and equals the number of 1's in the vector.

Suppose that $|\Delta^j F(X_k)|=q$, $q=1, 2, 3$, or $n(4\leqq n\leqq m)$. Then we say there is a single-error, double-error, triple-error, or n-bit error on the circuit output respectively. We call "q" the *degree* of the output error.

Suppose we append to C an associated error detecting circuit D having the following property. If an error in the output of C occurs, the output of D, called the *error signal* and denoted by ϵ, will be set to 1; otherwise its value will be 0. Hence $\epsilon=1$ indicates the detection of an output error in C, and the fault which caused this error is thus detected. We will be concerned with the design of D.

Let $\chi(\delta_j)$ be the set of inputs which detect δ_j in C, i.e. $\Delta^j F(X_i)\neq 0$ for each $X_i\in\chi(\delta_j)$. Since C is irredundant $\chi(\delta_j)\neq\varphi$.

Let χ' be a subset of $\chi(\delta_j)$ such that for each $X_i\in\chi'$, if δ_j is present then $\epsilon=1$.

(a) If $\chi'=\chi(\delta_j)$ then δ_j is said to be *totally checked*.
(b) If $\chi'=\varphi$, then δ_j is said to be *unchecked*, and
(c) If $\varphi\subset\chi'\subset\chi(\delta_j)$ then δ_j is said to be *conditionally checked*.

If all δ_j are totally checked then C is said to be *totally checked*, and if some faults in C are totally checked while others are conditionally checked then C is said to be *conditionally checked*. If some faults are unchecked, then C is said to be *partially checked*.

The combined circuit (C, D) forms a self-checking system which in turn is subject to faults. Our error detecting criteria is defined as follows. For each $h_{jk}\in H'=\Delta'\times\chi$, where Δ' is the set of faults associated with the new circuit (C, D), we require

$$\epsilon=\begin{cases}1 \text{ if } q\neq 0, \text{ or D has a fault}\\ 0 \text{ otherwise.}\end{cases} \qquad (2.1)$$

We assume that the fault ϵ *s-a-0 fault* and any fault equivalent to it in D is not included in Δ'. The problem of detecting output faults in a checker is discussed in Reference 3.

The general form of our forced parity checking system is shown in Figure 2. Here we augment C with the logic $\hat{c}$ having outputs $\hat{\alpha}_1, \hat{\alpha}_2, \ldots, \hat{\alpha}_\lambda$. $\hat{c}$ is designed such that any single fault in C or $\hat{c}$ causes an odd number of outputs from (C, $\hat{c}$) to be in error. $\hat{P}$ is a circuit which implements the parity function defined by the expression

$$\left(\bigoplus_{i=1}^{n} f_i\right)\oplus\left(\bigoplus_{i=1}^{\lambda}\hat{\alpha}_i\right).$$

The outputs of $\hat{P}$ and $\hat{P}'$ are then compared by T' to see if an error has occurred.

ALGEBRAIC STUDY OF CIRCUIT OUTPUT ERRORS

In a multi-output combinational circuit there is the possibility of several output lines being jointly dependent on one signal. If a fault causes an error on that signal then multiple errors may occur on the outputs.

Assume under condition $h_{jk}=(\delta_j, X_k)$ that f_p and f_q are in error, and that δ_j occurs at signal α, i.e. δ_j corresponds to α

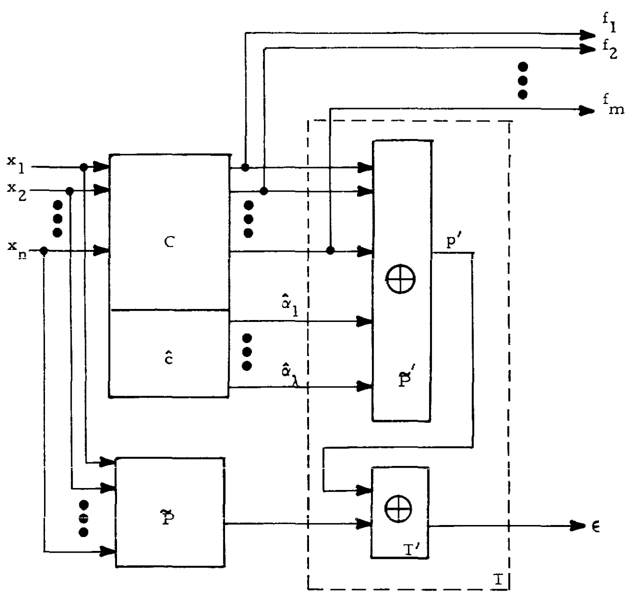


Figure 2—A forced-parity checking system

s-a-1 or α s-a-0. Then (p, q) is said to be the output error *location* of a double-error, and α is the location of the fault. If for condition h_{jk} output lines $f_{l_1}, f_{l_2}, \ldots, f_{l_q}$ are in error, then $(l_1, l_2, \ldots, l_q)$ is said to be the output error location and it consists of the following set of double-error locations: $\{(l_a, l_b) | a<b; l_a, l_b \in \{l_1, l_2, \ldots, l_q\}\}$. The number of output lines in error, their indices, and the output error pattern, are a few parameters one needs to know before an error detecting circuit can be implemented.

In a parity checking system, multiple errors of odd degree (q odd) can always be detected. It is those of even degree which will fail to be detected. We will first consider the case for q=2, since it is the simplest. We will then show how to extend the results to q=4, 6, . . . , etc.

An error is said to be *located* when the output lines in error are identified. One way of locating all possible double-errors in a circuit is to enumerate all pairs of outputs. This can be done for each fault which may exist in the circuit. Depending on the circuit structure, not all output pairs and faults need to be considered. Note, for example, that under the single fault assumption no multiple errors can exist in a fanout-free circuit.

Structural modeling and graph theoretic results

Consider two nodes in C, say α and β. If every path from node β to every PO includes α (α can itself be a PO), then α is said to be *essential* to β, and this is denoted by $\alpha E \beta$. In this case, if a fault in β causes an error in the signal at α when X_i is applied, then one can always assume there is a fault labeled "α s-a-$\bar{\alpha}$" occurring at this time. Thus, it is sufficient to deal with double errors due to faults occurring at α and we can ignore double errors due to faults at β.

A node α is called a *prime node* (PN) in C if there exists no other node β, $\beta \neq \alpha$, such that $\beta E \alpha$ (note that a PI can never be a PN). α is called a *prime fanout node* (PFN) if it is also a fanout node.

In the circuit of Figure 3, nodes α_2, α_3, and α_4 are the only three PFN's. The four primary outputs are PN's only.

Theorem 1: The set of all PFN's form a minimum set of nodes where faults associated with these nodes are sufficient to cause all multiple-output errors in a circuit. □

If all PFN's in a circuit can be identified, then the location of all possible double-errors can be made more easily and efficiently. In addition, the design of error detecting circuits, as we will see in the next section, depends heavily on this information. One method of identifying all the PFN's of a circuit is through a structural modeling process of Ko.[8] In this process, the final circuit model is represented by a directed graph G showing all the PFN's of the circuit. We call this graph G the fanout-graph for the circuit C and it has the following properties:

1. Every node in G is a prime node in C and G contains the complete set of prime nodes of C.
2. There are as many disjoint subgraphs in G as there are disjoint sub-circuits in C (assume PI's can be shared).
3. All nodes in G are singular if C is fanout-free or limited fanout-free.

From these properties and Theorem 1 we immediately conclude that no multiple-output errors can occur in C if C is fanout-free or limited fanout-free. The fanout graph shown in Figure 4 is obtained by applying the structural modeling process to the circuit of Figure 4.

Analyzing circuit output errors using the boolean difference

The *Boolean difference* of a switching function $f=f(x_1, x_2, \ldots, x_n)$ with respect to x_i is defined as

$$\frac{df}{dx_i} = f(x_1, x_2, \ldots, x_i, \ldots, x_n) \oplus f(x_1, x_2, \ldots, \bar{x}_i, \ldots, x_n)$$

and can be written as

$$\frac{df}{dx_i} = h(x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n).$$

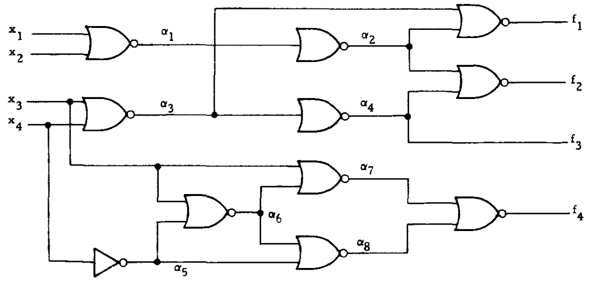


Figure 3—Example Circuit

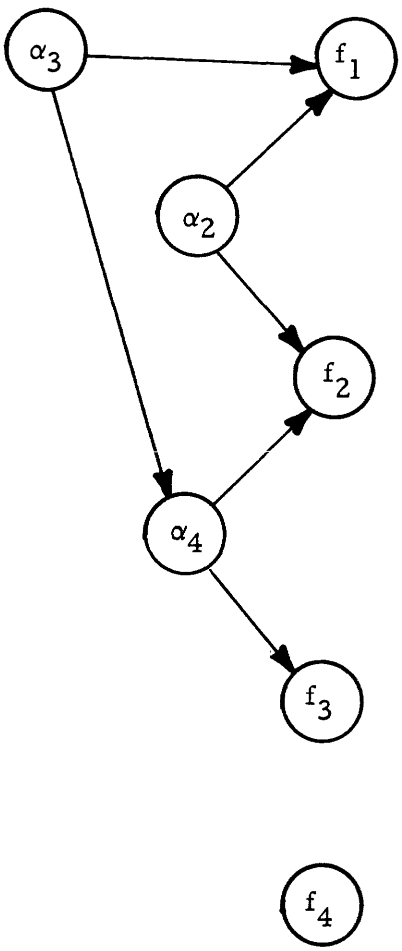


Figure 4—Fanout graph obtained by applying structural modeling process to the circuit of Figure 3

The Boolean difference of f with respect to an internal signal α rather than a primary input can be derived as follows. Write the output function f in the form $f=g(\alpha, X)$ where $\alpha=\alpha(X)$. Thus α becomes an explicit variable of f. Then

$$\frac{df}{d\alpha}=\frac{dg(\alpha, X)}{d\alpha}=g(0, X)\oplus g(1, X).$$

In general, let $f=g(\alpha_1, X)$ and $\alpha_i=\beta_i(\alpha_{i+1}, X)$ for $i=1, 2, \ldots, n-1$. Then

$$\frac{df}{d\alpha_n}=\frac{df}{d\alpha_1}\cdot\frac{d\alpha_1}{d\alpha_2}\cdots\frac{d\alpha_{n-1}}{d\alpha_n}. \tag{3.1}$$

Equation (3.1) is called the Boolean difference *chain* or the *partial* Boolean difference. Two important properties about the Boolean difference are that $df/d\bar{\alpha}=df/d\alpha$ and $d\bar{f}/d\alpha=df/d\alpha$. Note that: (a) if $df/d\alpha\equiv0$, then an error in α will not cause an error in f; (b) if $df/d\alpha\equiv1$, then an error in α will always cause an error in f; and (c) if $df/d\alpha=h(X)$ then an error in α will cause an error in f if and only if $h(X)=1$. Thus $df/d\alpha$ actually defines an *error function* whose value will be used in determining whether or not an error can be sensitized to the output. Let $w_i^\alpha=df_i/d\alpha$ be the error function of the output f_i with respect to α. We define a *pairwise error function* $w_{ij}^\alpha=w_i^\alpha\cdot w_j^\alpha$ as the logical product of two error functions. Three cases exist.

Case 1. If $w_{ij}^\alpha\equiv0$ then an error in α will cause either a single-error or no error on the output pair f_i and f_j.

Case 2. If $w_{ij}^\alpha\equiv1$ then an error in α will always cause a double-error on the output pair f_i and f_j.

Case 3. If $w_{ij}^\alpha=h(X)$ then an error in α will cause a double-error on the output pair f_i and f_j if and only if $h(X)=1$.

Example 1: Consider the circuit of Figure 3. Its fanout graph is shown in Figure 4. From Theorem 1, only the three PFN's namely α_2, α_3, and α_4 need be considered for possible causes of multiple errors. Since f_4 is a singular node, it can be ignored. For the remaining three terminating nodes we express their output functions in terms of the PFN's, i.e.,

$$f_1=\overline{\alpha_2+\alpha_3}=\bar{\alpha}_2\bar{\alpha}_3$$

$$f_2=\overline{\alpha_2+\alpha_4}=\bar{\alpha}_2\bar{\alpha}_4=\bar{\alpha}_2\alpha_3$$

$$f_3=\alpha_4=\bar{\alpha}_3.$$

Since $m=l=3$, there exist a total of $\binom{3}{2}\times3=9$ pairwise error functions. Among them, 4 are trivial. For instance, f_1 is not a function of α_4, hence $w_{12}^{\alpha_4}$ and $w_{13}^{\alpha_4}$ must be 0. The 5 non-trivial ones are

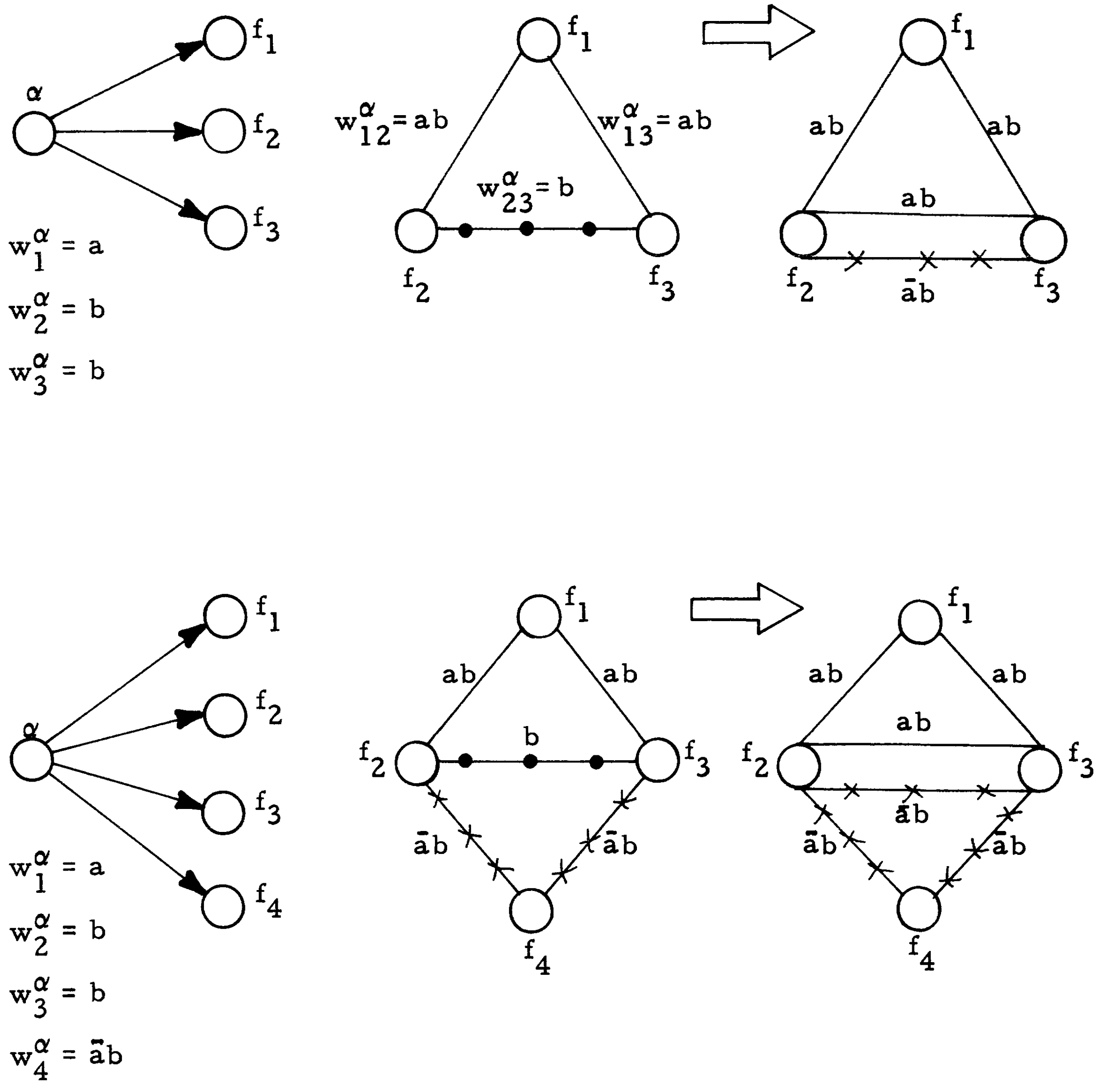
$$w_{12}^{\alpha_2}=\bar{\alpha}_3\cdot\alpha_3=0 \qquad w_{12}^{\alpha_3}=\bar{\alpha}_2\cdot\bar{\alpha}_2=\bar{\alpha}_2$$

$$w_{13}^{\alpha_3}=\bar{\alpha}_2\cdot1=\bar{\alpha}_2=w_{23}^{\alpha_3}=w_{23}^{\alpha_4}.$$

Thus when $\bar{\alpha}_2=x_1+x_2=1$, multiple errors can occur whenever there is an error in α_3 or α_4. The error in α_3 or α_4 can either be a fault in the node itself, or it can be caused by some other fault in a preceding node. In this example, since $w_{12}^{\alpha_3}=w_{13}^{\alpha_3}=w_{23}^{\alpha_3}$, an error in α_3 can cause all three output pairs simultaneously to be in error leaving a net result of a triple-error. Note that an error in α_2 can never cause any double-error because $w_{ij}^{\alpha_2}=0$ for all $1\leqq i, j\leqq4$.

A simple analysis reveals that a double-error 01↔10 will occur on the outputs f_1 and f_2 whenever there is an error in α_3 and the input condition is one which causes $\alpha_2=0$. A different type of double-error, namely 00↔11 can be found in the circuit for the output pair f_1 and f_3. We call "01↔10" and "00↔11" *error patterns*. Theorem 2 in the next section will be devoted to determining such error patterns.

In general, for $q>2$ a q-bit error can be considered as a group of k double-errors where $k=\binom{q}{2}$. Once all pairwise double-errors are located, multiple errors of higher degree can also be located. In order to do this, we introduce the notation of an *error-graph*. It is a non-directed graph such as the one shown in Figure 5(a). In this graph, every node is a terminating node of a fanout-graph. A *link* is entered into the graph if the pairwise error function of two outputs is not



Note: ———, -●-●-●- and -✕-✕-✕-

indicate multiple errors under different sets of input conditions

Figure 5—Fanout-graphs and error-graphs of two circuits

zero. Thus a link actually indicates the possibility of a double-error on its two end-nodes. We use different marks on the links to represent double-errors under different sets of input conditions. A closed triangle of identically marked links represents a possible triple-error.

For the graph of Figure 5(a) we have three pairwise double-errors. They are indicated by the three links representing w_{12}=ab, w_{13}=ab, and w_{23}=b. By rewriting w_{23} as the sum of two product terms ab and $\bar{a}$b and using two distinct links, the resultant graph, shown in Figure 5(b), indicates a triple-error plus one double-error. This obviously will enable us to predict circuit output errors more precisely. Shown in Figure 5(c) is a second graph which shows five pairwise double-errors. By applying the same technique, only two triple-errors can be found in the final graph. With the addition of an extra output f_4 (as compared with the first circuit), this circuit becomes free of any multiple error of even degree. Therefore, this circuit can be parity checked without any further work.

Fanout-observed output functions

Given a circuit C, let α be a prime fanout node in C. If for some output f there exists at least one path between α and f, then the output function of f derived after making a cut at node α (or treating α as a PI) can be expressed as follows:

$$f(\alpha, X)=A(X)\alpha+B(X)\bar{\alpha}+C(X). \qquad (3.2)$$

We call Equation (3.2) the *F*anout-*O*bserved *O*utput *F*unction (FOOF) of f with respect to α. The Boolean switching functions A(X), B(X), and C(X) are called the Boolean *coefficients* of $f(\alpha, X)$. If A(X)=B(X), then f is independent of α. For f to be dependent on α, at least one of the two coefficients A(X) and B(X) must not be zero. A shorthand form of the above equation is

$$f=A\alpha+B\bar{\alpha}+C.$$

By denoting $\alpha^1=\alpha$ and $\alpha^0=\bar{\alpha}$, two special forms of the FOOF can be written as follows:

1. $f=a\alpha^u+b$ (3.3)
2. $f=a\alpha^u\oplus b$ (3.4)

where $u\in\{0, 1\}$; a and b are arbitrary switching functions independent of α.

We call Equation (3.3) the *+-form* and Equation (3.4) the *⊕-form* of the FOOF. In the +-form, f is unate in α. In the ⊕-form, both α and $\bar{\alpha}$ can appear in a minimal normal form expression for f unless the Boolean coefficient b has a constant value.

Lemma 1: If no linear gate or reconvergent fanout exists between α and f, then f can be expressed by a FOOF of the +-form only. □

Consider a FOOF $f(\alpha, X)$ where $\frac{df(\alpha, X)}{d\alpha} \neq 0$. There must exist at least one input $X_k\in\chi$ such that $\left.\frac{df(\alpha, X)}{d\alpha}\right|_{X=X_k}=1$. When this condition is satisfied, α will be sensitized to the output. The value of f under this condition will be $f(\alpha, X_k)$. Let

$$Y_f=\left\{X_k\in\chi \,\middle|\, \left.\frac{df(\alpha, X)}{d\alpha}\right|_{X=X_k}=1\right\} \qquad (3.5)$$

be a non-empty set of all inputs under which α can be sensitized to the output. Set

$$Z_f=\{f(\alpha, X_k)\,|\,X_k\in Y_f\}.$$

Then Z_f is the non-empty set of all possible switching functions realized by f when sensitized by α. Note that Z_f is undefined if $Y_f=\emptyset$, or equivalently $df/d\alpha\equiv0$.

Lemma 2: $Z_f\subseteq\{\alpha, \bar{\alpha}\}$. □

For the next theorem we need the following definitions. Let f and g be two FOOF's with respect to α, where $\frac{df}{d\alpha}\cdot\frac{dg}{d\alpha}\neq0$. We define

$$Y_{fg}=\left\{X_k\in\chi \,\middle|\, \left.\frac{df}{d\alpha}\cdot\frac{dg}{d\alpha}\right|_{X=X_k}=1\right\}=Y_f\cap Y_g$$

and

$$Z_{fg}=\{f(\alpha, X_k)\cdot g(\alpha, X_k)\,|\,X_k\in Y_{fg}\}.$$

Lemma 3: $Z_{fg}\subseteq\{0, \alpha, \bar{\alpha}\}$. □

Theorem 2: Let $f(\alpha, X)$ and $g(\alpha, X)$ be the two FOOF's of a pair of outputs f and g. Then an error in α will cause a 01↔10 error pattern if and only if $Z_{fg}=\{0\}$. It will cause a 00↔11 error pattern if and only if $Z_{fg}\subseteq\{\alpha, \bar{\alpha}\}$. □

Now we will define the "variance" of a FOOF. The uniqueness of a double-error pattern can be determined by the variance of two FOOF's.

A FOOF $f(\alpha, X)$ is said to be *α-invariant* if $Z_f=\{\alpha\}$ or $\{\bar{\alpha}\}$. A FOOF $f(\alpha, X)$ is said to be *α-variant* if $Z_f=\{\alpha, \bar{\alpha}\}$.

A pair of outputs is said to have a *unique* double-error pattern if all possible double-errors associated with the outputs are of either 01↔10 pattern or 00↔11 pattern but not both.

Lemma 4: Any FOOF of the +-form is α-invariant. □

Theorem 3: A pair of outputs have a unique double-error pattern if both FOOF's are α-invariant (assume error in α only). □

Corollary 1: In a non-reconvergent fanout circuit, if no linear gates are in the circuit, then all double-errors have a unique error pattern. □

In the remainder of this section we will discuss some equivalent forms of FOOF's. An α-augmented function will then be introduced and an augmented parity function will be investigated. These results will aid us in the design of error detecting circuitry. Their application can be found in the next section.

Lemma 5: If $f=a\alpha^u+b$, then $f=\frac{df}{d\alpha}\alpha^u+b$. □

Lemma 6: For $\#\in\{+, \oplus\}$, if $f=a\alpha^u\#b$ then $f=\frac{df}{d\alpha}\alpha^u\oplus b$. □

Theorem 4: Any FOOF of the form $f=a\alpha^u\#b$ can be expressed in one of the following two forms:

1. $f=\frac{df}{d\alpha}\alpha^u\#b$
2. $f=\frac{df}{d\alpha}\alpha^u\oplus b$. □

We now define an *α-augmented function* as a Boolean switching function of the form $Q(\alpha, X)=w(X)\alpha^u$ where w(X) is an arbitrary switching function (for our application, w(X) is an error function). Note that $Q(\alpha, X)$ is also a FOOF of a special form. This function can be implemented to augment a prime fanout node α of a circuit such that a q-bit output error (q even) can be transformed into a (q+1)-bit error.

Consider the case when there exist two outputs f and g in a circuit C, where $f=a\alpha^u\#b$ and $g=c\alpha^u\#d$ are the two associated FOOF's. The pairwise error function is $\frac{df}{d\alpha}\cdot\frac{dg}{d\alpha}$. Suppose $\frac{df}{d\alpha}\cdot\frac{dg}{d\alpha}\not\equiv0$, then let $Q=\left(\frac{df}{d\alpha}\cdot\frac{dg}{d\alpha}\right)\alpha^u$ be an α-augmented function. Under any input $X_k\in Y_{fg}$, an error in α will cause a double-error on the two outputs f and g.

Since $\left.\frac{dQ}{d\alpha}\right|_{x=x_k}=1$, the output Q will also be in error. The net result is a triple-error on the outputs f, g, and Q. The parity function for these three outputs is $P=f\oplus g\oplus Q$ and is called the *augmented parity function*.

Lemma 7: $P=\left(\frac{df}{d\alpha}+\frac{dg}{d\alpha}\right)\alpha^u\oplus b\oplus d$. □

It is seen that both P and Q contain the terms $\frac{df}{d\alpha}$ and $\frac{dg}{d\alpha}$. In the implementation of P and Q, if $\frac{df}{d\alpha}$ and $\frac{dg}{d\alpha}$ can be built once and shared by both P and Q, then a saving in the hardware can be achieved. Provision must be made that a fault in the node $\frac{df}{d\alpha}$ or $\frac{dg}{d\alpha}$ must not cause any double-error on the outputs P and Q. Otherwise, it cannot be detected. Let $\beta=\frac{df}{d\alpha}$ and $\gamma=\frac{dg}{d\alpha}$ be the two nodes of interest. We require $w_{PQ}{}^{\beta}=w_{PQ}{}^{\gamma}\equiv 0$ where w's are the pairwise error functions for P and Q.

Theorem 5: $w_{PQ}{}^{\beta}=w_{PQ}{}^{\gamma}\equiv 0$. □

EXTENDED-PARITY CHECKING

In this section we will show how to apply the preceding theory to the design of checking circuits.

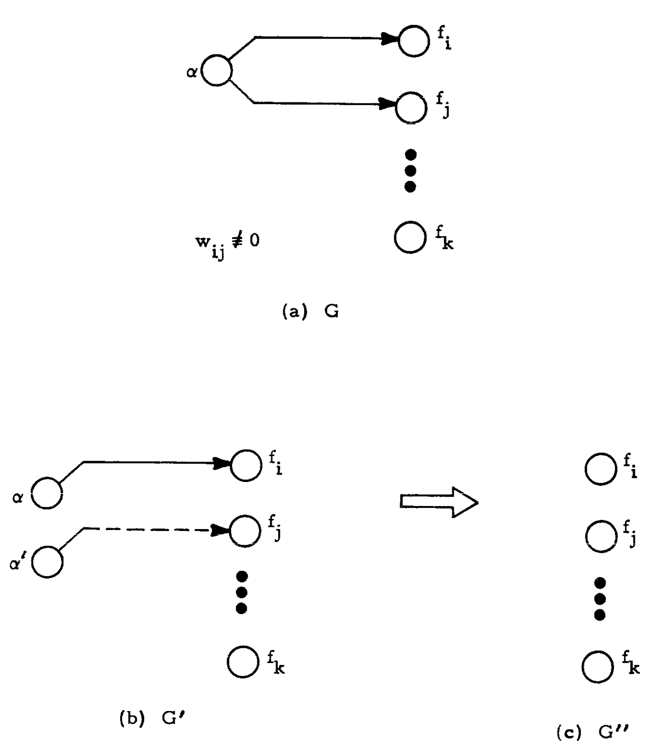
Forced-parity methods

Two methods will now be presented in which additional hardware is introduced. By using these methods one can be assured that the output error of a circuit will always be of odd degree. In this case, a parity checker alone is sufficient to detect all output errors. This approach is invalid if certain PFN's of a circuit are inaccessible. However, it can serve as a design guide in the initial layout of self-checking circuitry.

Fanout degeneration

Given a circuit C, let α be a PFN of C. The fanout value τ_α can be interpreted in two ways. One is the actual number of fanouts of α in C. The other one is the outdegree of α as it appears in the fanout-graph for C. Unless otherwise indicated we will use the latter definition.

Now consider a circuit C whose fanout-graph G is shown in Figure 6(a). The only PFN is α and $\tau_\alpha=2$. It has two associated outputs f_i and f_j. Assume an error in α can cause a double-error on the two outputs. In order to eliminate this double-error, we can remove either one of the two branches αf_i or αf_j from G. The resultant graph G′ with αf_j removed is shown in Figure 6(b). In G′, α is no longer a prime node (since $f_i E \alpha$) and hence can be deleted. The final graph G″ is shown in Figure 6(c). Since G″ is a singular graph, no multiple errors can occur on the outputs.



(a) G

(b) G′

(c) G″

Figure 6—Fanout-graphs for a circuit before and after fanout degeneration

The removing of the branch αf_j from G corresponds to a *degeneration* in the number of fanouts of α in C. This can be accomplished by constructing a new signal α' to replace one or more† of the fanout signals of α. Here α' is logically identical to, yet structurally independent of α. In other words, if C(α) and C(α') are two sub-circuits whose outputs are α and α' respectively, we have $\alpha=\alpha'$ with or without some commonly shared components. A check for new multiple errors must be made, unless C(α') is a duplicate of C(α) and is fed only by PI's. The new circuit, labeled $\tilde{C}$, can be parity checked. Note that for G″ to be singular does not necessarily imply $\tilde{C}$ is fanout-free. This method is essentially a resynthesis procedure since no new output leads are formed.

The method can be greatly enhanced if, instead of completely removing a PFN α from G, α is allowed to stay in G so long as an error in α cannot cause any multiple error of even degree in C. Consider a circuit whose fanout-graph G is shown in Figure 7(a). The error characteristics of this circuit are represented by a Venn diagram shown in Figure 7(b). In this diagram each element Y_i, Y_j or Y_k is a set of input n-tuples defined by Equation (3.5). There are two possible double-errors in the circuit as are indicated by their intersection Y_{ik} and Y_{jk}. By removing the set Y_k from this diagram, all double-errors can be eliminated. The removal of Y_k corresponds to the deletion of a directed branch αf_k from G. So we conclude that only one signal α' needs to be generated. This signal α' will be used to implement f_k. The result is a reduction in τ_a from 3 to 2, and the resulting circuit will be free of any multiple errors.

† f_j can be a reconvergent node.

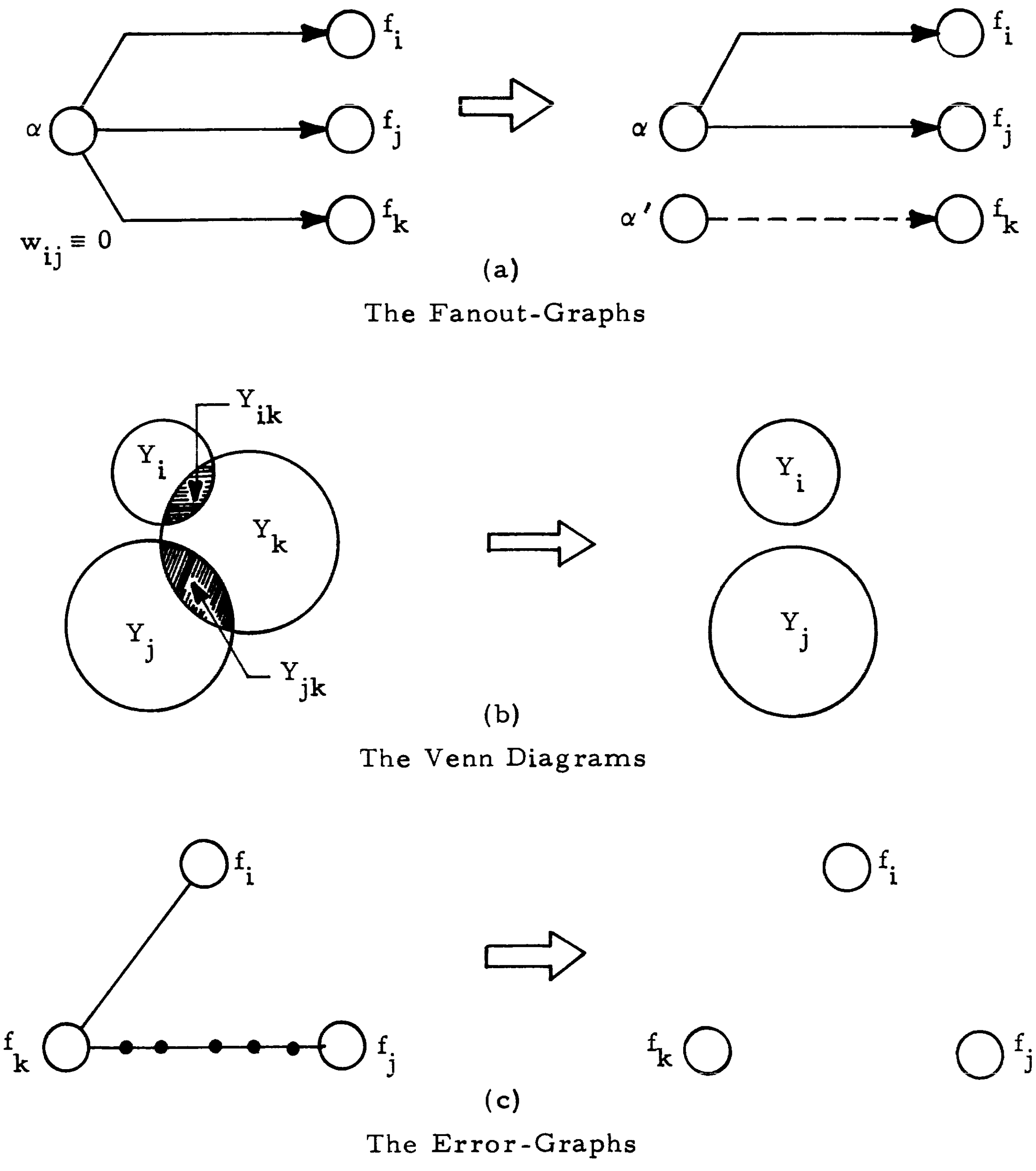
Shown in Figure 7(c) is the change in the error-graph for this circuit. Since there is a close resemblance between the Venn diagram representation and the error-graph, we will use the latter as a working model in our future applications.

For the circuit just presented, the decision on removing αf_k is obvious. For more complicated problems a general procedure is required in order to select pairs (αf_i) to be removed from G. One such heuristic procedure is given in Ko [8], and for brevity, will not be presented here.

Once a node has gone through the degeneration process the fanout-graph is simplified accordingly, and the process is repeated for another PFN. Since each iteration of this process removes a PFN from G, this procedure will terminate when the final graph reaches a singular graph.

Fanout augmentation

Contrary to the previous method, the Fanout Augmentation method does not require any duplication of the fanout



(a)
The Fanout-Graphs

(b)
The Venn Diagrams

(c)
The Error-Graphs

NOTE: ——— and —●—●—●— indicate double-errors under different sets of input conditions

Figure 7—Graphic results showing net changes in a fanout degeneration process

signals. Instead, a line is tapped off on a fanout node which after some gating logic is sent to a parity checker. The new output signal, say $\hat{\alpha}$ will perform the function of transforming any multiple error of even degree into odd degree as long as it is caused by an error in α.

Consider the fanout-graph of Figure 6(a). Instead of removing the branch αf_j from G, we want to add a new branch $\alpha\hat{\alpha}$ to G such that the double-error on f_i and f_j can be transformed into a triple-error on f_i, f_j and $\hat{\alpha}$. The transformation can be achieved by implementing an α-augmented function $\hat{\alpha}(\alpha, X)=w_{ij}(X)\cdot\alpha^u$ where w_{ij} (the error function) will be our gating function. Since $d\hat{\alpha}/d\alpha=w_{ij}$, an error in α will also cause $\hat{\alpha}$ to be in error whenever $w_{ij}=1$ under some input in Y_{ij}.

We will now show how this method will affect the fanout-graph and error-graph of a circuit. Consider the circuit whose fanout-graph is shown in Figure 7(a). In this circuit, there exist two possible double-errors on the output pairs (f_i, f_k) and (f_j, f_k). By implementing a new signal $\hat{\alpha}=(w_{ik}+w_{jk})\cdot\alpha^u$, each double-error can be transformed into a triple error. The resultant graphs showing these changes can be found in Figure 8.

Theorem 6: Given a fanout-graph G. If $B_\alpha=\{\alpha f_{l_1}, \alpha f_{l_2}, \ldots, \alpha f_{l_M}\}$ where f_{l_i}, $1\leq i\leq M$, is a terminating node in G, then

$$\hat{\alpha}=\left[\left(\overline{\bigoplus_{i=1}^{M} w_i}\right)\cdot\left(\sum_{i=1}^{M} w_i\right)\right]\cdot\alpha^u \quad \text{where} \quad w_i=\frac{df_{l_i}}{d\alpha}. \quad \square \tag{4.1}$$

Theorem 7: Let α be a PFN in G and $B_\alpha=\{\alpha\alpha_1, \alpha\alpha_2, \ldots, \alpha\alpha_M\}$ be the set of all M directed branches whose starting-node is α, and end-nodes are α_i, $1\leq i\leq M$. Associated with each node α_i is a set N_i consisting of all terminating nodes having a path from α_i. We will allow the case where $N_i=\{\alpha_i\}$ and call $\alpha_i\alpha_i$ a legitimate path. If $N_i\cap N_j=\emptyset$ for all $i\neq j$, and if every PFN in G is to be processed by the augmentation technique, then

$$\hat{\alpha}=\left[\left(\overline{\bigoplus_{i=1}^{M} W_i}\right)\cdot\left(\sum_{i=1}^{M} W_i\right)\right]\cdot\alpha^u \tag{4.2}$$

$$\text{where} \quad W_i=\sum_{f_k\in N_i}\frac{df_k}{d\alpha}. \quad \square$$

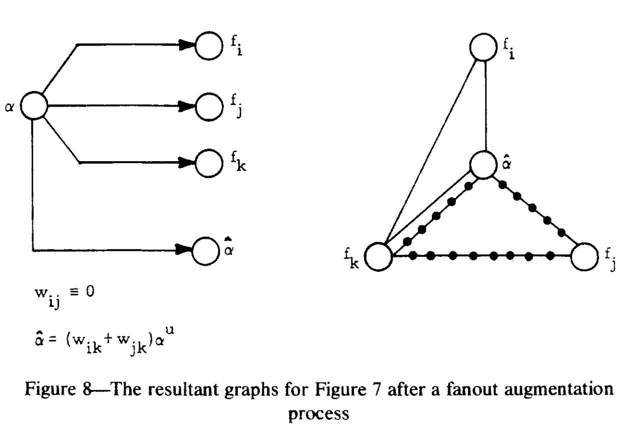


Figure 8—The resultant graphs for Figure 7 after a fanout augmentation process

Note that if every node α_i, $1\leq i\leq M$ is a terminating node, then $W_i=w_i$. In this case, Equation (4.1) and Equation (4.2) are the same.

Line sensing techniques

Under circumstances when circuit constraints or other factors prohibit the use of forced-parity techniques, the *Line Sensing* techniques should be investigated since they may provide a good result. In this section we will discuss two methods which do not require accessing to the PFN's.

Conditional line sensing

In the fanout degeneration method, if a branch αf_i is removed from G, we need to build a new signal α' to replace the line(s) being cut in C. In this method we will build the same α' (or its complement), not for replacement but for comparison. Consider the example of Figure 6 where $B_\alpha=\{\alpha f_i, \alpha f_j\}$. Suppose f_i is α-invariant and we decide to sense f_i. Under input conditions such that $w_i=1$ we will have $Z_i=\{\alpha\}$ or $\{\bar{\alpha}\}$. Since Z_i contains only a single element, we can associate with f_i a switching function α^u and call it the function realized by f_i under $w_i=1$. Let $\alpha'=\alpha^u$ and it can be used to compare with the "line" f_i under the "condition" of $w_i=1$. Shown in Figure 9(a) is such a scheme, and we call it the *Conditional Line Sensing* method. In this method, an Exclusive-OR gate is used to compare f_i with $\alpha^u(X)$. Its output is then gated by the switching function $w_i(X)$. The final output is an error signal ϵ_i which will be set to 1 whenever an error in α causes an error on f_i independent of whether or not f_j is in error. Let ϵ_p be the output of a parity checker checking on all the outputs. Then $\epsilon=\epsilon_p+\epsilon_i$ will be our overall error signal for the circuit. Note that an error of f_i caused by some other source may or may not set $\epsilon_i=1$. That is why f_i should also be included in the parity checking.

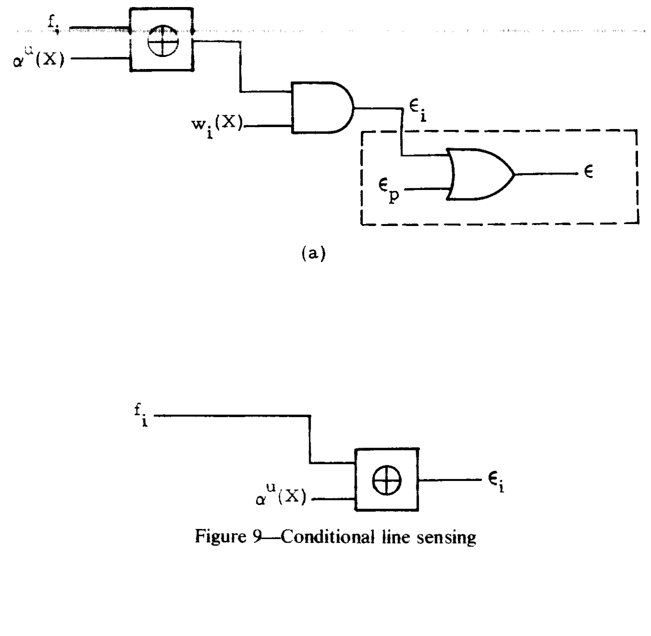


Figure 9—Conditional line sensing

Mathematically, we have

$$\epsilon_i = w_i(f_i \oplus \alpha^u)$$

A special situation is $w_i \equiv 1$ in which case

$$\epsilon_i = f_i \oplus \alpha^u.$$

Its implementation is shown in Figure 9(b).

It should be pointed out that the gating function w_i can actually be replaced by the pairwise error function w_{ij}. When this is done, ϵ_i will be set to 1 whenever a double-error occurs on the outputs f_i and f_j. Thus what is undetected by the parity checker ($\epsilon_p = 0$) will now be detected by the line sensing mechanism ($\epsilon_i = 1$) and the result is $\epsilon = 1$. For the graph of Figure 7 a gating function of $w_{ik} + w_{jk}$ will also be appropriate if f_k is α-invariant. In any event, one should select that implementation which is of least cost.

Now let us consider the case when an output function f_i is α-variant. Since $Z_i \subseteq \{\alpha, \bar{\alpha}\}$, α^u alone will no longer be sufficient to serve as a reference signal. In order to solve this problem, we express f_i in the general form $f_i = A\alpha + B\bar{\alpha} + C$. The error function w_i is readily found to be $(A \oplus B)\bar{C} = A\bar{B}\bar{C} + \bar{A}B\bar{C} = W_1 + W_2$, where $W_1 = A\bar{B}\bar{C}$, and $W_2 = \bar{A}B\bar{C}$. It is seen that when $W_1 = 1$, f_i will be equal to α, and when $W_2 = 1$, f_i will be equal to $\bar{\alpha}$. So clearly we can write the following equation:

$$\epsilon_i = W_1(f_i \oplus \alpha) + W_2(f_i \oplus \bar{\alpha}).$$

Again, all the previous arguments will still hold for each term in ϵ_i.

For any circuit, if more than one ϵ_i is generated, then ϵ should be set as follows:

$$\epsilon = \epsilon_p + \sum_i \epsilon_i. \qquad (4.3)$$

Unconditional line sensing

In this method the input condition plays no important role in the design. First of all, the double-error patterns of a pair of outputs (f_i, f_j) have to be determined. If it has a unique double-error pattern then before duplicating a line f_i, we first perform a functional mapping on f_i and f_j as follows:

1. If (f_i, f_j) has a unique 00↔11 pattern, then let g_i be a function defined by any one of the following expressions:

(a) f_i	(d) $\bar{f}_i$
(b) $f_i + f_j$	(e) $\bar{f}_i \cdot \bar{f}_j$
(c) $f_i \cdot f_j$	(f) $\bar{f}_i + \bar{f}_j$

2. If (f_i, f_j) has a unique 01↔10 pattern, then define g_i to be any one of the following:

(a) f_i	(d) $\bar{f}_i$
(b) $f_i + \bar{f}_j$	(e) $\bar{f}_i \cdot f_j$
(c) $f_i \cdot \bar{f}_j$	(f) $\bar{f}_i + f_j$

The criterion in selecting one of the expressions in each group as g_i is based on the cost of implementing such a function. Once g_i is selected, we can implement a comparison scheme such as the one shown in Figure 10. Let G_i be a circuit which realizes g_i and is implemented using only the signal PI's as inputs. We construct another circuit G_i' which realizes the same function g_i but its inputs are now taken directly from f_i and f_j. Call its output g_i'. We can perform the following comparison

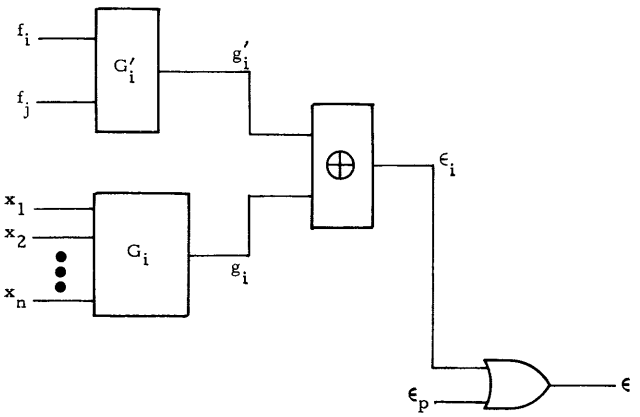


Figure 10—Unconditional line sensing

$$\epsilon_i = g_i(X) \oplus g_i'(f_i, f_j)$$

using only one Exclusive-OR gate. We call this method the *Unconditional Line Sensing* method since the function of w_i is no longer involved.

In this method, unless g_i is chosen to be f_i or $\bar{f}_i$, all the outputs are still required to be sent to a parity checker. On the other hand, if g_i equals f_i or $\bar{f}_i$, then f_i can be excluded from the parity checking. In this case, a partial duplication is implied. For the case when a pair of outputs do not have a unique double-error pattern, we require g_i to be either f_i or $\bar{f}_i$. As was mentioned before, if more than one ϵ_i is generated, then Equation (4.3) will have to be used.

As a final note we would like to point out that this method can be extended to include the case of setting $g_i = \alpha_i$ if a PFN α_i is accessible. We call this method the *Fanout Duplication* method.

In conclusion, these methods have been used to design checking circuits for a number of functional devices as well as random logic. Examples and conclusions dealing with the suitability of specific techniques of different types of logic circuits, such as iterative arrays can be found in Reference 8.

REFERENCES

1. Bouricius, W. G., W. C. Carter, K. A. Duke, J. P. Roth, and P. R. Schneider, "Interactive Design of Self-Testing Circuitry," *Proc. Purdue Centennial Year Symp. on Inform. Processing*, April 1969, pp. 73-80.
2. Carter, W. C. and P. R. Schneider, "Design of Dynamically Checked Computers," *Proc. IFIP*, Vol. 2, August 1968, pp. 878-883.

3. Carter, W. C., D. C. Jessep, and A. B. Wadia, "Error-Free Decoding for Failure-Tolerant Memories," *Proc. IEEE International Computer Group Conference,* June 1970, pp. 229-239.
4. Friedman, A. D. and P. R. Menon, *Fault Detection in Digital Circuits,* Prentice-Hall, Englewood Cliffs, New Jersey, 1971.
5. Gerrand, F. and H. S. Rasmussen, "Self-Correction in Large Scale Digital Computers," *Proc. National Symposium on Reliability and Quality Control,* Philadelphia, Pennsylvania, January 1961, pp. 351-360.
6. Harary, F., R. Z. Norman, and D. Cartwright, *Structural Models—An Introduction to the Theory of Directed Graphs,* J. Wiley, New York, 1965.
7. Kautz, W. H., "Automatic Fault Detection in Combinational Switching Networks," *Proc. 2nd Annual Symposium on Switching Circuit Theory and Logical Design,* 1961, pp. 195-214.
8. Ko, D. C., "Self-Checking of Multi-Output Combinational Circuits Using Forced-Parity Techniques," University of Southern California Electronic Sciences Laboratory Report No. 451, June 1973.
9. Russell, J. D. and C. R. Kime, "Structural Factors in the Fault Diagnosis of Combinational Networks," *IEEE Trans. on Computers,* Vol. C-20, November 1971, pp. 1276-1285.
10. Sellers, F. F., M. Y. Hsiao, L. W. Bearnson, "Analyzing Errors with the Boolean Difference," *IEEE Trans. on Computers,* Vol. C-17, July 1968, pp. 676-683.
11. Sellers, F. F., M. Y. Hsiao, and L. W. Bearnson, *Error Detecting Logic for Digital Computers,* McGraw-Hill, New York, 1968.
12. Wilcox, R. H. and W. C. Mann, eds., *Redundancy Techniques for Computing Systems,* Spartan Books, Washington, D.C., pp. 205-228, 1962.

Remote terminal emulation in the procurement of teleprocessing systems

by SHIRLEY WARD WATKINS and MARSHALL D. ABRAMS
National Bureau of Standards
Washington, D.C.

ABSTRACT

This paper addresses some of the problems which exist when benchmarking interactive computing. The teleprocessing workload may be emulated by a program running internal to the System Under Test (SUT), known as an internal driver or internal stimulator. The limitations of internal drivers are discussed, especially with respect to procurement testing. The use of live operators and tape loops are also discussed, but these are also limited techniques. The most attractive alternative is to employ another, external, computer system to emulate the teleprocessing workload; this approach is called remote terminal emulation. The emulation constraints are delineated; terms applicable to the process are defined, including: Remote Terminal Emulator (RTE), scenario, script, and scene. Ten RTE's, representative of current capabilities, are briefly described.

INTRODUCTION

Benchmarking is the process of executing a mix of representative programs on a computer system in order to validate the performance of that system.[1] The technique is intuitively appealing and yields numerical results which are useful for comparing alternative computer systems. In conducting benchmark tests, it is desirable that the workload be as controlled and repeatable as possible. While this problem has been solved for batch workloads, several problems remain for interactive computing. This paper focuses on a class of devices, called Remote Terminal Emulators (RTE's) which are most attractive for imposing the interactive workload.

Test conditions

It is important to note that the present state-of-the-art makes it necessary to do comparative rather than absolute evaluation. The performance of a multi-programmed computer system is the result obtained from a complex interaction of such variables as the workload, configuration, and operating system.[2]

Complete control of the environment is required for stress testing, tuning, and complete systems comparison. Complete systems comparison coupled with definition of performance criteria in functional machine-independent terms is especially important for the procurement of computer systems where the comparison of heterogenous systems is required.[3,4] Testing with partial control of the environment is easier, potentially less disruptive to normal operations, and less costly than with complete control. Among the applications for partial control are functional demonstrations, quality control, and comparison of computer services.

Test methodologies

Selection of test methodology must incorporate consideration of the measurement conditions and the functional performance measures.[5] Cost, complexity, and accuracy must weigh in the selection. Associated with every test methodology must be a monitoring technique to acquire data and a procedure for analyzing the acquired data. Both data acquisition and analysis must be capable of withstanding scrutiny concerning accuracy and precision.

The teleprocessing workload may be emulated by a program running internal to the System Under Test (SUT), either in the central processing unit, the communications front-end, or, when the architecture supports it, some other processor configured as part of the system. These programs are known collectively as internal drivers or internal stimulators. The monitoring function is included in the internal driver since there may be no external communication. These internal drivers range in sophistication from ones which simply read a simulated terminal communication from a storage device such as tape or disk and present it to the operating system or applications program, to ones which incorporate the use of a dedicated communications processor which is externally cabled to the communications device which would normally be configured.

One objection to the use of internal drivers in procurement testing is concerned with the consumption of resources in the SUT. This may not be a problem in testing services, where the emulated user is one among many, but it does pose serious problems when the total workload is to be emulated. There is no acceptable way to compensate for this resource consumption. Another objection in procurement has to do with the difficulty of verifying the operation of the internal driver. Many internal drivers can bypass various amounts of hardware and software, depending on how the system software is generated. It is therefore extremely difficult to establish exactly what is being tested.

Live operators at terminals may be employed to impose a controlled workload under certain limited situations. Due to the logistics involving the operators and terminals, conditions are difficult to control. In addition using live operators almost eliminates repeatability of testing.

When only a few terminal interactions need to be controlled, the use of tape loops is feasible. The operator input is stored on a tape and the terminal is operated with the tape serving as the source of input. Barring malfunction, the tape input is controllable and repeatable. However, the logistics of employing real terminals is a drawback.

When a large number of terminals and tests are required, the preceding two methods are unequal to the task. When the drawbacks of internal drivers prohibit their use, the alternative is to employ another, external, computer system to emulate the teleprocessing workload.

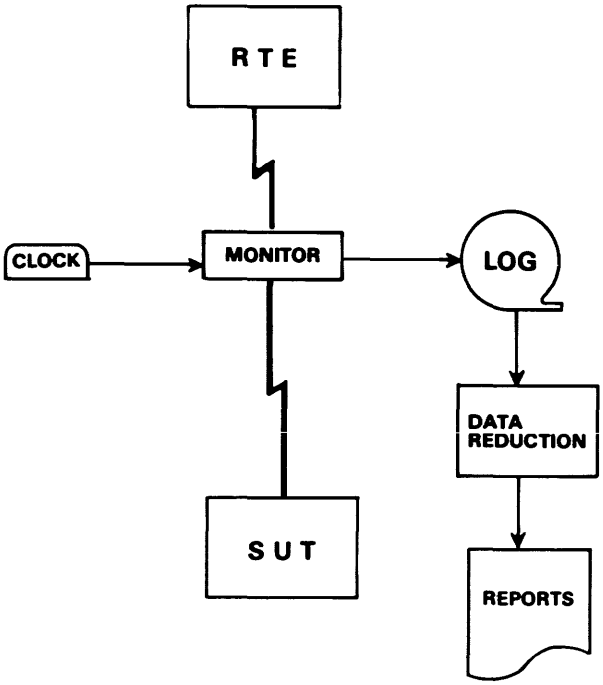


Figure 1—Remote terminal emulation

REMOTE TERMINAL EMULATION AND RELATED TERMINOLOGY

Remote terminal emulation is an approach to the performance evaluation of teleprocessing systems in which a driver external to and independent of the SUT connects to the SUT through its communications device interfaces, either locally or through a communications network, and interacts with the SUT as if the driver were a set of terminal devices and operators. The normal communication protocols are used. In fact, the SUT should be unable to distinguish between communicating with the driver, and with real users and terminal devices. Integral to this technique is a monitor which captures data descriptive of the driver/SUT interaction. Through analysis of this data, performance determinations are made.

Remote terminal emulator

An RTE is a specific implementation of a teleprocessing workload driver employed in remote terminal emulation. RTE's are implemented on various sizes of machines—from minicomputers to large scale computers. A monitor to record selected events is a required component of the remote terminal emulation process. All known RTE's incorporate this monitor component. Figure 1 demonstrates the components of the remote terminal emulation process.

Scenario

A scenario describes the user workload in a machine independent form. Functional activities to be emulated, such as the exercise of various subsystems (compilers, editors, application packages, etc.), are specified in the scenario. All actions, pauses, and decisions to be made by the emulated users are designated. Ideally scenarios are written independent of any SUT or RTE. Figure 2 provides a sample scenario.

Script

A scenario is translated into a script dependent on both the SUT and RTE. The script contains the characters which constitute the user/system interaction, and the time sequence information describing the relationship among the characters. The script contains many elements in addition to a source language program and a set of data. For example, an interactive script might specify logging onto a system, creating a program to be stored in the file system, compiling, linking and loading, and executing. In execution, input and output might be conducted between the program and the terminal as well as between the program and

Enter program "A."
Submit program for compilation.
Correct errors in lines 10 and 15.
Submit program for compilation.
Correct all remaining errors.
Enter data "B" into file system.
Execute program "A" using data "B."

Figure 2—Sample scenario

various files. A script may contain a mixture of commands at the executive command level, subcommands and other interactions with various subsystems, programs, and data. Embedded in the sequence of this task execution are typically commands to the RTE which indicate the elapsed time to wait after receiving a system message before issuing the next input from an emulated device. Figure 3 is an overview of the scenario-to-script process and Figure 4 shows the hypothetical translation of a scenario statement into the text which would be included in the scripts for three different SUT's.

Scene

When an RTE is applied to a SUT, the totality of characteristics associated with that application is called a scene. In order to determine the performance of the SUT, it is necessary to record predefined aspects of the RTE/SUT interaction. This procedure is called scene monitoring. The scene characteristics to be logged are determined by the features of SUT performance to be evaluated. An example of the type of information logged is SUT response time.

Integrity confirmation

Applying an RTE to a system for the purpose of performance evaluation raises three questions:

1. What is the performance of the SUT?
2. Can the RTE execute the necessary functions?
3. Is the RTE performing properly during its application to the SUT?

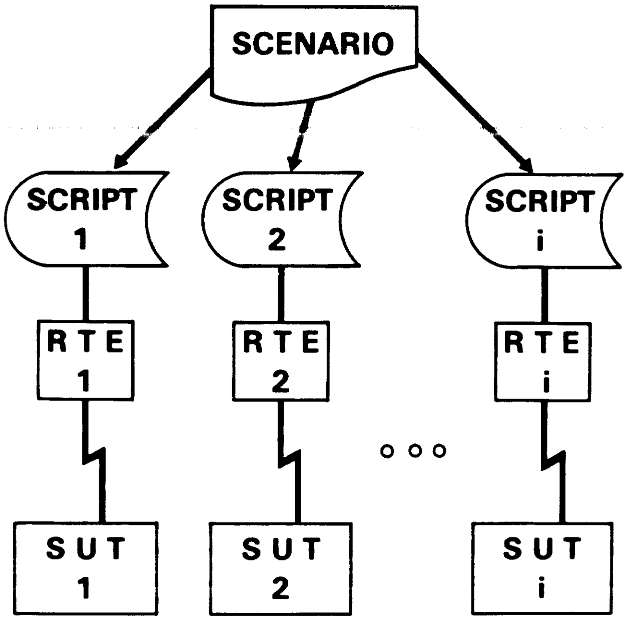


Figure 3—Scenario-to-script process

```
Scenario: Enter program "A."
Script 1: @ ED, I AYE
          (type text of program "A")
          @EOF
Script 2: MAKE AYE.FOR
          I (type text of program "A")
          ⟨ESC⟩EX⟨ESC⟩⟨ESC⟩
Script 3: TECO
          I (type text of program "A")
          ⟨ESC⟩
          ⟨ESC⟩
          AYE.FOR ⟨CR⟩⟨CR⟩
```

Figure 4—Scenario-to-script translation

Integrity confirmation encompasses these three concerns.

To insure the integrity of an application of an RTE or equivalent device, the performance of that device should be tested. This testing can be performed prior to and/or during its application to the SUT. Examples of the type of concern addressed in this process are: is the device executing the tasks specified in the scenario, is the device emulating the correct number and kinds of devices specified at the correct transmission speeds, and is the device recording the scene accurately for later analysis?

CURRENT RTE'S

RTE's have been implemented on varying sizes of systems for varying purposes. The majority of these devices were originally developed for in-house testing by vendors of computer systems and services. For the most part, vendors recognized the need for a tool to be used for such activities as the stress testing, software debugging, and tuning of teleprocessing systems. As the RTE's matured and their utility became recognized, in some companies they became marketing tools, and, in some cases, marketed software products. It is interesting to note that while vendor-developed RTE's were designed independently, they are functionally very similar.

The scope of the description of RTE's has been restricted to computer system testing for this paper. The purpose of the descriptions which will be provided is strictly to illustrate the level of capability and capacity which is currently available in RTE's.

This paper references ten RTE's which were developed for testing computer systems.[6] Of these ten, only one, the Air Force/MITRE RTE was designed to test any computer system; the others were designed by vendors for their product line. Many of these RTE's may be applicable to other systems by virtue of the generality of their design. Figure 5 summarizes the RTE's and the hardware on which each is implemented.

The three minicomputer-based RTE's are the Air Force/MITRE RTE, Digital Equipment Corporation's Script Machine, and Hewlett Packard's Timesharing Event Perform-

RTE	Hardware
Air Force/MITRE DVM	Data General Nova 800
Control Data Corp. BARTER	Peripheral Processing Unit of a CDC system
Digital Equipment Corp. Script Machine	PDP 11/20 *
Hewlett Packard TEPE	HP 2100
Honeywell CUESTA	H 6000 or Level 66 Series 60
Honeywell DATUS	Datanet 30
IBM DB/DC Driver	IBM 370/145 *
IBM TPNS	IBM 370/145 *
Univac CNE	Univac 1100 series
Univac CS1100	Univac 1100 series

* Minimal hardware on which the RTE may be configured.

Figure 5—Implementation hardware

ance Evaluator (TEPE). The Air Force/MITRE Design Verification Model (DVM) has been used for testing several different computers and operating systems: Burroughs 6700, Control Data Corporation 6600, IBM 370/155, Honeywell 6180 and 635 (under GCOS and Multics), and Univac 1108. DEC's Script Machine was originally designed to test the PDP 11/70, but there are no design constraints that would limit it to the testing of that system. HP's TEPE has been used strictly for the test of HP systems, but again it appears that it could be used to drive other computer systems.

Control Data Corporation's Benchmarking Approach to the RTE Requirement (BARTER) is an RTE which was originally designed as an internal driver, but was modified to run externally. BARTER runs in a Peripheral Processing Unit of a Cyber series computer.

Two RTE's are listed in Table 4 for Honeywell; the Communications User Emulated System for Traffic Analysis (CUESTA) is basically an extention of DATUS to provide expanded capabilities. CUESTA has been used in the testing of Honeywell systems under GCOS and Multics.

IBM has two RTE's: Data Base/Data Communications Driver (DB/DC Driver) and Teleprocessing Network Simulator (TPNS). DB/DC Driver was especially designed for data base application programs and supports 2741's and teletype-compatible devices. TPNS was designed to provide controlled generation of message traffic into a telecommunications subsystem or application and supports a large variety of IBM devices which communicate in a polled environment. Both of these RTE's are actively marketed by IBM.

Univac is another vendor which developed two RTE's. The Communications Network Emulator (CNE) is available strictly for use internal to the Univac benchmarking facility. Communications Simulator (CS 1100) is available to all Univac 1100 series installations at no charge.

All three of the minicomputer-based RTE's support asynchronous communications protocol, and the Air Force/MITRE DVM supports synchronous communications protocol. The Air Force/MITRE DVM supports 64 asynchronous devices and 64 synchronous devices; the Script Machine and TEPE both support 32 asynchronous devices.

Of the seven RTE's which are implemented on computers other than minicomputers, all support asynchronous communications; all except DATUS support a synchronous communications protocol. As far as the RTE software is concerned, the number of devices which can be emulated is generally in the hundreds. However, there is an interesting problem associated with the emulation of a large number of asynchronous lines (where "large" is defined as between 50 and 100). The difficulty appears to be more fiscal than technical. While there is no difficulty in configuring a system for installation at the customer's site, there is a problem with the test system at the vendor's benchmarking facility. There is a cost, on the order of $750 to $2,000, in configuring each line between the RTE and SUT. Non-multiplexed asynchronous terminals each require one line, thereby significantly impacting the cost of setting up an emulation.

RTE's emulate not only terminal devices but also the human operators of those devices. Therefore, the interaction between humans and teleprocessing systems must be modelled. Basically, there are two human characteristics which can be emulated. One is think time or the amount of a time a user delays or thinks between receiving a message from the SUT and entering the next user message. All RTE's provide script commands to emulate think time. Some RTE's allow think time to be changed for each user input, others don't allow changes within the script but allow a different global think time for each emulated device, and others allow the calculation of think time based on a probability distribution.

Another teleprocessing user feature is typing rate. Certainly typing rate varies from user to user dependent upon individual typing ability and upon user familiarity with the keyboard being employed. The RTE's which are implemented on large-scale computers do not have explicit script commands to regulate emulated user typing rate. Rather emulated user characters are transmitted to the SUT at the full speed of the transmission line. The only exception to this implementation is BARTER which has a typing rate command.

As mentioned earlier, application of a device in a procurement environment requires special attention to the area of integrity confirmation. All current RTE's employ at least one procedure which can be used to validate the performance of that device during the test. Every RTE has the capability of logging the characters exchanged between emulated devices and SUT. This log can be perused following a test to determine if the scripts were indeed executed properly.

SUMMARY

The trend in computer usage has been from on-site batch processing systems to systems which encompass on-site batch, remote batch, and interactive teleprocessing. Batch programs when timed by observers using stopwatches constituted adequate benchmarking methodology for on-site batch systems. Benchmarking of teleprocessing computer systems requires different techniques to evaluate the system quality of service. Remote terminal emulation is an approach to such evaluation.

Remote terminal emulation represents a technically valid approach to such testing and is currently available in the majority of vendor benchmarking facilities. This paper has discussed ten RTE's for testing teleprocessing computer systems. The intent of the discussions was to indicate the type of capability and capacity which these current implementations employ.

REFERENCES

1. *Guidelines for Benchmarking ADP Systems in the Competitive Procurement Environment,* Federal Information Processing Standards Publication 42, December 1975.
2. Bell, T. E., "Computer Performance Variability," *Proceedings National Computer Conference,* 1974, pp. 761–766.
3. Bell, T. E., *Computer Performance Management Through Control Limits,* TRW-SS-76-01, TRW Defense and Space Systems Group, January 1976.
4. Crothers, C. G., *Workload Determination and Representation for On-Line Computer Systems,* ESD-TR-74-54, The MITRE Corporation, Bedford, MA (NTIS AD-779 818), January 1974.
5. Abrams, M. D., S. Treu, and R. P. Blanc, *Measurement of Computer Communication Networks,* National Bureau of Standards Technical Note 908, July 1976.
6. Watkins, S. W. and M. D. Abrams, *Survey of Remote Terminal Emulators,* National Bureau of Standards Technical Note, in preparation.

Application of remote terminal emulation in the procurement process

by E. J. McFAUL
U. S. Geological Survey
Reston, Virginia

ABSTRACT

The procurement of communications-oriented computer systems in the Federal Government has recently embraced a technological concept which will significantly affect the trend of such activities for many years to come. This concept, generally referred to as Remote Terminal Emulation, provides the potential customer with a means of testing and evaluating a proposed system under loading conditions that closely approximate the intended live environment but without the logistical problems that have plagued such attempts in the past. This capability takes on added significance when viewed in the context of the federal procurement process which usually involves the execution of a benchmark by all participating vendors at some point in the procurement cycle.[1] In order to be equitable among all vendors, as well as provide meaningful results for subsequent evaluation, the benchmark must not only be representative of the anticipated workload but must also exhibit one very important characteristic—repeatability. Until recently, this latter requirement reduced most benchmarks to essentially batch-oriented exercises, with but a smattering of interactive processing that did little more than establish the functional capability. With the use of Remote Terminal Emulation, the situation has changed. Potential customers are now able to test and evaluate proposed systems with benchmarks that provide significant workload levels (benchmarks with over one hundred emulated terminals are not uncommon) such that meaningful evaluation data can be obtained. This paper documents the experience of the U. S. Geological Survey in using Remote Terminal Emulation during a recent procurement of a nationwide communications-oriented computer system.

INTRODUCTION

The recent energy-related problems of the United States have had a far-reaching impact on the scope of computer-based activities within the U. S. Geological Survey. In order to meet the myriad demands of both Government and industry concerning the management of our nation's energy resources, the USGS has had to dramatically expand its computer resources, particularly in the area of interactive and remote processing. The planning for this expansion culminated in May of 1975 in a set of user requirements that would ultimately serve as the basis for a solicitation document (RFP) in an industry-wide competitive procurement.

BENCHMARK OBJECTIVES

The Computer Center Division within the USGS was responsible for translating the stated user requirements into a complete RFP, as well as for designing a benchmark that would best evaluate a vendor's ability to meet those requirements. In order to accomplish the latter objective, the benchmark not only had to be truly representative of the projected user workload, but also had to provide an equitable test across all of the participating vendors. Also, the benchmark had to rigorously test all components of any proposed system, with all subsystems functioning as they would in a non-test environment.

EXISTING METHODOLOGIES

With the above objectives in mind, an investigation was begun into existing methodologies for the benchmarking of teleprocessing systems. The traditional method of employing a batch-oriented workload with a small number of live terminals was rejected because of the difficulty in equating the batch activity to a desired teleprocessing workload.[2] Large number of live terminals were unacceptable from a repeatability standpoint, not to mention the logistical nightmare that such an approach would entail. Simulation was not considered a feasible alternative due to Federal regulations restricting the use of this technique.[3] There was, however, one more approach to the problem that appeared to merit further investigation. Referred to as terminal emulation, this concept involved generating the required teleprocessing workload via software. However, unlike simulation which attempted to model the teleprocessing system (and was thus only as valid as the assumptions upon which the model was based), terminal emulation captured the actual teleprocessing dialog in software, with the additional capabilities of being able to control such parameters as think time, line speeds, and typing delays. At the time, two types of terminal emulators were being developed by industry—internal and external. The internal emulators (or measurement drivers) were typically run on the actual system under test (SUT) but, in doing so, perturbed the object of measurement.[4] This factor, together with reservations concerning the completeness of such a test (usually the complete communications subsystem was not exercised) resulted in the USGS insisting that any methodology employed must generate the required teleprocessing workload completely external to the SUT. The external or remote terminal emulators (RTEs) met this requirement in that they all employed separate computers to house the emulation software and entered the SUT via the normal communications interface. In addition to external generation, the USGS required that no portion of the SUT could be used in the workload generation, and that transmission of the workload to the SUT must use the vendor's standard hardware and software interfaces. All of these requirements were intended to insure that the installed system would need no hardware or software modifications in order to support a live teleprocessing workload equivalent to that emulated during the benchmark.

INDUSTRY RESPONSE

Upon release of the RFP to industry in August of 1975, it became clear that four major mainframe manufacturers could meet the benchmark requirements as stated, with two others needing only additional development time to fully comply. Unfortunately, the USGS procurement timetable could not afford the requested delays, resulting in three vendors (one of the four qualified vendors eventually chose not to bid) going to benchmark.

USGS RTE WORKLOAD

The teleprocessing workload to be emulated by the vendors was comprised of both asynchronous and synchronous activity. The asynchronous portion was to reflect a population of between 48 and 96 low speed (300 baud) terminals, depending on the particular point in the system's life being tested. Each terminal was to be actively engaged in one of four preassigned functional scenarios for the duration of the timed benchmark. The scenarios consisted of continuously-repeated sessions, with a session being initiated when the emulated terminal logged into the SUT and terminated when the emulated terminal logged out. Each session was individually identified with an incrementing two-digit field as part of the emulated dialog to aid in tracking the activity of the emulated terminals in subsequent data analyses. The four functional scenarios consisted of the following: (1) interactive FORTRAN compilation and execution; (2) file manipulation involving media transfer; (3) intra- and inter-file text editing; and (4) data base query generation and execution. Each of the four scenarios had a specified think time to be incorporated into the dialog as a fixed delay between the last character of the response transmitted from the SUT and the first character of the next stimulus transmitted from the RTE.

The synchronous portion of the workload consisted of 12 to 28 emulated remote-job-entry terminals operating at line speeds of between 1200 and 9600 baud. The scenarios of the RJE terminals were required to function in continuous job-submission mode, with specified delay times incorporated between individual job submissions. Depending on the point in the system's life being tested, from four to seven functionally distinct batch programs were required to be transmitted to the SUT and accumulated into input job queues. Once sufficient numbers of these batch jobs were queued, one of each of the different batch jobs was set into execution. This set of multiprogramming batch jobs formed a continuously-replenished batch background running concurrent with the aforementioned asynchronous and synchronous activity. A graphical representation of the combined SUT workload over the system's life is presented in Figure 1.

RTE COMPARISONS

The three vendors who participated in the final benchmark employed functionally similar RTEs. The general benchmark configuration is indicated in Figure 2. All vendors chose to implement the synchronous emulation by using separate physical communications cables between the RTE and SUT for each emulated RJE terminal. However, for the asynchronous workload, the vendors all elected to

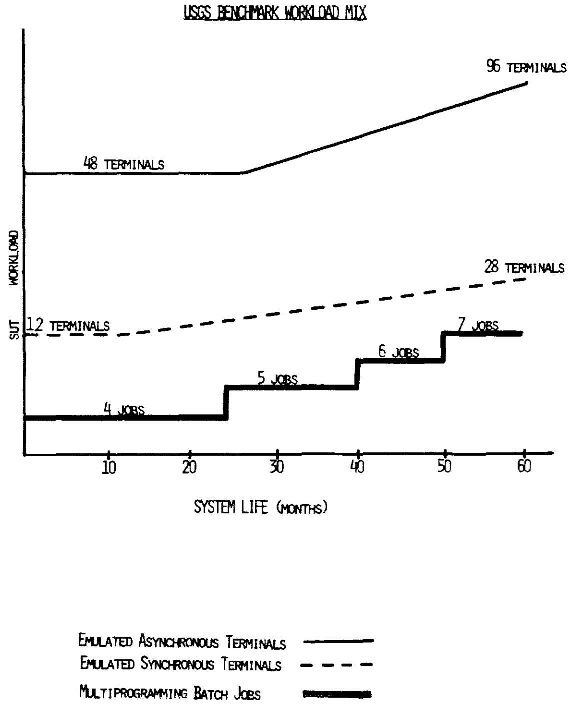


Figure 1—USGS benchmark workload mix

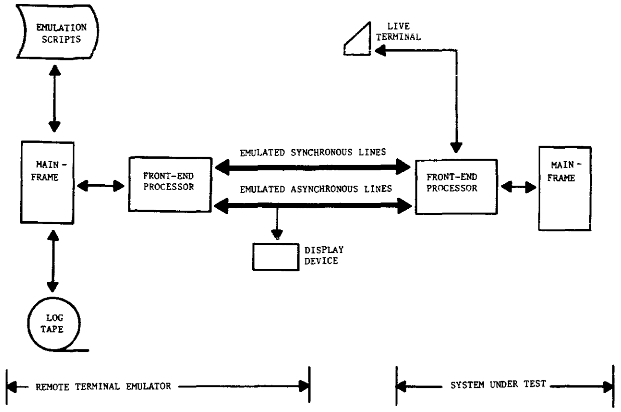


Figure 2—General benchmark configuration

emulate multi-dropped terminals in order to reduce cabling requirements. In the area of verification of the emulated terminal activity, several observations are in order. There existed sufficient capability common to all three vendors to provide a reasonable degree of assurance that the benchmarks were being performed properly. All vendors could display the status of any selected emulated line using the console on the SUT. All vendors could capture and time-tag all emulated terminal traffic and store it on a magnetic tape for post processing. And all vendors could display the dialog on any selected emulated asynchronous line on an output device. The vendors differed, however, in the degree to which each of these capabilities had been refined. For example, while one vendor could dynamically change the selected asynchronous line to be displayed by simply entering a command on the RTE console, other vendors had to either physically move a pair of connectors on a back panel or actually make minor modifications to the RTE software. Although all vendors could display the status of any selected emulated line (i.e., active, inactive, etc.), only one vendor could display the actual position of RTE execution within any given terminal session. While some vendors were able to automatically time-tag any displayed asynchronous terminal sessions, other vendors required manual annotation on corresponding output listings.

BENCHMARK CONSTRAINTS

The salient features of the USGS benchmark can be described as follows:

(1) The vendor was allowed up to two augmentations (hardware and/or software changes) over the 60-month system's life;
(2) The vendor had to benchmark the month-60 workload plus all augmentation points;
(3) A live interactive terminal with a priority identical to that of the emulated asynchronous terminals was used to measure various command response times. The resulting averages could not exceed stated maximums;
(4) The transmission times associated with the emulated synchronous activity could not exceed stated maximums;
(5) The elapsed times of batch jobs could not exceed stated maximums;
(6) All workload types must have been concurrently executing when any timing measurements were made; and
(7) For acceptance-test purposes, a "batch-only" version of the workload would be run and timed.

BENCHMARK PROCEDURES

The procedural implementation of Remote Terminal Emulation within the context of the overall USGS benchmark can be seen through the following event descriptions. At the commencement of the live benchmark run, the system configuration was verified by the USGS benchmark team. Following this, the emulated asynchronous terminals were allowed to begin logging into the SUT. At approximately 30 minutes into the benchmark, the emulated synchronous terminals were released to begin transmitting batch jobs into the SUT input queues. After approximately 50 minutes, when all of the emulated terminals had established communications and reached steady state, a record was made of the current SUT job queues. Following this, the required mix of multiprogramming jobs was set into execution. Once all portions of the benchmark workload were being processed, the first set of live-terminal timings were taken. At approximate 10-minute intervals, the remaining sets of timings were taken until six complete sets were recorded. As soon as the timings were completed and at least one copy of every type of batch job had finished execution, the benchmark was terminated. Ending job queues were recorded for subsequent quantitative analysis of the processed workload.

OBSERVATIONS

At the conclusion of the benchmark associated with the USGS procurement, the following observations were made:

(1) All vendors were eventually given over six months to prepare the benchmark; nevertheless, every vendor attempted major hardware and/or software modifications during the week of the live benchmark test;
(2) All vendors ultimately made some modifications to their RTEs to conform to the USGS requirements or to correct uncovered faults;
(3) All vendors suffered significant SUT hardware failures during the live benchmark tests;
(4) Most vendors were plagued by RTE hardware errors or unexpected communication problems associated with the RTE-SUT interface; and
(5) All vendors found that traditional "on-the-fly" tuning

and reconfiguring produced somewhat unpredictable results.

CONCLUSIONS

In retrospect, the USGS feels that the efforts required to incorporate the technique of Remote Terminal Emulation into a major computer-system procurement were well justified in terms of the resulting level of confidence in the capabilities of the acquired system. It was not an inexpensive undertaking for either the Government or the participating vendors. Major competitive procurements never are. However, when one considers the overall cost-effectiveness of having obtained the lowest-priced system that has actually been tested against its ultimate projected workload, the real merits of Remote Terminal Emulation begin to become evident. And as this tool is refined and standardized through continued Government-industry cooperation, it will surely become an inseparable part of all future communications-oriented computer procurements.

REFERENCES

1. Federal Property Management Regulations, Subpart 101-32.404-1, "Restrictions on the Use of Simulation to Describe Data Processing Requirements," revised by *The Federal Register,* page 43548, October 1, 1976.
2. Hyman, B., "Stability and Workload Definition for Time Sharing Systems," Federal Information Processing Standards Coordinating and Advisory Committee, Task Group 13, July 30, 1975.
3. Federal Procurement Regulations, Subpart 1-4.1107-5, "Restrictions on the Use of Simulation in the ADPE Procurement Process," revised by *The Federal Register,* page 43538, October 1, 1976.
4. FIPS PUB, 1977, *Guidelines for the Measurement of Interactive Computer Service Throughput, Turnaround Time and Response Time,* Federal Information Processing Standards Publication, to be published in 1977.

Remote terminal emulator development and application criteria

by C. T. ARTHUR
Honeywell Information Systems
Phoenix, Arizona

ABSTRACT

This paper is a general discussion of remote terminal emulation; its history, processes, implementation techniques, and application criteria. The remote terminal emulator is a measurement tool, and as such, requires a functional understanding of its operation for proper application. Too often the emulator has been applied as an external loading device without the user comprehending that it is also a network simulator operating under the constraints of the communication disciplines and limitations.

This paper does not attempt to establish guidelines or rules respective to the emulator's usage, but rather, discusses areas which should be investigated and analyzed prior to the experiment definition.

INTRODUCTION

The intent of this paper is to reflect on how we, as developers of a remote terminal emulator (RTE) system, view its evolution, implementation, purpose and usage. Like any technology that has been developed from many independent sources, there are many areas of agreement and disagreement in methodology and use.

The development of these systems is discussed in general terms, realizing that the hardware will impose implementation concept variations between developers. When specific functions and capabilities are mentioned, the system cited as an example will be, for most part, Honeywell's current Communication User Emulation System for Traffic Analysis (CUESTA), although no formal presentation of CUESTA is intended or provided.

The remote terminal emulator is regarded by its developers as a measurement tool, and as such, a functional understanding of its operations, capabilities, and limitations is necessary for its proper application. Thus the intent of this document is to not only discuss the RTE application criteria, but also present the basic concepts of the emulation process, the communication network, and the functions of the script. In addition, the various data reductions techniques and the current RTE usage are mentioned.

BACKGROUND

Although not new, remote terminal emulation is also not old. Programs were being developed under different nomenclature in the late 1960's which, by current definitions, were true remote terminal emulators. Previously, many specialized programs were written by system developers in order to validate their communication software.

The early versions developed by Honeywell, were called User Simulators, User Exercisers and External Load Generators. However, these nomenclatures only added confusion to an already confusing area, because at that time, internal user simulators, exercisers and load-generators were also being developed and used. In the use of one Honeywell system, DATUS (DAta Terminal User Simulator), an attempt was made to change its name because of the suspicions raised by the word "simulator."

By 1969 Honeywell had three separate projects developing terminal emulators. Two were oriented toward performance testing and the third was oriented toward functional testing. All were implemented in small communication Front eNd Processors (FNP), which were memory limited, and confined to the teletype disciplines. In 1970, the two performance emulators were merged into one and called DATUS.

Also in 1970, a fourth system was developed specifically for exercising the Honeywell H635 system with Remote Job Entry (RJE). Later to be known as the Remote Computer Simulator, this too, was implemented in a small FNP. Consequently, it was limited to "broadcasting" the input job streams over a number of lines simultaneously. This was one of the first attempts to emulate a complex high speed (9600 baud), synchronous device.

By late 1971, another project was begun which was to utilize the entire GCOS/H66 system to emulate users. It was during this project that the GCOS* operating system's self-protecting features were found to be too restrictive for a true emulation process. In 1972, the project was altered to develop special RTE front end software which would be compatible with the GCOS operating system software and

* GCOS is the acronym for General Comprehensive Operating Supervisor for Honeywell's H66 systems. GCOS is a trademark of Honeywell.

also allow typical GCOS application programs to control the communication line interface, and thus define their own disciplines. The project was later named CUESTA.

Because of the limitations experienced with the standard GCOS communication interface, the initial CUESTA effort was to develop only the FNP communication software. At this point in time, little effort was made to define the structure of the H66 program. As it turned out, this was beneficial to the developers as it allowed them to gain the experience needed to construct a sound executive for data handling which, in concept, could be moved to the H66 as a mini-executive, operating under GCOS software, and be open-ended with respect to the emulation capabilities. Thus, CUESTA became an emulation executive system which allows the RTE user to define his own terminal disciplines and dialog through application of the "script" and FNP parameters.

THE EMULATION PROCESS

As mentioned above, the remote terminal emulator is in itself a tool and not an application. It provides a means for the RTE user to simulate an operator at a terminal and then emulate that terminal at the communication link interface, so that to the System Under Test (SUT), the interface appears to be the actual terminal.

Even so, depending upon the methods and hardware in which the remote terminal emulator is implemented, the emulation process may not be exact. It must be remembered that the communication interface being utilized is a computer/computer and not a terminal/computer. While the emulating software attempts to be exact, it is governed by the processor's resources available for program execution instead of hardware control as in the terminal. This can lead to small timing differences which would not be discernible with low-volume, small-network configurations, but they may become magnified to noticeable levels when operating with high-volume, large-networks. Whether these timing variations are of any consequence depends upon the application of the RTE experiment and its objectives.

Early in the development of remote terminal emulators, the primary design objective was to automate the mechanics being used in the engineering testing and checkout procedures and to enlarge these procedures to provide heavier workloads to the SUT. The communication network size was minimal by today's standards. RTE support capabilities were "after thoughts" and the script equivalent was an integral part of the software coding. The communication processor was utilized as a stand-alone system, because that was where the communication hardware resided.

As the technology evolved, more optimistic and demanding objectives were defined. No longer was "keeping up with the SUT" acceptable, but now it must be overpowered. Emulation of large networks with many differing terminal types, being used in many differing applications, was desired. As the complexity grew, it became apparent that script generation must be automated and independent of the network configuration and the number of users being emulated.

Through the evolution process, many kinds of RTE implementation techniques were studied and evaluated. It became readily apparent that they could be categorized into four basic groups depending on their line capacity, terminal driving and executive control capabilities.

1. Single line, Single driver, Single control—Probably the first version of a RTE, this was the quick test program that was capable of appearing like a single terminal. The script equivalent was encoded and imbedded directly into the program and its flow.
2. Multiple lines, Single driver, Single control—This program type emulates one terminal type, utilizing a single script (or equivalent) and a single control process. Instead of communicating on just one line, the data is "broadcast" to a number of lines in parallel. The received data may be verified in any number of ways, but the program anticipates that all the lines will receive the same data. The Remote Computer Simulator mentioned above, used this technique.
3. Multiple lines, Multiple driver, Multiple control—This technique was quite popular in the late 1960's. It is essentially the collection of a number of Single-line, Single-driver, Single-control program segments, bundled under one mini-control routine. Although this method was not utilized to a great extent to emulate different terminal types, it was used to provide scripts for different concurrent applications.
4. Multiple lines, Multiple driver, Single control—In this type of programming, there is a single executive capability which provides control for all the overhead functions and common emulation processes. The script or script segments are oriented toward the application processes in the SUT and they are capable of emulating any number of users. The script may utilize common data and is still capable of accessing a user/terminal selectable data base. CUESTA falls into this category.

The terminal/user emulation process is composed of two functions; operator simulation and terminal emulation. The division between these two is not always obvious as it is dependent upon the hardware in which the remote terminal emulator is resident and its method of implementation. The terminal emulation process can be further subdivided into control logic simulation and line handling. For teletypes, the control simulation is almost non-existent as it is inherently done in the line interface hardware. The balance is then relegated to the operator. However, for the more complex terminals such as remote computers and synchronous CRT's, the data formatting logic and protocol control can become quite involved. In the case of the remote computer (R/C) not only must the hardware functionality be simulated, but also the functions of the R/C software.

Line handling is the second aspect of terminal emulation. In CUESTA, we have defined four modes of line handling, applicable to both synchronous and asynchronous lines. We have found the terms "full duplex" and "half duplex" to be incomplete, and have defined line modes as follows:

1. two wire, alternate,
2. four wire, alternate,
3. two wire, simultaneous,
4. four wire, simultaneous.

The alternate mode provides a gate in the line handler which forces the line to be cycled through the send/receive sequences. While this may appear self-limiting for the general case, it does relieve some simulation control processes of many timing considerations. The simultaneous mode keeps the receive mode active at all times and allows simultaneous transmitting. The wire mode selects the appropriate data-set or modem controller.

Operator simulation is normally observed to be the controlling of the data entry process and scanning of the received data (interplay between the operator and terminal). This is only one facet. Much more subtle is the interplay between the operator and the SUT and its application program. Depending upon the SUT's application program and its communication facilities, the SUT's software may be designed to take advantage of the "slow" human response. Unless the RTE is properly governed for that application, timing and synchronization problems may develop between the two. It can be successfully argued that this condition arises only if the emulation is not realistic. However, it has been our experience that many non-realistic experiments are undertaken in order to "see what the SUT can do."

Another condition which must be considered, is the proper detection of the end of transmission from the SUT. Although the ASCII character set provides the capability for end of text, end of block, and end of transmission, etc., not all of the SUT's application software will consistently use them. Instead, other techniques, including operator intuition, have been devised to signal the operator that the SUT is finished with its transmission and is now available for operator inputting. Thus, the detection of key words or phrases in the latter portions of the text (commonly called prompts) becomes exceedingly important if the proper synchronization is to be maintained between the SUT and RTE.

THE SCRIPT

In remote terminal emulation, Script is the notation given to that portion of the software that defines the simulation of the operator. It may be a segregated, coherent, logical entity, or it may be integrated into the overall implementation in such a fashion as to be indistinguishable. The most obvious objective of the script is to specify the data exchanged between the RTE and the SUT. In order to accomplish the operator simulation, the script must provide or invoke a sequential series of operations that will result in some coherent, logical function.

Generation of the script can be accomplished by various means. In emulators where it is an identifiable entity, it is typically a separate program module or subroutine. The ease in which this module can be coded is dependent upon the implementation. In some RTEs, the script is coded in machine language; in others, in a high level language such as FORTRAN; while the balance are written in a special language specific for the emulator. CUESTA has utilized the latter in that we developed a specialized compiler which generates the appropriate assembly language source code. Thus, the result is compatible with the assembly process, but will still allow the user to develop his script without concern to the mechanical details of formatting and exchanging the data.

Other terms have been given to the script which either enlarge its meaning or define more specifically the associated detail components. To date, no attempts have been made to rigidly define these terms and as a consequence, many ambiguities have resulted. In CUESTA, the terms in Table I have been defined, but they are inherently associated with the implementation.

With CUESTA, the operator dialog is normally defined by script statements generated by the CUESTA User. These statements, which are "compiled" by a special Conversion Map Generator, are not limited to data transfer operations, but also include the capability for decision making, data base handling, transition control definitions, on-line statistical summaries, and special terminal operations. In addition, the experimenter is given the capability to define his own statement(s) and the associated processor.

It is in the Conversation Map (via the script statements) that the experimenter defines the operator characteristics. These include such items as think-time, data-entry-rate, and transaction rate capability. The characteristics may be defined for specific user groups, the individual user or for individual operation type script statements.

The Conversation Map is also used to specify task transitions. A task is normally defined to be those operations required to simulate a given logical function or application (file building, edit, compile, etc.). The transition control is that process which specifies which task is to be executed upon the completion of the previous task in order

TABLE I—Script Terminology

Task Step Processor—A software routine that performs certain predefined functions.

Task Step—That association of operands to a Task Step Processor which normally is defined by a Script Statement.

Task—Any combinations of Task Steps that result in some logical function (i.e.; a user session).

Script—A collection of one or more independent Tasks.

Scenario—A collection of Tasks which are dependent upon one another to emulate a high order function (i.e.; remote computer).

Script Statement—The statement generated by the script writer to specify the emulation operations, operator characteristics, transition probabilities, on-line summary operations, etc..

Conversation Map—Sometimes referred to as the Map or Script, this is the program module generated by the Conversation Map Generator. The generator utilizes the script statements and a library as its source of input.

to maintain a coherent continuation of the emulation process.

CUESTA currently has three modes of transition control;

1. Random Control
2. Profile Control
3. Initiation Control

Random Control is a process which randomly selects the next task to be executed from a table of probabilistic occurrences. The probabilities are specified by script statements, which in turn, are used to generate a $n \times n$ selection matrix. The x/y axes of this matrix can then be utilized to establish the desired from/to transition relationships.

The Profile and Initiation Control modes are based on the assumption that transition from any task specified within a given group, can be coherently accomplished to another task in that same group. It should be noted that if a given task is compatible for more than one group, it need not be recoded for each group.

Profile Control selects the next task via a dynamic mathematical process based upon the number of users currently executing specific tasks and the desired probability of that number. Initiation Control is similar to Profile Control, except that the calculations are based upon the current number of tasks that have been executed with respect to the desired probability. Thus, the selection process forces the emulated user into execution of a task which currently possesses the smallest ratio of executions with respect to its desired percentage. The latter two methods were implemented upon information gained from a paper by B. Hyman.**

When the script segment is written, it is normally divided into tasks. The segment is written for a single type of user executing a specific application with the SUT. At this point, the CUESTA user does not have to concern himself with the number of users that will be emulated by this script segment (the user count will be specified later with the network). If differing and completely independent applications are desired for other users, additional segments are appended to the existing script and kept segregated by the transition probabilities. This segregation is originated by defining which user will start with what task or application.

As mentioned above, should the standard script statements prove to be insufficient for the experimenter, he may define his own. In addition, should he desire additional control over the execution of the simulation process, the Conversation Map has the capability to allow him to design his own CUESTA Control Commands.

A problem which has been superficially addressed, is "How does one specify an emulation exercise in terms that are applicable to all the various RTE systems, that can define the network, user sessions and operator characteristics?" Currently, the English language is being used with sprinklings of examples which are normally based on a particular vendor's system. This in turn, leads to occasional unnecessary script translations, interpretations, and misunderstandings. Other methods have been suggested, but none have been exploited to determine the relative merits and limitations. Table II depicts a simple CUESTA script "description" for an emulated user who will build, list, change, and compile a FORTRAN file. This is not only an incomplete CUESTA script, as many details are omitted, but it may not be definitive enough for another RTE system.

THE COMMUNICATION NETWORK

The communication network, as it is used in remote terminal emulation, is composed of two elements; the live network and the simulated network. Referring to Figure 1, the live network consists of the actual data lines that connect the RTE to the SUT, while the simulated network resides totally in the RTE and is the balance required for the emulation exercise. Exactly how these two components are specified, depends upon the RTE's implementation and capabilities. In any case, the configuration parameters in the SUT and RTE are inter-related. The SUT specifies what the network should be, while the RTE specifies what the network actually is.

Since CUESTA utilizes a front end processor, it is via the FNP parameters that the live network configuration and characteristics are defined. These parameters include such

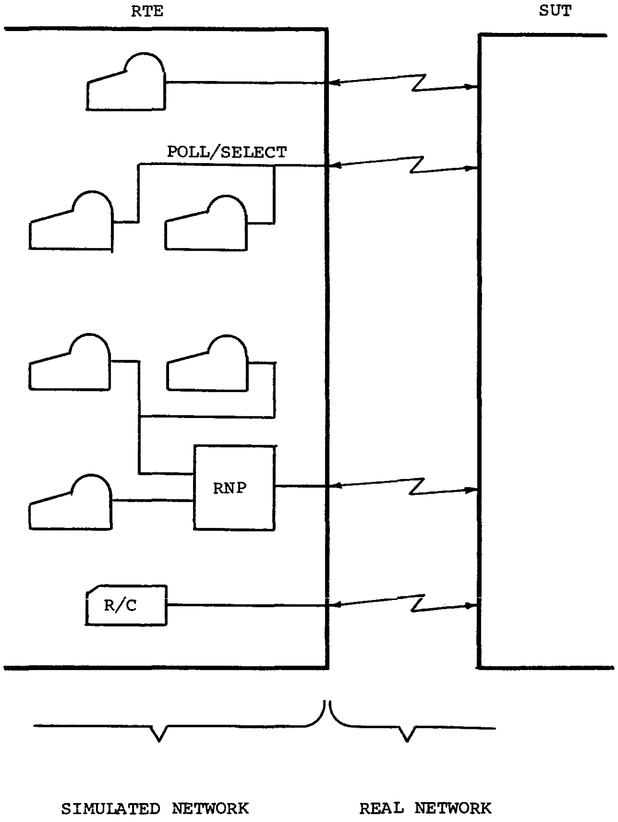


Figure 1—RTE communication network segments

** Hyman, B., *Stability and Workload Definitions for Time Sharing Systems*, Bell Laboratories, 1975.

TABLE II—Sample Script Description

TASK NAME	TASK DESCRIPTION
LOGON	set user characteristics transmit and receive identification sequence select the FORTRAN subsystem
BUILD	build a FORTRAN source file
LIST	list the file
COMPILE	compile the file
CHANGE	change source statements in the file

TASK SELECTION PROBABILITIES

Start task = LOGON
Next task selection:

from LOGON	goto BUILD/100%
from BUILD	goto LIST/50%, COMPILE/25%, CHANGE/25%
from LIST	goto COMPILE/50%, CHANGE/30%, LIST/10%, BUILD/10%
from COMPILE	goto CHANGE/40%, LIST/20%, BUILD/30%, COMPILE/10%
from CHANGE	goto LIST/70%, COMPILE/10%, BUILD/20%

items as line assignment, line type, speed, etc. The H66 program parameters specify the simulated network. This is accomplished by defining table structures that will allow the various emulation modules to interact with each other (and the script) so as to simulate multi-drop lines, remote concentration, message routing, etc..

TEST CONFIGURATIONS

The system under test is normally envisioned to be the central computer system, and the remote terminal emulator to be a collection of terminals. Thus, the most common test configuration is where the RTE is connected directly to the central computer. However, as shown in Figure 2, other RTE configurations are readily possible, such as the case of remote concentrator or Remote Network Processor (RNP) loading and the driving of multiple independent SUTs. An interesting observation in the case of RNP loading, is that the system(s) under test can be defined based on one's point of view. Is the RNP a SUT or just a communication facility?

Also as shown in Figure 2, a system being driven by a RTE is not precluded from being loaded by real terminals. In fact, this may be a most desirable configuration as it provides the experimenter with an on-line "feeling" of the effect of the RTE's workload upon the SUT.

APPLICATION

The definition of the application or experiment, in general, is not a trivial task. In most cases, research and measurements should be done with a current system, and extrapolated if necessary, in order to determine how the communication network will be utilized and what kind of terminals/operator characteristics will be incorporated. In all cases, the experiment's objective will be to measure the system under test with respect to some loading capacity and/or capability. Many benchmarks have been conjured on the premise that the SUT's central processor is to be the only component monitored and the communication environment is only a means to provide the load. However, it must be remembered, that the SUT's communication facilities are designed to be an integral part of the total system and its capabilities and limitations with respect to the network must be carefully considered. This is not meant to say that the network specified should be the one that will be

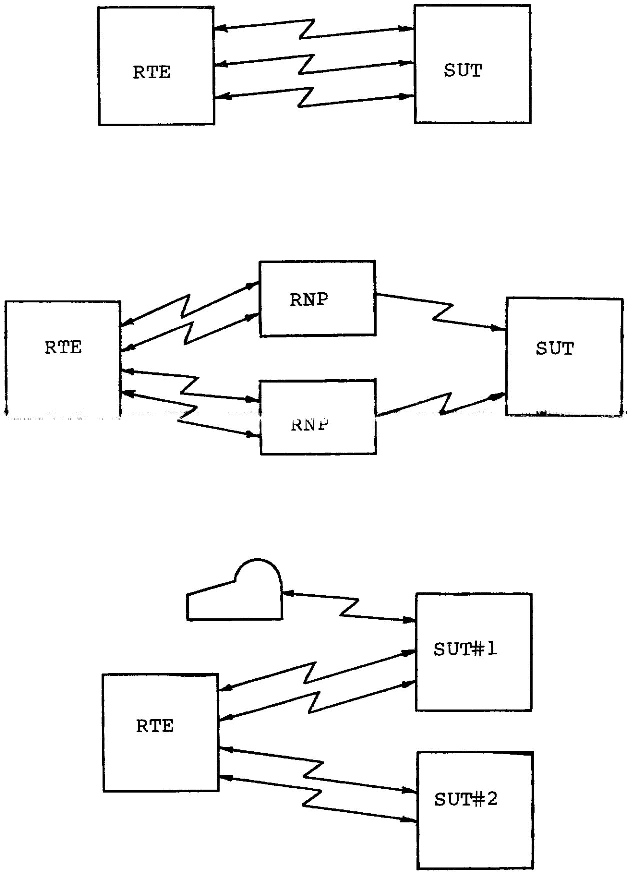


Figure 2—RTE test configuration

utilized in the real-world, but it must be consistent with the real-world operations.

During the processes of RTE development we have observed two modes of measuring or testing that utilize RTEs; functional testing and performance testing. These modes can be visualized as being at opposite ends of a spectrum, with all degrees of mode combinations in between. The actual degree that will be found in any one application depends upon the capabilities of the remote terminal emulator itself. This is not as obvious today as it was earlier in the RTE development phases, because the current versions are being implemented in more powerful systems and the hardware limitations imposed previously are no longer applicable. Consequently, applications that incorporate both modes of testing are becoming common.

Functional testing is that mode which has as its primary objective, the requirement to perform detailed validation on all data and terminal control functions. This requires the RTE to maintain a relatively large data base for comparing the data received from the SUT and sufficient RTE system resources must also be available in order to maintain a constant monitor of all error conditions. Since this mode is used primarily to test the functional capabilities of the SUT, high volume throughput and critical response time measurements are not required. Network sizing is normally kept to a minimum with only a sufficient number of terminals being emulated as to effectively utilize the SUT and RTE.

On the other hand, performance testing is not as concerned with the detailed validation, but concentrates on the methods to effectively provide a system workload. Data validation is normally limited to testing for "prompts" from the SUT and providing the operator input. Response timings and throughput are measured as the system loading and network configurations are altered.

Insofar as the remote terminal emulator is concerned, the primary difference between these two modes is its capability to access large data bases and its means of controlling the data links. If the data can be stored in quick access devices and the central system has an efficient means to compare the SUT's data against that data base, then for most part, the RTE is capable of performing both types of testing with one script. However, if the RTE is limited by data storage or manipulatable communication facilities, then it should be used with only one mode in mind.

DATA CAPTURE AND SUMMARIZATION

An aspect often deemphasized during the experiment definition phase is the availability of proper data reduction capabilities. If data reduction is to be facilitated, this implies that the RTE has the capability to capture data in such a form as to make the reduction process feasible.

Visibility has always been a problem when emulating a large number of users. One cannot dynamically display all the data being interchanged between all users at one time. Even the detection of "user hung" (because of an unexpected response from the SUT) is not always a trivial task and this problem is grossly compounded when the user count becomes large. For this reason, CUESTA has been designed to provide two modes of data capture.

Like most RTEs, it is capable of logging all incoming and outgoing data, with the associated control functions and time stamps, on magnetic tape. In addition, the capability is provided to the experimenter to log any information he may deem advisable via the Conversation Map.

After the experiment has been terminated, the log tape(s) can then be summarized by an off-line data summarization program. A number of programs have been developed for CUESTA that will summarize response and think times for the individual emulated user or user groups, display complete conversations, task executions, etc. In addition, the experimenter is not precluded from developing his own special program.

The second mode of data capture is by a dynamic statistical summarization, which is also provided via the script. We have observed that the bulk of measurement taken during an RTE experiment can be expressed in terms of some time increment. Thus, script statements were developed to capture selected times (send, receive, time of day), and buffer them until they can be summarized against a later time. The results can then be accumulated into "summation buffers" which are identifiable by name. The CUESTA executive will, upon request, print the contents of these summation buffers. The information normally listed is number of occurrences, average, minimum, maximum, and standard deviation for each buffer. We have found that utilizing this capability is very beneficial in monitoring the total experiment in that, with properly placed capture and summing statements, the percentage of script task execution can be observed along with the stabilization process via standard deviations. In addition, the timings to be monitored are readily available.

RTE USAGE

Currently, the primary usage of remote terminal emulator has been within our own Engineering Group and the Benchmark Support Group. The remote terminal emulators were originally developed as an engineering tool, but a large amount of the usage has been in the arena of benchmarks. Actually, the initial exposure of CUESTA was done during benchmark preparations. As stated earlier, the RTE was developed for either functional testing or performance testing. As the capability has evolved, we have seen the functional testing aspect enlarged to incorporate performance testing, rather than the reverse. This is probably because with the current system, the primary changes are in the reconfiguring of the network rather than redesigning the script.

Today, Engineering's primary use of RTEs is for new product testing and regression testing. As the SUT's capabilities are expanded and enhanced, the effects of those changes with respect to throughput and response timings are of concern. Also, the system software must be subjected to all conditions, in order to determine that the original functionality has not been unknowingly altered. Thus, both types of test modes are being used.

A supplementary engineering procedure which is gaining in popularity, is augmenting the RTE experiment with the capabilities of an external hardware monitor. By properly configuring the hardware monitor to the SUT, the monitor can be more effectively utilized to measure the SUT's program execution and resource activity under the precise controlled loading that the RTE is capable of delivering.

The test mode used by the Benchmark Support Group is exclusively performance testing. In this environment, very specific, highly specialized scripts are required. The network sizing and configuration are constantly varying from benchmark to benchmark, and in some cases, within the same benchmark. Large network configurations are becoming the rule rather than the exception.

In some instances, when a large network is to be installed, a ''pseudo-simulation'' is done for the network in order to reduce the data base in both the SUT and RTE. This technique, which has become quite common, is normally accomplished by having a single or small group of terminals simulate the workload for a larger group of terminals when using the multi-drop line protocol. Whether or not this technique is viable, depends upon the objectives of the benchmark and the analysis that has been done with respect to the line's protocol.

SPECIAL CONSIDERATIONS

As each vendor develops his own emulator, he will undoubtedly tailor it to meet some special requirements specific to his own hardware. For example, the Honeywell DN6600 has the capability to remotely operate its Time Division Multiplexer (TDM), which is capable of handling 52 users operating at 110 baud. In order to alleviate the logistical problems of locating and connecting these TDMs, CUESTA was designed to emulate the TDM so that the 52 users could in turn, be emulated on a single full duplex communication link. As a consequence, this feature is only applicable when the SUT is another Honeywell H66/DN6600. Other areas of consideration would be the methods used to interconnect the SUT to the RTE (i.e.; the use of modems and datasets versus direct connect cabling).

Capabilities for debugging of the script and its application must be provided. Again, depending upon the hardware's features, some manual intervention must be supplied to the experimenter. To date, CUESTA has incorporated over sixty-five system control commands for system and user status, script intervention and recovery/restart, user tracing, user reconfiguration, and general system controls in order to maintain this capability.

CONCLUSIONS

Previously, the bulk of all communications was done via the teletype disciplines and the limited use of computer to computer communications was almost exclusively relegated to the specialized tasks of remote job entry and bulk data transfers. Now with the advent of the microprocessor, intelligent terminals, and sophisticated line protocols, this environment is quickly accelerating into a highly complex world. These changes are not only placing greater demands for the development of new system capabilities, but they are also being reflected into the remote terminal emulator development. In order to ''keep up,'' the current RTE systems are being continuously modified and new systems are being developed. Remote terminal emulator development has historically lagged behind the system software and hardware development, but hopefully, that time can be reduced now that emulation concepts have been established and accepted.

The one thing that will not change will be the responsibility of the RTE user to objectively define his experiments in terms of operator characteristics, terminal characteristics, and network configurations. With the multitude of new communication devices being offered, this function will not be diminished, but will become of more concern (and more complex) than ever before, both to the user and the vendor alike.

A survey of structured programming practice

by I. ST. J. HUGO
Infotech International
Berkshire, England

ABSTRACT

This paper refers to the results of our survey of over 300 companies worldwide to determine the extent and type of their involvement in structured programming.

INTRODUCTION

Reason for survey

In November 1974 and March 1975, Infotech staged a total of two conferences and one tutorial on structured programming. Many of the speakers were well-known, e.g., E. W. Dijkstra, C. A. R. Hoare, D. Parnas, F. T. Baker, and M. Jackson, but nevertheless the total attendance of some 800 people from 19 different countries exceeded expectations. Because of this high degree of interest in the subject, a program of training courses was launched in mid-1975.

At the beginning of 1976, it was decided to survey the use of structured programming techniques with the following goals in mind.

1. To assess how well the program of training courses fitted the current needs and plans of software development groups.
2. To extend Infotech's own information on the subject.
3. To develop a useful, saleable product.

A note on Infotech

A note on the Infotech organization may help to put some of the remarks in this paper in context. Infotech is engaged in the education and training, primarily of experienced personnel, in data processing and computer-related subjects. The company is independent of any software house, hardware manufacturing or consultancy interest. It differs from many other organizations engaged in similar activities in that, while it has its own in-house expertise in data processing, this expertise is used only in the specification and monitoring of conferences and courses; all lecturers and the material they present are selected from appropriate organizations outside Infotech.

A note on terminology

The term "structured programming" has acquired many meanings that often lead to confusion. Unless otherwise stated, the term is used in this paper in its most general, but not abstract, sense. That is, it denotes a range of design, coding, documentation and management techniques that use structure in some way to impose greater clarity and control on the software development process. These techniques have become popularly known collectively as structured programming and many are often wrongly attributed to Professor E. W. Dijkstra. Where appropriate, specific techniques are mentioned by name.

The published survey

This paper is concerned with the analysis of the completed questionnaires returned by computer users, on the follow-up to these questionnaires (see Survey Method below) and on the case studies obtained for the published Survey and Report. The complete published work is based additionally on an analysis of 690 documents, papers and reports in the existing literature. The full Survey and Report consists of the following five volumes:

Survey

Volume 1: International Overview
Volume 2: Survey and Analysis of User Experience
Volume 3: Methodologies and Techniques

Report

Volume 1: Guide to Techniques and Implementation
Volume 2: User Experience in Structured Programming

SURVEY METHOD

The survey project was publicied in advertisements and mailing shots, computer users being invited to provide brief data on the extent of their organization's involvement in structured programming and to request a detailed questionnaire. The motivation for participation in the survey was that each organization completing and returning a detailed questionnaire would receive a copy of the analysis of the questionnaire returns. A total of 1080 organizations provided brief data and requested detailed questionnaires. The request form and detailed questionnaire are shown in Appendix 1.

The total of 1080 detailed questionnaires requested was reduced to 309 usefully completed and returned, which form the basis of the questionnaire survey, by the following occurrences:

1. Some of the questionnaires requested were not returned and no explanation was given.
2. Many of the organizations who requested questionnaires were in fact planning to use structured programming techniques rather than actually using them and so were unable to complete the questionnaire.
3. Miscellaneous reasons that rendered returns irrelevant to the purposes of the survey; e.g., inconsistencies in the data, jokers, etc.

On the basis of the adequately completed questionnaires, 54 organizations were selected and contacted by telephone for clarification and expansion of the reported data. A further 22 organizations were visited by Infotech staff in person, again to achieve greater understanding of the use being made of structured programming and of the results obtained. Finally, 13 specific case studies were commissioned and rights to a further existing 19 case studies acquired.

SURVEY RESULTS

Profile of responders

The questions in section A of the questionnaire (see Appendix 1) were designed to provide a profile of the responders. Figure 1 gives the breakdown of responding organizations by size of software department.

The geographical spread of responders was very wide, although few responses were obtained from most countries. Approximately one third of the responding organizations were in the USA, a quarter in the UK and a further quarter from 15 countries in continental Europe. The remaining responses were from Australia, Brazil, Canada, Japan, the Middle East and South Africa.

Twenty-seven percent of the organizations placed themselves in the software house category. However, slightly over half of these were software departments within user companies that had been established as a service division independent of other functional divisions within their organization. Allowing for this, 87 percent of responders were from organizations whose primary business was not data processing.

This figure is surprising in that organizations whose primary business is not data processing account for only about half of the usual participants in Infotech activities. True, responses to advertisement could have produced a slight bias outside Infotech's normal profile of participants but hardly a bias of that size. However, it is apparent, in the UK at least, that the primary interest in the use of structured programming lies with the computer users outside the DP industry. The large majority of software suppliers show more interest in selling aids, courses, etc., than in using the techniques themselves.

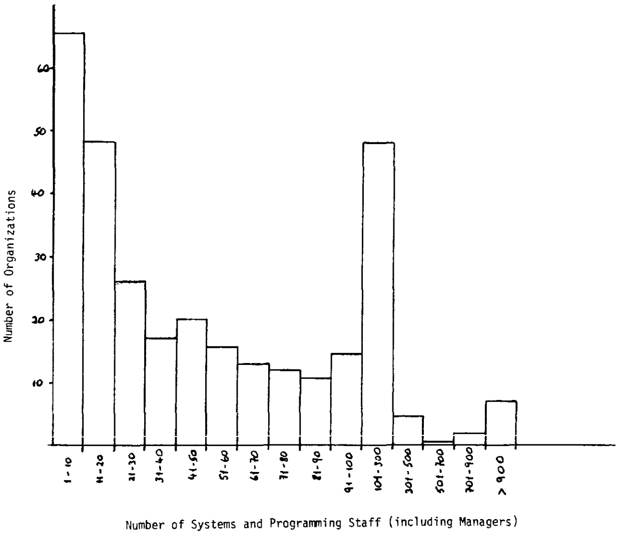


Figure 1—Numbers of Staff

Given this, the percentages of effort devoted to scientific/technical, commercial and system software are very much what might be expected (see Figure 2). So too were the languages used; over 50 percent of responders used COBOL. Approximately 25 percent used Assembler or PL/1 and rather less, about one sixth, used FORTRAN.

Interestingly, about one sixth also reported using some other language, which is rather higher than expected, given

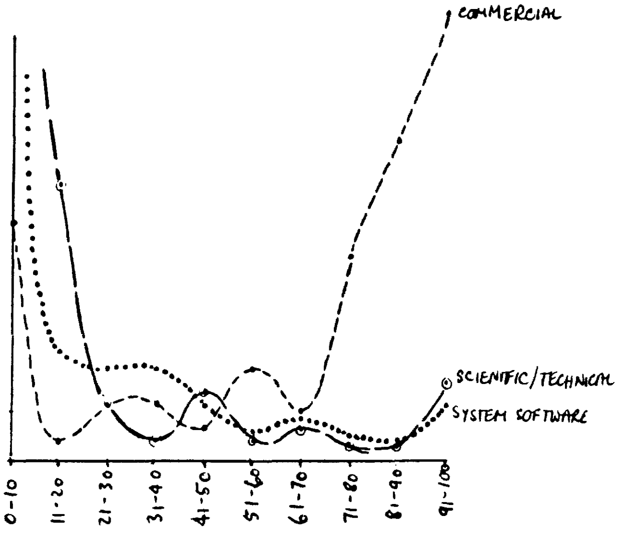


Figure 2—Types of computer application and percentage of systems/programming staff currently working on each type

the commercial orientation of the responders. True, ALGOL, BASIC and RPG accounted for a number of these but comments on the unsuitability of the major current languages for structured programming were frequently made and there was significant evidence of a search being made for more suitable languages. No estimate of the amount of use of each language was asked for; had it been, it is likely that these other languages would have paled into insignificance. However, it would appear that the use of structured programming techniques is creating a greater awareness among commercial programmers of how a language shapes and constrains one's approach to problems.

Regarding the mix of maintenance and new development work, two thirds of responders reported maintenance work involving between 20 and 60 percent of their effort and new development work between 40 and 80 percent. The median was 71-80 percent new work, 20-29 percent maintenance, with the curve heavily skewed towards new work. These figures bode well for the adoption of new techniques and suggest a limited demand for structuring cosmetics for existing software.

The processors in use at the installations surveyed are of no particular relevance. The purpose of the question was simply to check on the sample obtained. As expected, IBM machines predominate and a wide spectrum of other suppliers' equipment was also represented in the sample.

INTRODUCTION TO STRUCTURED PROGRAMMING

Section B of the questionnaire sought to establish the means by which structured programming came to the attention of users and how implementation of the techniques was started. As might be expected, nearly all responders reported first learning of structured programming through the general technical press or at conferences. Most then sent one or two people on courses before using one or more of the techniques on a project.

Within the survey sample, there was not a single organization that had adopted a set of techniques as standard, to be used by existing staff and to form part of the training program for new staff. A few, but only a few, organizations were planning toward this goal and had made measurable progress. Given the general familiarity with structured programming, the practice of it is perhaps surprisingly spasmodic. The general impression gained is of apostles spreading the gospel where they can but generally with little co-ordinated support from software managers. Many reasons can be advanced for this but they will not be discussed here.

Regarding specific techniques used first, the largest group of responders used structured design and structured coding in combination. The next largest group described themselves as using structured coding and top-down development in combination. Smaller groups had used structured design or structured coding alone. Overall. across all combinations of techniques, structured design, structured coding and top-down implementation were most frequently cited as techniques used in the first step and HIPO and structured analysis were the least frequently cited.

EFFECTS OF STRUCTURED PROGRAMMING

Sections C and D sought to establish the noted effects of using structured programming techniques and the length of experience on which the observations were based. The large majority of responders had two years or less experience of structured programming, most of those having between one and two years experience. Three organizations claimed 13 or 14 years experience.

The principal effects reported from the use of the individual techniques listed in section D of the questionnaire are given below.

STRUCTURED ANALYSIS

The most noted effects of the use of structured analysis were a decrease in the number of errors made and a decrease in the number of man-hours spent debugging. The next most significant gains reported were, in order of decreasing importance, greater control of project progress, greater motivation/job satisfaction of DP staff and greater user satisfaction. It was perhaps also relevant that very nearly equal numbers of responders reported (a) a slight increase (b) no change and (c) a slight decrease in both the elapsed time of projects and the total number of man-hours of effort expended.

STRUCTURED DESIGN

The results for structured design were almost exactly the same as for structured analysis, except that an increase in technical coordination replaced increased user satisfaction as a major gain.

It will be seen from the questionnaire that responders were asked to state the type of structured analysis or design. It was obvious from the responses that there was great difficulty in differentiating between analysis and design and that, when unable to differentiate adequately between the two, most responders opted to describe the overall process as design. This being the case, it may be that the noted effects of structured analysis and structured design should be considered together rather than independently.

STRUCTURED CODING

The results for structured coding were remarkably consistent. It was claimed that benefits were noted right across the board and there were virtually no dissenters from this glowing picture. The greatest gains were observed in the reduction of debugging man-hours and of computer time for testing. Only slightly lower gains were claimed in relation to project elapsed time, project man-months, errors made and maintenance man-hours.

TOP-DOWN IMPLEMENTATION AND TESTING

One benefit stood out above all others for top-down implementation and testing and that was the reduction noted in the number of man-hours spent on debugging. Significant claims of benefits were also made in relation to the number of errors made and the productivity of DP staff. Once again there was no significant contention in the results noted.

TEAM OPERATIONS

The noted effects of team operations were somewhat difficult to interpret. For instance, gains were consistently reported in the productivity of DP staff whilst, at the same time, there was no consistency in the observed effects on project man-months and project elapsed time. There was, however, unanimity in reporting greater motivation/job satisfaction of DP staff, so it is possible that the productivity gains were more imagined than tangible.

In general, the noted effects of team operations were much less certain than was the case with most of the other techniques. The apparent misgivings regarding team operations showed up strongly in the follow-up to the questionnaire, particularly in relation to the chief programmer team structure.

PROJECT LIBRARY OPERATIONS

Project library operations was one of the least used of the techniques listed and the benefits noted were generally rather muted. However, there was little inconsistency in the results, the greatest gains being claimed, as might be expected, in the areas of technical co-ordination, control of project progress and the number of errors made.

STRUCTURED WALKTHROUGHS

The most consistently noted effect of structured walkthroughs was, the expected reduction in the number of errors made and of the man-hours spent de-bugging. Rather surprisingly, however, the next most consistently reported gain was an increase in the motivation/job satisfaction of DP staff. The fourth most consistent gain was in the area of technical coordination.

Follow-up to the survey, and section G of the survey itself, revealed a considerable amount of disagreement in user's experience of structured walkthroughs. Particularly noticeable was a tendency for opinion to polarize either very strongly in favor of structured walkthroughs or very strongly against them.

Contention reveals itself in this section of the survey in that, despite the consistently reported gains in relation to errors made and debugging man-hours, there was no agreement on the effect of structured walkthroughs on project elapsed time and project man-months. As many responders reported these times increased as reported then decreased.

HIPO

The sample of responders with experience of HIPO was relatively small (44 in total). The most consistently reported benefit was a reduction in the number of errors made, followed by gains in user satisfaction and technical co-ordination. Once again, there was considerable disagreement over the effect on project elapsed time and project man-hours.

PROJECT MANAGEMENT SYSTEM

This title was devised to allow experience on any of the several formalized project management systems around to be recorded. Two results were noteworthy. Despite the fact that gains were consistently reported under the control and management headings, there was no consistency in the noted effects on project elapsed time and project man-months. Also, very few responders claimed any benefits other than under the control and management headings.

FACTS AND FIGURES

Section E of the questionnaire was an attempt to gather some hard data. Such data is notoriously difficult to find and, even when available, often cannot easily be interpreted because of the complex of factors affecting it. For this last reason, no attempt was made to try to obtain data with a view to producing direct, quantified comparisons between the results obtained in different organizations.

Figures quoted for the productivity of programmers writing COBOL programs illustrate this point very well. A number of responders quoted the productivity achieved, using various combinations of structured programming techniques, in terms of COBOL statements per man-day over the duration of a project from design through to acceptance testing. The figures quoted are as follows: 26, 28, 30, 36, 40, 48, 56, 60, 65, 75, 100 and 150! Perhaps the only significance of these figures is that each represented, for a different organization, a considerable jump in productivity. Only five responders gave the productivity achieved prior to using structured programming and all five quoted figures in the range of 10-12 COBOL statements per man-day.

Most responders had no data to offer. A number had collected data but had introduced structured programming on projects involving a new type of work and often new hardware as well; the effects due to the new software production methods were therefore uncertain.

Fifty-six responders provided facts and figures of one sort or another. The aspects of the software development process for which figures were most frequently given were elapsed time, design time, testing time, maintenance effort, productivity (lines of code per day) and documentation. Before these are discussed, it should perhaps be pointed out that the significance of improvements depends to some extent on the state of affairs before the introduction of the new methods. In many cases, the introduction of structured programming would appear to be the first time that what could properly be termed a method was used at all.

PROJECT ELAPSED TIME

Few responders reported gains in elapsed time. Those that did had achieved gains of between 10 and 20 percent. The most striking result was the number of reports of project time estimates being consistently met. One is so used to hearing of projects being behind schedule that one does not stop to think of the general state of depression in which most software project managers live. A single quote sums up the normal situation: "The certain knowledge of failure is the only thing that keeps you sane." The relief with which it was reported that time estimates were consistently being met was most noticeable and appeared to provide project managers with a general confidence in their work that they had never had before.

DESIGN TIME

Design time was consistently reported to have been increased by the introduction of structured programming. Increases in design time reported were between 50 and 200 percent with most responders reporting an increase of 100 percent or more. There was evidence of a more thorough approach to design, which may have accounted for some of the higher figures reported: "We felt free to examine two or three alternatives instead of grasping at the first working solution."

DEBUGGING, TESTING AND MAINTENANCE

Large gains in debugging man-hours, computer time for testing and the time and number of people devoted to maintenance were generally reported. Overall, the reductions were in the range of 50 to 80 percent and comments such as "at no time was it necessary to alter any processing sequence," were common. The example below is fairly typical.

Techniques used:

Structured design and coding, top-down implementation and testing, team operations, project library, walkthroughs, HIPO

Type of work:

Commercial programming in PL/1

Results	Pre- 4/75	4/76
Average lines of code per man-day	13	36
% staff on maintenance	60	40
Total m/c usage on testing		−15%
Errors per program not identified until acceptance testing or final systems testing		−65%

DOCUMENTATION

Time for documentation was generally reported to be increased, often with the corollary that it hadn't been done either properly or at all before. "Our gain in code/test throughput has somewhat been used up by doing tasks which were not done before, such as user documentation." Where percentage increases were quoted, 50 percent was a typical figure.

MAJOR BENEFITS AND PROBLEMS

Section F of the questionnaire sought an overall assessment of the major benefits and problems experienced in relation to structured programming. Broadly, the major benefits most consistently reported were the quality and maintainability of the product, the productivity and job satisfaction of the DP staff and the greater visibility and control in the software development process. The most consistently reported problems were in gaining acceptance and backing for structured programming, both by higher management and by senior programmers, the time and cost involved in retraining staff, the difficulty in producing programming standards, especially in relation to current languages, and in enforcing them and the cost of documentation. Some interesting points emerged.

The most evident benefit can be described as greater visibility and control; this was shown in many ways. Several responders said that their program specifications were much clearer than before and that staff now had the confidence to reject specifications that were inadequate. Many also said that modifications to large programs were no longer dreaded and that programs could be passed between staff easily. Many claimed great improvements in documentation, although many also reported the time and cost involved in producing the documentation as a problem, which suggests that documentation is still often regarded as an adjunct to rather than an integral part of software. Overall, the comments displayed a new confidence that understanding and control of the software development process had been achieved. This had an important side-effect. Increased user satisfaction was widely reported and several responders said that user departments were now coming forward and asking for more jobs to be programmed, rather than seeking to avoid involvement with DP.

An interesting contrast were the claims made for greater motivation and job satisfaction and the many comments that gaining acceptance of structured programming was a problem. Gaining acceptance was, in fact, the major problem reported. Lack of acceptance and support by senior management appears to be an unresolved problem and is one that could seriously impede the progress of structured programming. Lack of acceptance by senior programming staff appears in many cases to have been a temporary problem, a number of different methods being used to win these staff over. A few responders issued a caution against over-optimism, pointing out that the benefits reported had been hard won and involved a great deal of discipline and monitoring.

A number of responders reported confusion caused by the number of different versions of structured programming and had difficulty in assessing whether the differences between them were real or imaginary; this resulted in difficulty in deciding in detail which course to follow.

An interesting difficulty is that programmer team structures apparently are rendered impossible by some of the rigidly enforced personnel hierarchies that are encountered in some public bodies. There was apparently no way round this problem.

After lack of acceptance, the major problem reported was the cost of retraining DP staff. Several responders pointed out that having just a few staff trained in structured programming could give you the worst of both worlds but that the cost of retraining everybody in a short space of time was unacceptable; moreover, the time involved made such a course impractical. Some large organizations have got round this problem by introducing structured programming to relatively self-contained subsets of the DP staff one at a time. Another problem reported in this area was the lack of training courses in structured programming for junior trainees.

RECOMMENDED IMPLEMENTATION SEQUENCE

There was no agreement on the sequence in which the various structured programming techniques should be introduced except that most responders recommended that structured coding and/or structured analysis and design should be introduced first. This, as it happens, was what most of the responders had done so this result is perhaps to be expected.

HINDSIGHT

Section G of the questionnaire sought to gain the benefit of hindsight. Most responders had approached structured programming in a rather ad hoc manner and were very conscious of the fact. Therefore, overall, the principal recommendation was to adopt a planned approach and to stick to it. This general recommendation broke down into the following list of most consistently mentioned points:

1. Get advice from an outside consultant or an experienced user
2. Appoint an internal consultant
3. Do not introduce structured programming in conjunction with other major changes
4. Select a pilot project that is not too difficult
5. Use a team on the pilot project and then use that team to organize and control the training of others
6. Use your most competent personnel for the pilot project
7. Monitor the pilot project closely and collect data on it
8. Get management commitment
9. Structured programming must be sold to staff
10. Don't oversell
11. Anticipate acceptance problems
12. Apply more time and money to training
13. Reduce training elapsed time
14. Have follow-up courses
15. Apply more effort on standards
16. Don't enforce standards by fiat.

One or two conflicts emerged. There were differences of opinion on the amount of selling of structured programming that should be done within the organization and also on the degree to which adherence to the new techniques should be enforced. There was strong disagreement on the number of techniques that should be introduced initially, responders polarizing into the all-at-once camp or the little-at-a-time camp. There was also strong disagreement on the usefulness or otherwise of programmer teams and walkthroughs.

Some of the recommendations were of the more-in-hope-than-anger variety. Getting management commitment, for instance, was recommended by very many responders but had been achieved by very few. Similarly, many of the responders who recommended more training and more concentrated periods of it recognized the impracticality of their recommendations in many situations.

SUMMARY AND CONCLUSIONS

The results of the questionnaire survey are subject to a number of qualifications that should be made explicit. In the first place, no major failures were encountered in the survey sample. This is hardly surprising, in that organizations asked to volunteer information are rather more apt to do so about their successes. Clearly, it is likely that a number of failed attempts to implement structured programming do exist and, if the experience gained could be made public, it might be even more revealing than the experience of success.

Secondly, the organizations using structured programming are probably among the most experienced in producing software and hence more likely to succeed. Also, the very attempt to introduce new techniques suggests a commitment to improving their methods of software production (despite the lack of management backing).

Thirdly, it can only be guessed how much of the improvement reported is due to the techniques themselves and how much to the training and extra effort put into their introduction. Despite the generally reported need for more training and greater planning, some extra effort must have been involved in introducing structured programming.

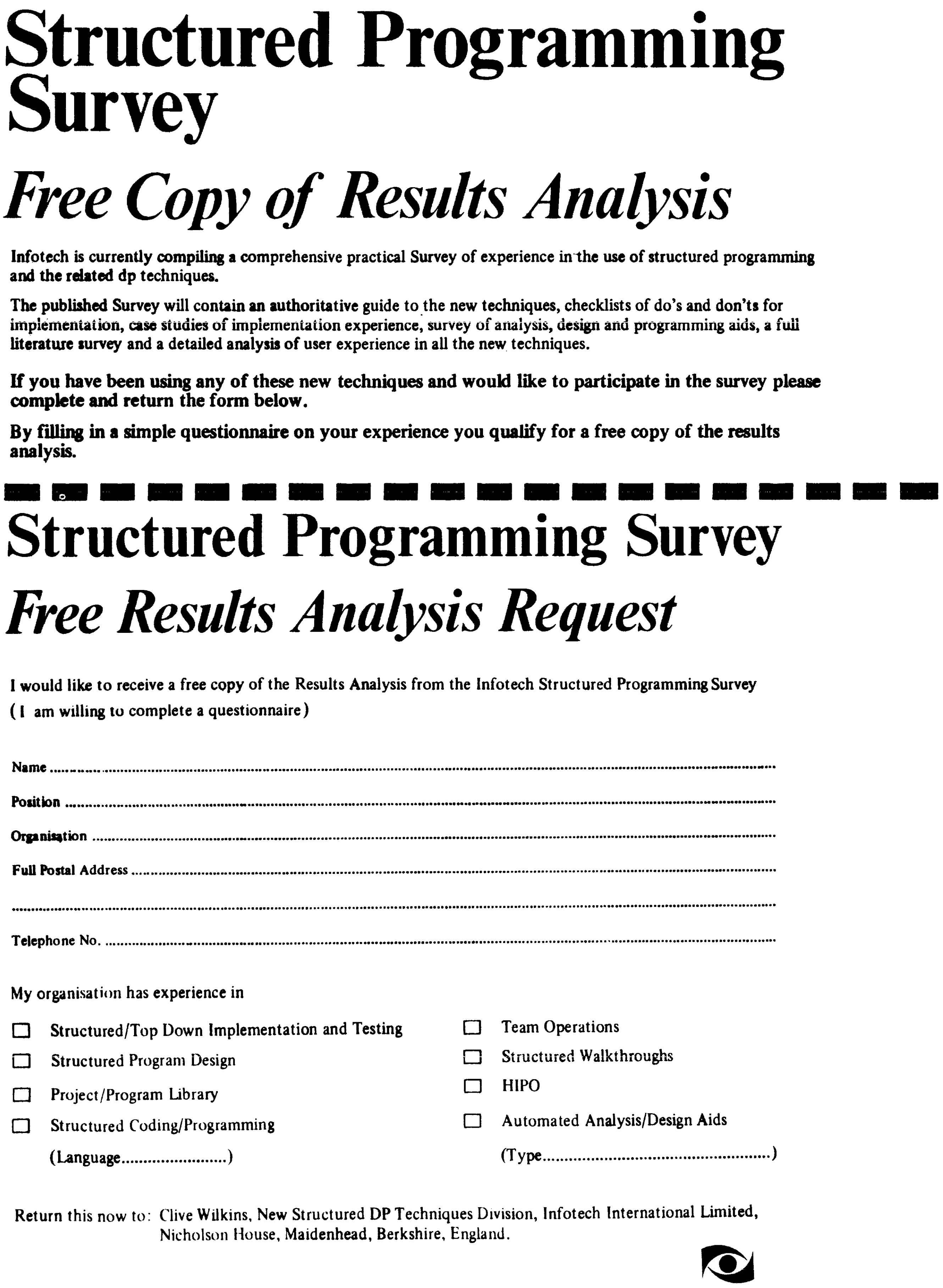
Fourthly, many of the figures quoted were taken from pilot projects, which often show a higher degree of success than subsequent general implementation.

Fifthly, the "specific" techniques listed might more aptly be described as somewhat broad categories of techniques. It is clear that a certain amount of confusion exists about the nature of differences between different versions of the true gospel. No attempt was made to be precise in the definition of terms because it was felt that any such attempt would be bound to fail or else to reduce the surveyed population, probably to 1. However, the purpose of this survey was not to provide insights that might further the theoretical basis of programming but to seek the opinions of those practising programming under normal commercial constraints.

In spite of these qualifications, it is contended that the survey was a useful exercise and did produce helpful information. It is certain that many organizations are making significant gains, on a continuing basis, both in the quality of their software and in the cost of producing it, through the use of structured programming techniques. Above all, there emerges the clear impression that, at least for the large majority of software projects, the software development process has become visible and controllable. That is a significant step for the management of software production in the commercial field.

ACKNOWLEDGMENT

The survey described in this paper is the work of several people within Infotech, notably C. Watkins and D. Bates. The raw data was obtained with the kind co-operation of organizations throughout the world that are too numerous to mention individually but to whom thanks are due.



Structured Programming Survey

Free Copy of Results Analysis

Infotech is currently compiling a comprehensive practical Survey of experience in the use of structured programming and the related dp techniques.

The published Survey will contain an authoritative guide to the new techniques, checklists of do's and don'ts for implementation, case studies of implementation experience, survey of analysis, design and programming aids, a full literature survey and a detailed analysis of user experience in all the new techniques.

If you have been using any of these new techniques and would like to participate in the survey please complete and return the form below.

By filling in a simple questionnaire on your experience you qualify for a free copy of the results analysis.

Structured Programming Survey

Free Results Analysis Request

I would like to receive a free copy of the Results Analysis from the Infotech Structured Programming Survey
(I am willing to complete a questionnaire)

Name ....

Position ....

Organisation ....

Full Postal Address ....

....

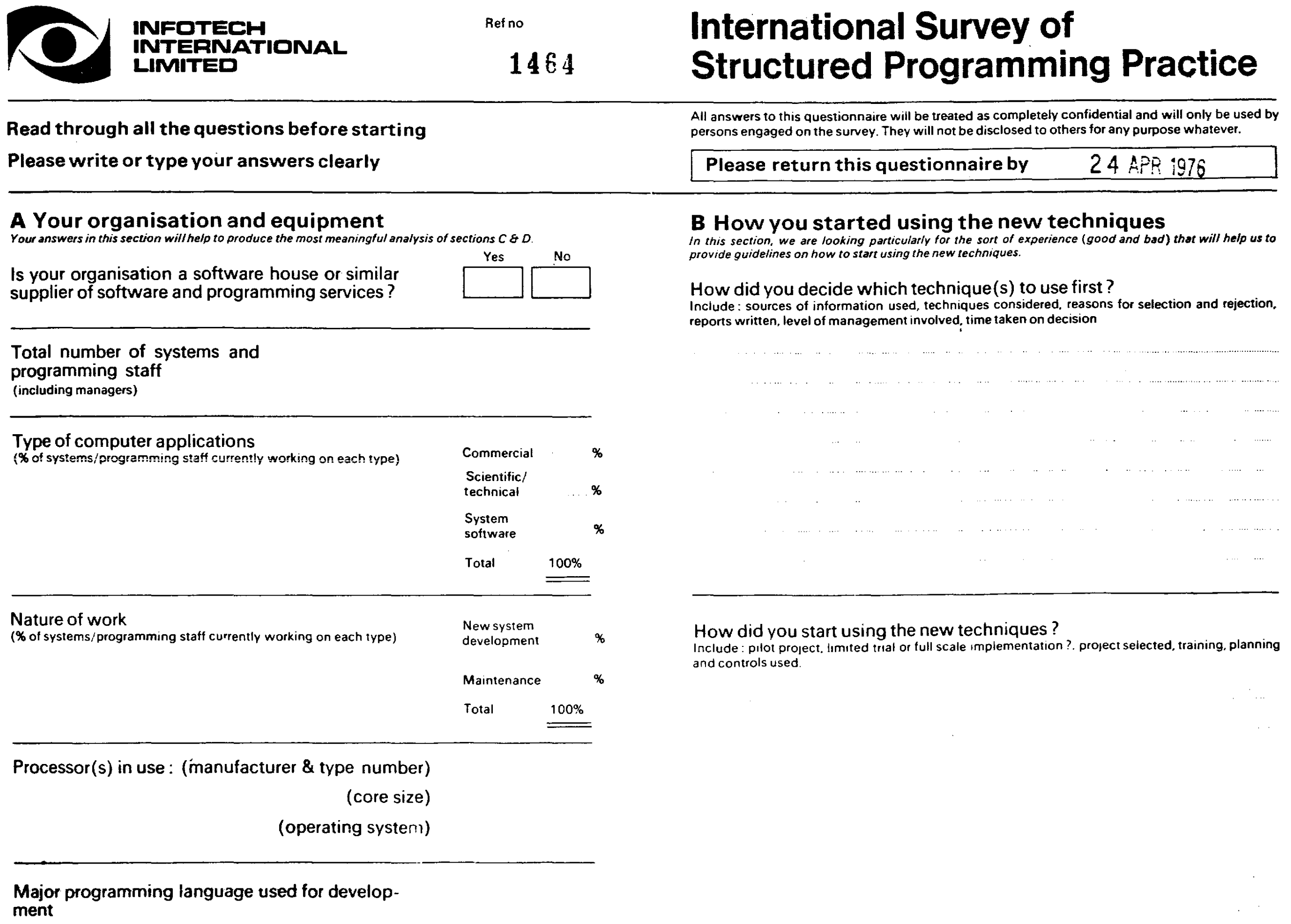
Telephone No. ....

My organisation has experience in

- □ Structured/Top Down Implementation and Testing
- □ Structured Program Design
- □ Project/Program Library
- □ Structured Coding/Programming
  (Language........................)
- □ Team Operations
- □ Structured Walkthroughs
- □ HIPO
- □ Automated Analysis/Design Aids
  (Type...)

Return this now to: Clive Wilkins, New Structured DP Techniques Division, Infotech International Limited,
Nicholson House, Maidenhead, Berkshire, England.

APPENDIX I—Brief questionnaire



INFOTECH INTERNATIONAL LIMITED

Ref no

1464

International Survey of Structured Programming Practice

Read through all the questions before starting

Please write or type your answers clearly

All answers to this questionnaire will be treated as completely confidential and will only be used by persons engaged on the survey. They will not be disclosed to others for any purpose whatever.

Please return this questionnaire by 24 APR 1976

A Your organisation and equipment

Your answers in this section will help to produce the most meaningful analysis of sections C & D.

	Yes	No
Is your organisation a software house or similar supplier of software and programming services?	□	□

Total number of systems and programming staff
(including managers)

Type of computer applications
(% of systems/programming staff currently working on each type)

Commercial		%
Scientific/ technical		%
System software		%
Total	100%	

Nature of work
(% of systems/programming staff currently working on each type)

New system development		%
Maintenance		%
Total	100%	

Processor(s) in use: (manufacturer & type number)
(core size)
(operating system)

Major programming language used for development

B How you started using the new techniques

In this section, we are looking particularly for the sort of experience (good and bad) that will help us to provide guidelines on how to start using the new techniques.

How did you decide which technique(s) to use first?
Include: sources of information used, techniques considered, reasons for selection and rejection, reports written, level of management involved, time taken on decision

How did you start using the new techniques?
Include: pilot project, limited trial or full scale implementation?, project selected, training, planning and controls used.

APPENDIX I—Detailed questionnaire

C Length of experience

This section is designed to show the sequence in which you are using, or going to use, the techniques and the extent of your experience
(see instructions below table)

D Noted effects of using the new techniques

This is the most difficult but most fruitful section for you to complete. We would like to know, technique by technique, the effects you have noted resulting from their use. Our analysis of this information will show whether your use of particular techniques has produced similar benefits and penalties to others.

Use this scale to show in the table below the effects of each technique you are using (see instructions below table)

Much more/ much greater = **5**	More/ greater = **4**	Same as before = **3**	Less = **2**	Much less = **1**	Not noted = **0**

			AREAS AFFECTED	Development				Production & maintenance		DP staff				Users	Control & management				
Date commence using	Date first project complete	Date fully used on all projects	THE TECHNIQUES	Total project elapsed time	Total project man months	Computer time for testing	Debugging man hours	Run time on computer	Maintenance man hours	Technical capability	Motivation/job satisfaction	Errors made	Productivity	User satisfaction	Technical co-ordination	Control of project progress	Control of project costs	Control of dp resources	Other effects noted
month/ year	month/ year	month/ year																	
			1 Structured analysis (type)																
			2 Structured design (type)																
			3 Structured coding																
			4 Top down implementation & testing																
			5 Team operations																
			6 Project library operations																
			7 Structured walkthroughs																
			8 HIPO																
			9 Project management system																
			TECHNIQUES USED IN COMBINATION (circle used techniques)																
			1 2 3 4 5 6 7 8 9																
			1 2 3 4 5 6 7 8 9																

i) Complete all dates for each technique
ii) Give planned or estimated dates for techniques implementation not yet completed or started
iii) If not planned write NP

i) Complete only rows relating to techniques used
ii) Where techniques have been used in combination and the noted effects cannot be attributed to individual techniques, complete the TECHNIQUES USED IN COMBINATION rows.
iii) Enter a number from **0 - 5** (see scale above) in each column to indicate effect of using technique.

Example
For a particular technique, eg. 2 Structured design, a **5** entered in the first column indicates that one noted effect of using structured design is that total project elapsed time is much greater. Similarly a **3** entered in the second column indicates that the total man months needed to complete a project using structured design is the same as before.

APPENDIX I—Detailed questionnaire (continued)

E Facts and figures

In section D, we asked you to note the effects of using the new techniques. We are asking you here to provide us with any facts and figures you have that will help us to quantify the experience summary we produce.

What facts and figures do you have on your experience with the new techniques?

If you wish to enclose a separate report please tick box below:

□ Please see report enclosed.

F Results of your experience

This section is provided to help you compare the problems you have met and the benefits you have gained with those of other organisations.

What are the major problems you have encountered so far and how have they been overcome?

e.g. lack of training, management commitment and staff acceptance; over-optimism; time, cost and control of change.

What would you say were the most important benefits to your organisation of using the new techniques?

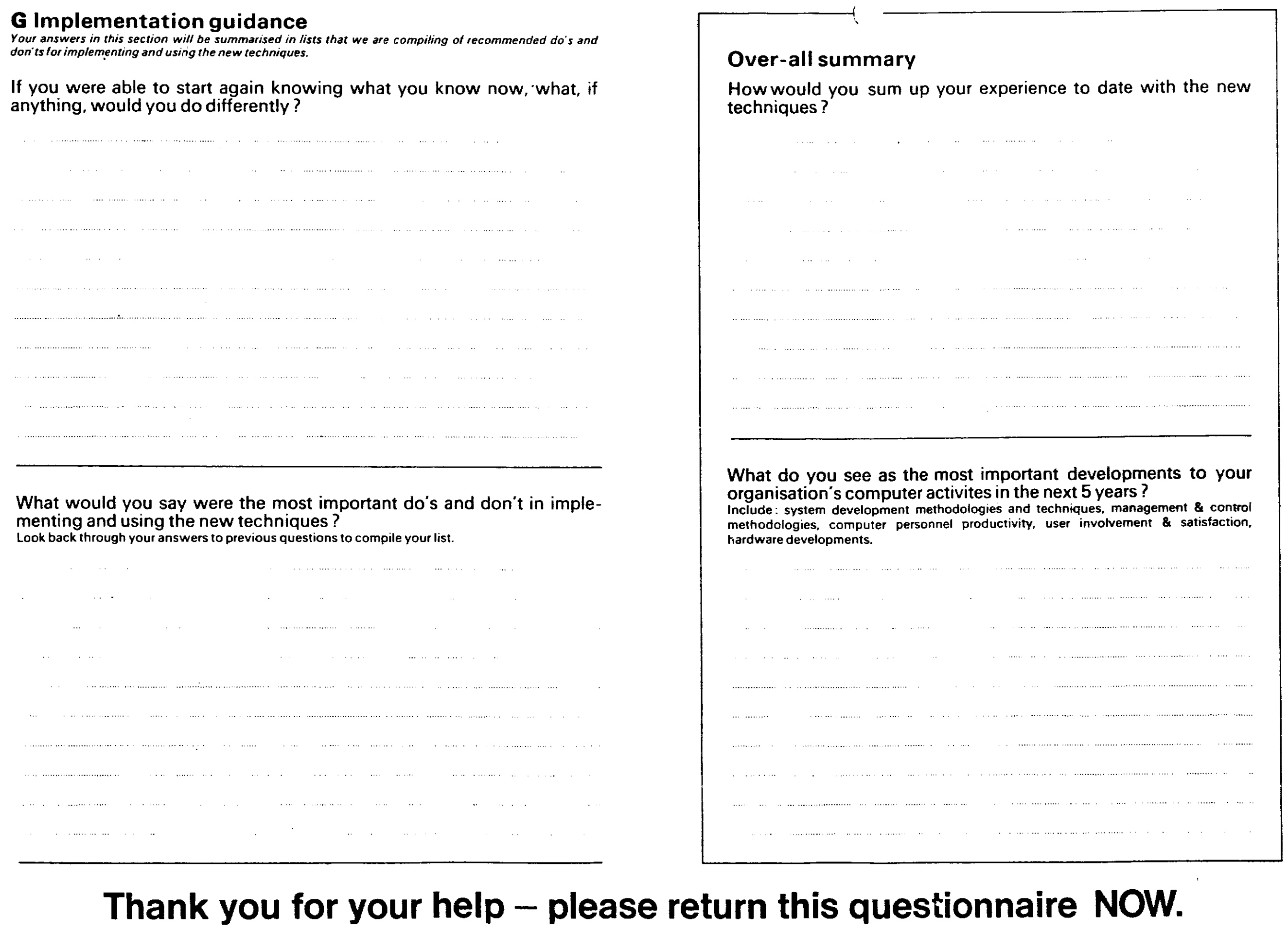
See 'effects' headings to columns in section D for possible benefit areas.

As a result of your experience so far, in what order would you recommend that the new techniques be implemented?

Use the technique numbers 1–9 as in sections C, D above to indicate recommended implementation sequence. You may group techniques at any stage.

1st □ 2nd □ 3rd □ 4th □ 5th □ Never □

APPENDIX I—Detailed questionnaire (continued)



G Implementation guidance

Your answers in this section will be summarised in lists that we are compiling of recommended do's and don'ts for implementing and using the new techniques.

If you were able to start again knowing what you know now, 'what, if anything, would you do differently?

What would you say were the most important do's and don't in implementing and using the new techniques?

Look back through your answers to previous questions to compile your list.

Over-all summary

How would you sum up your experience to date with the new techniques?

What do you see as the most important developments to your organisation's computer activites in the next 5 years?

Include: system development methodologies and techniques, management & control methodologies, computer personnel productivity, user involvement & satisfaction, hardware developments.

Thank you for your help – please return this questionnaire NOW.

APPENDIX I—Detailed questionnaire (continued)

An interactive system for aiding management decision making

by ROBERT C. GAMMILL and HERBERT J. SHUKIAR
The Rand Corporation
Santa Monica, California

ABSTRACT

An interactive application program running on the PDP11 UNIX time-sharing system is described. This program allows files describing sets of objects to be searched, and each object evaluated against a selection expression. Objects satisfying the expression are kept in primary storage as sets (linked lists), and the user can delete or move objects from one set to another under control of other interactively composed selection criteria. The system thereby "assists" the user's decision making process. Examples of objects might include people, machines, rooms, organizations, cities and tasks. The program itself is not revolutionary in concept, although it does take some ideas from artificial intelligence. However, its rapid and evolutionary development on a minicomputer shows how highly accessible and affordable computers with good software production tools can bring the computer, as a decision making aid, to organizations not previously able to make effective use of computation. Furthermore, the utilization of advanced software tools has permitted a flexible English-like man-machine interface to be developed, allowing use by computer-naive managers.

INTRODUCTION

The Information Sciences Department at Rand has been studying the use of personal computers in a number of environments, including clinical research, publication, communication via memos and messages, clerical and secretarial services, and management decision making. The system to be described falls into the latter area. Our work on personal computers has not attempted to use micro-computer hardware, since the present state of that technology does not represent what will be available in the future, and because we choose to study the software problems involved in smallness and in providing man-machine interfaces for computer-naive users. We assume that affordable hardware will ultimately reach sufficient power to support the mini-computer software upon which we do our research. These assumptions can certainly be questioned, but it is our purpose here only to state them as the basis of operation. The hardware configuration being used is a PDP11/70 interfaced to the ARPAnet and running the Bell Labs UNIX operating system. Special software added at The Rand Corporation includes the network host interfaces and a special two dimensional text editor, upon which all software for the system being described and this paper were generated.

The goal of this paper is to describe the development of a management decision aiding software system which has been designed and developed in an evolutionary fashion. This system is not revolutionary in concept, unless one considers smallness and flexibility to be revolutionary characteristics in a management oriented system. What is of interest in this system is the ease and speed with which it was constructed, the clarity and ease of modification and English-like nature of its interaction with the user. All of these things resulted from the availability of superb software development tools available in the UNIX operating system at Rand. The goal of this paper will therefore be to describe how these tools supported and guided the development of this system and affected the structure of the software in a very positive way. In the following section we itemize the most important tools.

THE TOOLS

The UNIX operating system[1] was an important tool used in this effort. The file and directory system was especially important in allowing us to structure and store our information as text streams with readable syntax, and collected in logical units. Documentation, plans, source code, command files, data files and memos were kept in a tree structure of file directories, that were shared among the two people working on the project. Latest source versions were always easy to identify, and independent and shared activities within the design and development were helped considerably by the ease of structuring and the files of documentation and notes.

The two dimensional Rand text editor[2] running under UNIX, was another important tool. Used on the Ann Arbor CRT terminal, which allows 40 lines of 80 characters to be displayed, this editor allows most subprograms to be viewed in their entirety. This allows both declarations and code to be in view simultaneously, making visual scanning for problems much easier. The editor also allows parts of files to be displayed in different windows on the screen at the same time. This is especially important when keeping a global data structure declaration in view while examining some routine that manipulates an instance of it. Another use of the window capability is in scanning code for a bug, while keeping the input data and output trace (showing the bug) on the screen at the same time. Such capabilities can make interactive debugging (without hardcopy) a much less tedious process than it would otherwise be.

The C programming language,[3] a Bell Labs product, was also an important tool. This language resembles BCPL,[4] and its compiler has been carefully constructed so that the penalties for using it are relatively small. The UNIX operating system is almost completely coded in C, so a subsystem implementor feels considerable confidence that little is being lost, and much is to be gained by using this language.

Yacc, "Yet Another Compiler-Compiler," and yet another Bell Labs product whose bottom up parsing methods are superbly described in Reference 5, was the most important tool of all. It created a helpful structure for this command interpreting system by separating the syntactic portions of the code from the semantics. The top level of our management decision aiding system is a parser whose flow of control is defined by a LALR(1) grammar for the commands. The parser is fed input tokens by a lexical analyzer written in C. When a sentence which is defined to be a command is recognized by the parser, semantics routines associated with the grammar rule are called. In a compiler these routines would be code generators, but for an interpreter these routines carry out the actual work specified by the command.

An important result of the use of Yacc was the ability to implement, in a direct manner, a syntax for a fairly large subset of English allowing the man-machine interface to be very user oriented. This provides good readability, although ease of typing suffers as in COBOL. We believe the interface makes contemporary list processing and other computer science techniques available to computer-naive managers in a problem oriented manner and does not necessitate understanding of the computer science involved. Our belief has been supported by successful use of the system by computer-naive individuals.

The use of Yacc has also permitted the syntax of the interface to be defined without the complexity of the syntactic description becoming so great as to interfere with its easy extension and modification. Such ease is of great importance here, because the whole design and development effort was planned as a phased, evolutionary approach. This was done so that design flaws, subsequently discovered requirements and user feedback from early phases could be used as corrective input for later phases. A less diplomatic way of saying this is to note that when one discovers he has completely botched some part (or perhaps all) of a system design, the best situation that can be hoped for is that little emotional and monetary investment will have been put into the bad design, so that to discard and start again is a viable solution. Some of us who work with software have the feeling that this is the primary difference between "little" software and "BIG" software, and why the latter has had so many spectacular failures.

In the remainder of the material we will describe the system. Some material is also included showing how the software development tools aided us. We hope that software practitioners will see some interesting ideas on how to build a decision aiding system on a minicomputer, and that computer scientists will note the ways in which software production tools can be of aid in speeding and easing the development of useful application systems. It is precisely this boundary, with all its tensions, flim flam and lack of communication that we feel is of great importance to the future of computing. With the growth of personal computers, an even stronger spotlight will be thrown upon this arena. We would be remiss if we did not note here that Kernighan and Plauger in their recent book, "Software Tools,"[6] have done a careful job of examining some approaches to this area.

SYNTAX ANALYSIS

The management decision aid uses a lexical analyzer written in C, a LR(1) parser generated by Yacc, and a symbol table management package written in C, as its primary functional elements for implementing the command interpreter. The lexical analyzer identifies key words such as load, display, perform or file, and noise words such as "a", "an" and "the" among the identifiers and quoted character strings which are the content of commands in this system. The syntax analyzer recognizes sequences of tokens which make up phrases and commands, a command being an English language sentence terminated by a period. An example of a command is: "Load the cities whose region is Northwest and for which there is an industry whose type is electronics and whose number_of_facilities is greater than 10 from the data file." The symbol table management package allows important names like "cities," "region," and "industry" to be converted to code numbers. No information about the attributes or meaning of any word is kept in the symbol table, since that information is embedded in the data structures, to be described next.

DATA STRUCTURES

The primary data structure is the set. A set is a linked list of objects, each object representing a data entity, e.g., a person, a city, an organization, a machine, etc. Each object has a number of attributes that convey the information content of the object. Both the types of objects and their attributes are defined by the user at execution time via a template or data definition mechanism, that identifies to the system the names of sets of objects and their attributes. Stated differently, an object is a contiguous block of primary storage, and the template indicates where in the contiguous block the value of each attribute can be found. Objects are organized in sets, with each set owning a set header that serves as the starting point of a linked list spanning all the objects in that set. The set header contains, in addition to the pointer to the first object in the set, a pointer to the set's template (see Figure 1). With certain exceptions, all sets are linked together in a universe list, the exceptions being sets of objects that are owned by objects themselves, e.g., the set of departments in an organization, or the set of beds in a hospital room. The template is a pair of arrays, the first containing symbol table indices of names of attributes and the second containing attribute type codes for the values in the objects. In Figure 1, each of the person objects has two attributes, name and age. The type of the name attribute is text, indicating that the value found in the object will be a pointer to a text string, and the type of the age attribute is integer. Examples of useful noun phrases that can be formed about this structure are: "the age of the person whose name is John", which yields the integer 38, and "the name of the person whose age is greater than 40," which yields the text string "Jack."

At present six attribute types have been used. These are:

(a) Symbol table index—allows frequently used symbols to be compactly represented.

(b) Text—the value is a pointer (machine address) to a stored text string.

(c) Integer—normal machine signed integer.

(d) Date—a tightly packed representation of year, month and day, which can be algebraically compared with another date.

(e) Domain—a symbol table index with the added feature that a special set called "domain" will have an object with this attribute name, where an upper and lower index limit is specified. These limits, along with the numeric ordering of the indices between, specify a totally ordered set of symbols. This mechanism can be used to order a set of organizational titles so that greater than and less than comparisons may be carried out.

(f) Set—a pointer (machine address) to the set header of a set owned by this object. An example use of this attribute type is to give each of a set of companies, a set called departments, which contain objects having attributes that tell who heads the department and the date he was given the job, where it is located and what type of work is done.

The latter two types are of the most interest. The domain type is useful in a number of ways. The specification of a collection of legal symbols that may occur as the value of the attribute restricts the data that may be input to that position in the data structure. Thus, typing errors in symbols for an attribute with type domain can be easily caught. The ability to carry out algebraic comparison between symbols of a totally ordered domain also provides a powerful tool. Furthermore, we have extended the domain type to allow arbitrary binary relations to be specified between the symbols in a domain. This mechanism makes it possible to describe, for example, complex relations between organizational units, or job categories. Such areas provide some of

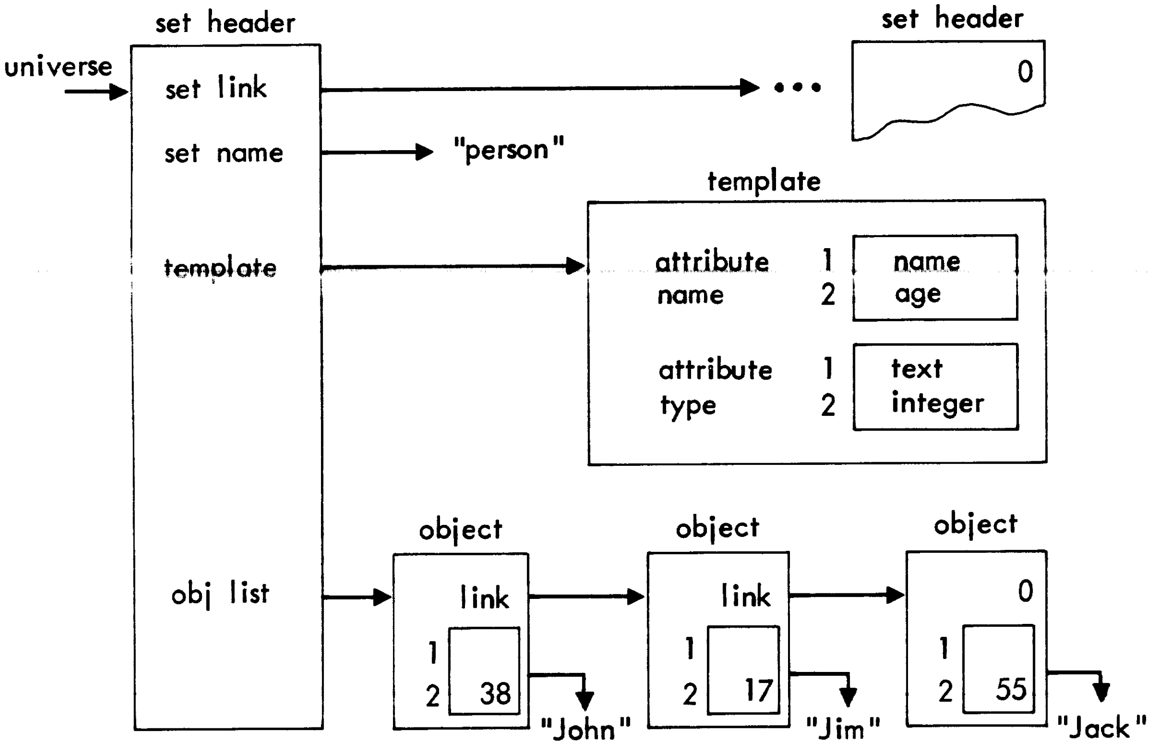


Figure 1—The set person, containing 3 objects, and its template that shows two attributes, name and age, with type text and integer

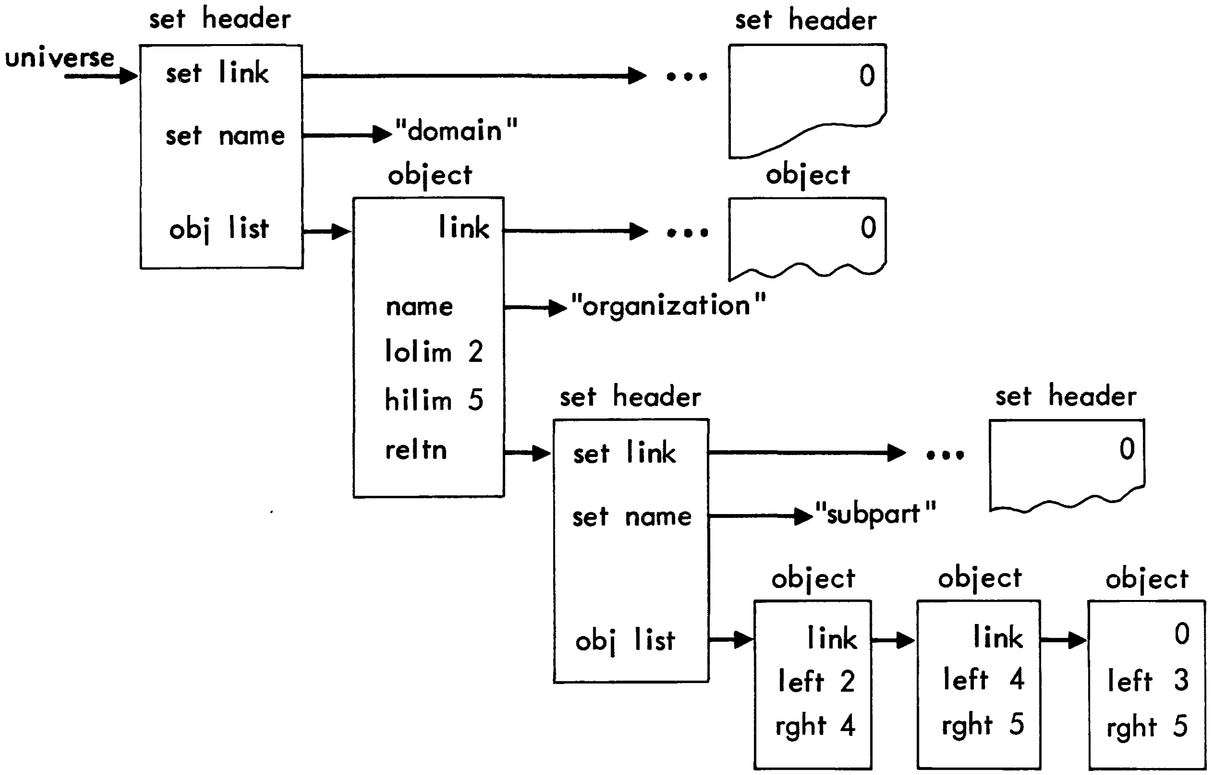
the most difficult problems for management oriented computer systems.

The set type is important because it provides a method for extending the power of an existing set data structure. This proved very helpful when it was decided to allow specification of arbitrary binary relations on a domain. As shown in Figure 2, the "domain" set contains objects which specify the upper and lower limits of symbol indices which are in the domain. By adding a "relation" attribute whose type is set, it is possible to store a set of ordered pairs of symbols which make up a binary relation on that domain.

Besides the data structures already described, there is also the format. Formats are objects in a set named format, which describe how output is to be generated, using a particular template and object. This allows numerous different kinds of reports to be generated concerning a set of objects. Formats can be used to produce tabular output forms with headings, or to produce running text in a more prose like style. Furthermore, all formats are user specified, allowing users to develop output displays tailored to their individual tastes and application areas.

FILES

The management decision aiding system is not a database management system in any normal sense. Files are read in a sequential fashion, to collect declarations of templates, formats and domains, to load data into a set, or to interpret a standard sequence of commands. However, all manipulations of sets, and the other data structures described, are carried out in primary storage. This, coupled with the fact that implementation is on a minicomputer, severely limits the amount of data that may be manipulated at one time. This design limitation was chosen at the outset. It results in



represents "subpart" = { < publicatns, admin_svcs > ,
< admin_svcs, admin > ,
< comp_ctr, admin > }

on the domain "organization" with symbols "publicatns" = 2,

"comp_ctr" = 3, "admin_svcs" = 4, and "admin" = 5.

Figure 2—A relation "subpart" on the domain "organization"

quick response to most commands (except those that parse long input data files), and serves to keep things simple and flexible. As a practical matter, sets containing 1000 objects, each having 8 attributes, can be loaded. This is sufficient for all tasks that we intend to undertake, but certainly limits the range of applications for which this system is applicable.

COMMANDS

The flexibility provided by Yacc and the lexical analyzer has made a wide variety of commands possible. As a result, it is easiest to describe the commands by giving a few examples, for a complete description requires some form of grammatical notation, and more space than can be devoted here. We will use the simple convention that ⟨name⟩ is a non-terminal symbol. An example of a command is:

Perform the ⟨filnam⟩ file. (1)

This command allows subsequent commands to be taken from the specified file. This command may occur in a file of commands. After completion of interpretation of the file, control returns to the next command after the perform command.

Load the ⟨setnam⟩ from the ⟨filnam⟩ file. (2)

The contents of the file is parsed and loaded into the set whose name has been given. Since files used by managers often contain information which must be protected (e.g., salary information), one can say:

Load the ⟨setnam⟩ from the encrypted ⟨filnam⟩ file. (3)

Here the user will be asked to type in a password for the specified file, so only those having the password for a particular file can examine its contents. The most important use of the load command is in selective loading from a file. In this mode a requirements expression is included in the command (or a file name is given where one can be found), that specifies which objects from the data file should be loaded and which discarded.

For example:

Load the doctors whose title is surgeon and whose age is less than 40 from the ⟨filnam⟩ file. (4)

This command will load only those surgeons that are less than 40 years old. In actual fact, each object in the data file will be parsed and entered into the specified set, but after each object is loaded it is tested against the requirements expression. If the object does not satisfy the expression, it is discarded. This makes it possible to load a subset of a data file into primary memory, even though the file is too large to fit there in its entirety. Finally, we should mention that any set being loaded will normally have been previously declared to have a template describing its members. If the set is being mentioned for the first time in the load command, a template name must be given.

Load the ⟨setnam⟩ using the template ⟨templnam⟩ for ⟨reqts_filnam⟩ from the ⟨data_filnam⟩ file. (5)

More will be said about requirements expressions in the next section. For now we will begin to use the symbol ⟨requirements⟩ to indicate that an expression, or a reference to a file name, or nothing is to be written there. There is another group of commands that allows objects to be removed from sets or moved from one set to another under the control of a requirements expression. These are:

Remove the ⟨setnam⟩ ⟨requirements⟩. (6)
Keep the ⟨setnam⟩ ⟨requirements⟩. (7)
Move the ⟨old-set⟩ to ⟨new-set⟩ ⟨requirements⟩. (8)

Another useful command is sort. The sort command allows a set to be sorted on some particular attribute in its template. If the attribute is of type text, the objects are sorted into alphabetical order. Any other type of attribute is sorted into numerically decreasing order (symbol table indices are sorted numerically). These conventions are inverted when the word "reversed" is added to the command. Examples are:

Sort the cities by name. (9)
Sort the secretaries by hire_date reversed. (10)

The display command is used for generating output. Some simple forms of the display command allow special information about the state of the system to be displayed. For example:

Display the size. (11)
Display the sets. (12)
Display the formats. (13)
Display the templates. (14)
Display the format called ⟨format_name⟩. (15)
Display the template called ⟨template_name⟩. (16)

Command (11) tells how much of primary memory (high water mark) has been used. Command (12) gives the name, template name and number of members for each set contained in the system universe. Only two sets in the universe do not have templates, the format and template sets. This prevents these sets from being displayed in the same way that other sets are, necessitating commands (13) through (16). All other sets can be displayed using the full form of the display command. For example:

Display the companies whose incorp_date is less than today minus 2 years and for which there is a department whose type is data_proc, using format list. (17)

Command (17) shows a fully expanded form of the display command. Each template has a slot for retaining the last format used, so once a format has been specified, it need not be mentioned again until a change is desired. Thus, if the full set is to be displayed, the command can be simply:

Display the companies. (18)

Other commands allow the display or editing of text files from within the system, and the declaiation of formats or templates.

REQUIREMENTS EXPRESSIONS

A number of examples of requirements expressions have already been given. The basic form of these expressions is a sequence of algebraic comparisons or special relation operations (such as "substring" for text or "is" for symbols) tied together by the logical connectives "and," "or" and "not." Great care has been taken to allow the logical connectives (especially "not") to occur in their natural English prose positions e.g., "a is not b" instead of "not a is b." Normally a requirements expression begins with one of the phrases, "whose," "for whom," or "for which," and those phrases may be freely used within the expression to improve readability. Examples are:

whose type is physics (19)
whose name contains "John" (20)
whose date is less than today minus 18 months (21)
whose category is greater than hourly (22)
whose organization is related by subpart to admin (23)

Expression (19) is useful between symbol table items, while (20) checks a text item for the specified substring. Expression (21) is useful for comparing dates, while (22) allows comparison of symbols in a totally ordered domain. Expression (23) shows how an arbitrary relation (see Figure 2) declared on a domain may be used to specify a requirement. In each case shown above, the expression relates the value of an attribute to a constant, or a simple expression made up of constrants. These seem to occur most frequently, but the us^r is not limited to them. Almost complete generality is allowed. The biggest danger at present is that type checking is not imposed, so the user may attempt to carry out comparisons which are meaningless, e.g., comparing an integer to a text string, and receive inexplicable results. We plan to remedy this omission in the near future, although that will eliminate many serendipitous discoveries of new ways to test for certain conditions in the data.

Finally, the most important relational test has been kept for last. The "there is" operator allows testing of a set owned by the object being evaluated, to see if that set contains an object satisfying the requirements expression following the "there is." For example:

for which there is an overhaul whose type is major (24)
and date is greater than today minus 15 months,
and date_of_manufacture is less than today minus 4 years.

Expression (24) tests for a major engine overhaul in the past 15 months, and then returns from the overhaul set (at the comma) to test the owning object for age greater than four years.

USER ORIENTATION

Great care has been taken to keep the command language as user and problem oriented as possible. Users of the management decision aid can state commands in terminology related to the application area. We don't claim that someone who is totally computer-naive can utilize the system as effectively as as a more computer-sophisticated user. However, by allowing an intelligent but computer-naive user to speak in a "comfortable" language, we believe that a natural transition will occur, with the user becoming aware of the primary computer techniques involved. We hope this transition process will take place in a natural fashion, supporting and reinforcing the user's intuition.

In addition, the system does not make decisions itself, but provides the user with information which enables him to make decisions. The user is given the ability to organize, investigate and keep track of many more alternatives. Viable alternatives can be separated from nonviable ones by applying selection criteria that move from the general to the specific constraints of the decision's environment. The system allows the user to make more precise the intuitive nature of the decision making process. We believe all of this increases the computer awareness of the user.

THE DESIGN AND DEVELOPMENT PROCESS

As can be seen from the preceding material, the management decision aiding system, although small and limited by design, has considerable power and flexibility within those limits. The interesting question is how a system of such power could have been created in seven months, with four man months of programming effort, and in only forty pages of source code? Some of our answers are:

(a) It is important to use a responsive and accessible interactive computer system, that provides good tools for software development and testing.
(b) Hire the most competent and experienced programmer possible. Quality appears to overcome price in the long run, especially when the software being developed is not very well understood.
(c) Make sure that more than one person is involved in the design process. Someone should have user sensitivity and someone should have computer science background. All major issues should be constructively argued. If an argument begins to go in circles and it's not critical, let the subject rest in the subconscious for a while and wait for the light to dawn. If it is critical, implement something simple, try it out and use that as the basis for further argument.
(d) Keep it simple. If it's bad, it was cheap. If it's good, it may be possible to make it elegant.
(e) Remember how fallible we all are! Some of the initial design will be garbage. Maybe all! Willingness to learn from our mistakes and the mistakes (and suc-

cesses) of others is essential. Stated another way, "steal" as many good ideas as possible from predecessors. In our case, one previous system[7] was similar in its handling of syntax and symbols and another (DOSS) had similar computer-naive manager users. These gave us an important initial boost in the right direction.

We plan to remain committed to an evolutionary approach to design and development. This system is beginning to harden, and we are confident that there are many faults in it that we have not yet recognized. We plan to carry out a design review in the near future, in order to get input from a group of uninvolved peers. We also will continue to pursue a high degree of user involvement to guide changes and extensions to system capabilities. Hopefully these methods will give us many new ideas about what should be redesigned or discarded. In this manner we hope to keep this system alive and improving, despite escalating user needs, increasing ego involvement by the designers, and increasing size of the project staff.

REFERENCES

1. Ritchie, D. M. and K. L. Thompson, "The UNIX Time-sharing System," *CACM*, Vol. 17, No. 7, July 1974, pp. 365-375.
2. Kelly, J., "New Editor Guide," R-2000 (draft), The Rand Corporation, Santa Monica, Calif.
3. Ritchie, D. M., "C Reference Manual," unpublished memorandum, Bell Telephone Laboratories, 1973.
4. Richards, M., "BCPL: A Tool for compiler writing and system programming," *AFIPS Conference Proceedings,* 1969 SJCC, Vol. 34, pp. 557-566.
5. Aho, A. V. and S. C. Johnson, "LR Parsing," *Computing Surveys,* Vol. 6, No. 2, June 1974, pp. 99-124.
6. Kernighan, B. W. and P. J. Plauger, *Software Tools,* Addison-Wesley, 1976.
7. Anderson, R. H. and J. J. Gillogly, "The Rand Intelligent Terminal Agent (RITA) as a Network Access Aid," *AFIPS Conference Proceedings,* Vol. 45, 1976 AFIPS Press, pp. 501-509.

An overview of independent, third-party computer maintenance

by HOWARD D. PONTY
Raytheon Service Company
Providence, Rhode Island

ABSTRACT

Describing the major reasons for its growth and providing a brief history, this paper gives a synoptic view of a new computer sub-industry: independent, third-party computer maintenance and field support services. It details the major modes of delivery of maintenance and field support services now available to users and compares them to one another. The organization and the technical services provided by independent, third-party contracting firms are described. Particular attention is paid to the special needs of mixed-vendor computer system users.

BACKGROUND

In the early days of the computer industry, user needs for system maintenance and field service support was not a matter of particular emphasis. The few large manufacturers each took pride in the service reliability of their products, and it was this aspect, and not the need for maintenance and field service, which was stressed. In addition, virtually all early systems were supplied by a single vendor. The vendor-manufacturer provided maintenance and field support services through his own service organization, which, in most cases, was adequately distributed geographically. The end-user, then, had a convenient single source to meet his maintenance and field support needs.

This basic situation began to change with the advent of plug-compatible equipment and the resulting proliferation of mixed-vendor systems. User requirements became more complex. And no longer could the end-user rely upon a single source to meet these requirements for all elements in his system. At first, his only course of action was to contract with the service organizations of each manufacturer whose equipment he used.

Compounding the situation was the entering into the field of numerous small manufacturers. While offering excellent products, many could not establish and maintain the extensive service networks needed to serve users dispersed nationwide or even worldwide. In order for these smaller manufacturers to market their products over larger geographic areas, they, too, needed a way of providing maintenance and field support services to their customers. If these manufacturers found it economically unfeasible to establish their own service networks, they had to find an alternative, as well.

These developments led to the formation of several "third party" organizations, independent of equipment manufacturers, whose business was providing maintenance and field support services for computer and computer peripheral equipment. At first, these organizations limited themselves to maintaining only the more commonly used equipment. Within a relatively short time, however, the need to provide services for mixed-vendor systems involving many smaller manufacturers' products became evident. Not only do these "third-party" organizations offer users a single source of maintenance, they also offer small manufacturers a nationwide network which can, under contract with these manufacturers, provide services to their customers.

GOVERNMENT USERS

The U. S. Government has been one of the major factors contributing to the rapid growth of "third-party" maintenance as a sub-industry within the computer field. A multi-~~million-dollar contract for third-party maintenance of the~~ U.S. Marine Corps systems in 1971 was one of the earliest and largest of its type.

Later, a government agency audit of this pioneer "third-party" maintenance contract found significant benefits from this kind of arrangement. In addition, a formal Office of Management and Budget policy encouraging the contracting of these and other services to the private sector.

Today, the U.S. Government remains one of the largest users of independent, "third-party" maintenance and field support services.

ALTERNATIVES FOR MEETING MAINTENANCE AND FIELD SUPPORT SERVICE NEEDS

In order to minimize system downtime and to obtain maximum service life, most data processing managers today are alert to the need for maintenance and field service support. This need can be met in four different ways:

1. By the user's own "in-house" service organization
2. Under separate contracts with manufacturer's service organizations
3. Under contract with a single "third-party" source
4. Through a combination of two or all of these

User "In-house" service organizations

In theory, it is possible for a large user organization to establish its own "in-house" service organization to maintain and repair its computer and computer peripheral equipment. In actual practice, however, this rarely is done, largely for economic reasons. In order to justify a full-time service force, an organization must have extensive equipment to be maintained and serviced. And, this equipment must be located either entirely or largely in a close geographic area. For when equipment is scattered over wide areas, the advantages of an in-house service organization are negated by time and distance.

One user with ample computer equipment to justify an "in-house service organization and enough equipment "clustered" in tight geographic areas to establish several regional "in-house" service offices, is the U.S. Government, which, as I have mentioned, has found other alternatives preferable.

In general, then, except for routine, day-to-day maintenance which all user organizations perform to some degree, "in-house" user maintenance has been found not to be a viable means of meeting most users' requirements.

Manufacturer service organizations

Before discussing manufacturer service organizations as an alternative source for maintenance and field support, it is important to understand what is meant by the term "manufacturer," and more specifically, "OEM" or original equipment manufacturer, as used in this paper.

In most cases, both "manufacturer" and "OEM" refer to a company which manufactures and sells computer and/or computer peripheral equipment. Sometimes, however, the term "manufacturer" can also apply to companies which might more properly be called "sales agents." These firms sell computer systems, with only some of the components, or even none at all, actually manufactured by that firm. Some of these firms have their own service organizations; others do not. These latter must arrange for maintenance and field support services for the systems they sell, either through a series of contracts with the actual equipment manufacturers or through a contract with a "third-party" maintenance firm which in this situation would in reality be a "fourth party."

For purposes of this discussion, we shall define "manufacturer service organizations" as those organizations established and operated by actual equipment manufacturers for the sole purpose of providing needed maintenance and field support for the products sold by that manufacturer.

Manufacturer service organizations offer essentially all maintenance and field support services a user would need for any and all equipment produced by that manufacturer. This includes routine maintenance at the user's site, both on-site and manufacturer repair capability, replacement parts capability, national technical support, and, installation system design and system modification capability.

The larger manufacturer service organizations have built up many years of experience; many of their customers have dealt with the same service engineers over long periods of time and most have geographically well-distributed field service offices.

These service organizations are set up in a pyramidal form: the base consisting of a large number of geographically distributed field service offices, each staffed by one or more service engineers. If a supply of the most commonly needed replacements parts is not provided at the user site, it frequently is provided at this level. Above this base is a smaller number of regional offices, each primarily staffed by administrative personnel and a small number of technical experts, available on demand to local users. At the top level is the home office of the manufacturer service organization. Usually it is here that in-house repair services are available along with national technical support, major replacement parts supply, and systems design and modification services.

Users commonly expect the highest degree of technical expertise from manufacturer service organizations along with excellent response time and reasonable cost. A manufacturer service organization is not commonly expected to provide mixed-vendor equipment capability and, in general, it does not.

Independent, third-party maintenance organizations

"Third-party" maintenance contractors differ from manufacturer service organizations in two significant ways. First, they are independent of manufacturers in the sense that they are not adjuncts to an operation whose primary purpose is product manufacture and/or sales. Second, most offer mixed-vendor equipment capability.

In terms of a table of organization, most "third-party" maintenance contracting firms are set up in a manner similar to the manufacturer organization: a nationwide network of local field service offices staffed by customer engineers, supported by regional office and national headquarters technical and support personnel. A parts supply and distribution system exists that meets customer needs at all levels of the organization, along with a shop repair system.

The independent maintenance contractor performs a variety of services for his user-customers under contract. These services may include preventive maintenance, or planned maintenance services performed at specific time intervals based on engineering failure estimates (changing filters, for example); predictability maintenance, consisting of services beyond the normal preventive maintenance, provided for equipment that has shown an increasing rate of non-catastrophic failure suggestive of a worsening performance trend; and remedial maintenance or services required to restore failed equipment to operation.

Other services, broadly defined as field support services, include equipment installation, spare parts support, technical documentation, training for user personnel, and maintenance management services.

The independent contractor is expected to offer the same high quality service, prompt response time, national technical assistance, and reasonable cost that are expected of manufacturer service organizations. But in order to develop customers for his services, the independent contractor must offer customers some advantage over the alternatives. In some cases the deciding factor is cost. In others it may be more frequent or perhaps better quality service under certain circumstances; in still others, it may be a matter of response time.

Most often cited, however, is single-source capability for mixed-vendor systems, the one advantage manufacturer service organizations cannot offer. It is the independent contractor's mixed-vendor capability, too, which permits tailoring of services to each user's specific equipment and requirements.

HOW DOES THE INDEPENDENT CONTRACTOR PROVIDE SERVICES?

Most services are provided at the user's site by a customer engineer dispatched from a nearby field service office. Normally a customer engineer is assigned to each customer on a permanent basis. The customer engineer is trained and qualified to perform scheduled maintenance and also render the more routine types of field service support.

Whenever possible, arrangements are made for user-site storage of the most commonly needed replacement parts. The "independent" contractor also arranges to have other parts available at the user site when necessary, within an agreed period of time determined largely by the user's system needs. A period of 6 to 24 hours is typical.

Because the user's primary requirement is a properly operating system, many independent maintenance contractors offer temporary or permanent replacement parts in exchange for malfunctioning parts which are then brought to the contractor's service center for repair, overhaul, or sent back to the equipment manufacturer.

The local customer engineer is empowered to call in additional support whenever required from the various levels of his own organization or in rare circumstances from the manufacturer of the affected equipment.

Independent contractors, as do manufacturer service organizations, offer these services in a variety of combinations tailored to each user's needs. There are, however, two basic types of arrangements: the first provides for preventive predictability, and remedial maintenance services for a prime period of time, usually a normal work day; the second provides for preventive, predictability, and remedial maintenance on a 24-hour-per-day, seven-days-per-week basis. Most users for whom any downtime poses a critical problem prefer the latter basis, which is, of course, somewhat more costly.

INDEPENDENT CONTRACTOR OR MANUFACTURER SERVICE: WHICH IS MORE EFFECTIVE?

It is impossible to give a definitive answer to this question that would apply in all cases. It must be answered on an individual basis by each system manager based on the nature of his system and its specific maintenance and field support needs, his previous experience with any of the alternative sources, and, most important, his determination of which alternative could best fulfill his anticipated requirements.

In all fairness, it should be pointed out that a majority of both government and private-sector computer systems have been and continue to be maintained under contract with manufacturer service organizations. Most (but not all) single-vendor systems are maintained under this type of arrangement. And many mixed-vendor systems are still maintained in this way.

No doubt this is due in part to the fact that independent contractor maintenance is still in its infancy, relatively speaking, and that manufacturers often include maintenance services at attractive rates as part of their lease or rental plans. There are but a few independent contractors offering full mixed-vendor system capability on a nationwide or worldwide basis. But the independent contracting sub-industry is growing rapidly. Some projections estimate the market for maintenance and field support services to exceed $1.3 billion annually by 1980. The same projection estimates that nearly 25 percent of this market may belong to independent contractors by that time.

In deciding how to best meet his maintenance and field support requirements, the system manager must first determine what those requirements are. Does he, for instance, require "around-the-clock" support capability? What is the maximum response time he can allow? What frequency of programmed maintenance does he desire? And so on.

After eliminating from consideration those alternatives which cannot meet his requirements, the determination should be based on other factors including estimations of quality of service, convenience, and cost.

Again, assessments of quality of service and cost will be highly specific to the system or user and, in part, based on a series of trade-offs between the two. The one generalization that can and does apply in all cases involving mixed-vendor systems is that independent contractor maintenance is more convenient. Independent contractor maintenance, implies a single contract to administer, a single programmed maintenance schedule, a single source to deal with for virtually all maintenance and field support needs, as opposed to multiple sources, and multiple schedules.

While this is not alway's the deciding factor, it is a very important one for many system managers with large and complex systems. It is becoming increasingly common to find a system involving products of six or eight different manufacturers and located at dozens of sites throughout the country. It can readily be seen that providing adequate maintenance and field support service for this kind of system under several contracts with different manufacturer service organizations could prove to be extremely time-consuming and costly from both administrative and logistical points of view.

A COMBINATION: THE FOURTH ALTERNATIVE

There are circumstances under which a system manager might obtain maintenance and field support services from both independent contractors and manufacturer service organizations. This situation arises when a user prefers to use a single-source independent contractor, but cannot find one capable of meeting all his needs. Typically, some equipment is located in an area not served by an independent contractor or the contractor is unable or unwilling to service certain items of equipment in the system. Because of the costs involved in personnel training, parts acquisition, and documentation, no independent maintenance contractor offers full capability for maintenance and field service support for even a majority of computer or computer peripheral equipment in use. More and more, however, the major independent firms are providing support for all of the more common equipment.

In situations of this kind, the user must out of necessity develop an arrangement involving both his preferred independent contractor and the appropriate manufacturer service organization or organizations.

CONCLUSION

Independent contractor maintenance and field support of computer and computer peripheral equipment has proven itself a viable and practical alternative for meeting the needs of data processing system users, particularly mixed-vendor system users. Today, roughly 12 percent of all such work is being performed by independent contractors on the simplest of single-vendor systems to the largest and most complex multi-vendor systems. Both government agency and commercial users have evaluated the alternatives and selected independent contractors for some or all of their maintenance and field support needs. In each year since 1971, independent contractors have increased their share of the U.S. market for these services, and this trend is projected to continue into the 1980's.

Independent contractor maintenance is encouraged as a matter of policy for certain government systems. It has proven effective and economical for public and private systems, large and small. It is now possible to say that there is now a mature alternative method of obtaining computer system maintenance and field service support services available to most computer users.

A perspective of standard form contracts in the data processing industry

by STEPHEN N. HOLLMAN
Attorney at Law
San Francisco, California

ABSTRACT

The data processing industry has been weaned on the standard form contracts of one dominant vendor and many derivative and close variants of clauses from those written agreements are now found in the standard form contracts of many vendors. As such, standard form contract clauses deriving largely from a single source may well control the rights and liabilities of wholesale numbers of buyers within this industry. In so doing, these contractual provisions act not as terms bargained for and assented to, but as a type of unchecked commercial legislation within the industry.

The trend is for such standard form contracts to acquire the attributes of adhesion (preprinted forms presented on a "take it or leave it" basis) producing an attitude of complacency in contracting among the management of buyer companies.

Although courts are loath to rewrite contracts among business participants, there has been an increasing trend of judicial disapproval of unconscionable contracts. In non-class litigation, such judgments affect only one standard form contract while the objectionable clause may have far broader impact thereby leading one very respectable law journal to espouse legislative intervention to correct this marked imbalance.

Management of vendee data processing companies should be encouraged and motivated to negotiate the terms of standard form contracts of adhesion to have those documents reflect the particular transaction affecting the business in their charge. Correspondingly, management of vendor data processing companies should be alert to the capacity of clauses in its standard form contracts to be oppressive or wreak surprise and of the circumstances under which these documents are presented to buyers either of which could bring about an untoward judicial result.

With these thoughts in mind, there is a further discussion of the often misunderstood or overlooked limitation of liability clause, "hell or high water" clause, most favored nations clause, integration clause, and arbitration clause, many of which are found in most standard form contracts.

INTRODUCTION

Dismal as the future may sometimes seem, it is not unreasonable to expect an increase in the volume of claims and disputes between contracting parties in the data processing industry as a by-product of continued growth and expansion within the industry. With this hypothesis in mind, one would do well to look at the nature of the contracts upon which such claims and disputes will undoubtedly be founded.

In executing agreements for data processing equipment or services, the management of a company which is a buyer should (but all too often does not) take cognizance of fundamental matters found in the contract which may likely affect its company's liability. A frequently heard reason for the lack of attention or inaction by such management is an anticipation of the vendor presenting its standard form of contract on an "accept this or nothing" basis and the concomitant judgment that an expenditure of effort to effect changes in a contract which is in general use by the vendor with its other customers may be futile.

Focusing upon such management attitude, this discussion will seek to place in perspective the use of standard form contracts in the data processing industry, highlight several obscure clauses, and conclude with certain recommendations to the management of both buyers and vendors regarding the use of such agreements.

THE STANDARD FORM CONTRACT

At one time, a contract in its most basic form was the product of a "bargain," i.e., it was the written expression of parties who had negotiated an agreement to exchange promises. With certain exceptions wrought by the law which are not here relevant, the "bargain" was the *sine qua non* of an enforceable contract for breach of which legal remedies would likely follow. However, a contract embracing the concept of the "bargain" no longer characterizes the preponderance of such documents which are today executed in the ordinary course of business.[1] Rather, there is the prolific use of the standard form of contract.[2] This type of written agreement has definite economic justification in that, among other things, it permits the vendor to select and control its assumed contractual risks, to exclude risks which are difficult to calculate, and to provide for situations arising from unforeseen contingencies such as strikes, floods, fires, labor disputes, and the like.[3]

Just as mass production and mass distribution have become indigenous to the data processing industry, so too has mass contracting using standard forms of written agreements. In the infancy of this trend, the data processing industry was weaned on the standard form contracts of one dominant vendor[4] and many derivative and close variants of clauses from those early written agreements are now found in the standard form contracts of many vendors.[5] Accordingly, standard contract clauses deriving largely from a single source[6] may well control the rights and liabilities of wholesale numbers of buyers of goods and services in the data processing industry. In so doing, these contractual provisions act not as terms bargained for and assented to, but as a type of unchecked commercial legislation within the industry.[7]

In executing a standard form contract for data processing equipment or services which has been presented on a "take it or leave it basis," that which is signed may in the eyes of the law be known as a contract of adhesion.[8] A contract of adhesion, simply stated, is one which is presented by a party in a position of superior bargaining power, usually pre-printed in great numbers, and offered on an "accept this or nothing" basis.[9] Standing alone, a standard form of contract drafted by a vendor to limit its risks and exposure is not, in and of itself, one of adhesion. However, when presented on a "take it or leave it" basis and pre-printed usually in smaller point type than the normal office typewriter to discourage attempts at typewritten changes, it acquires the attributes of adhesion.

With contracting in the data processing industry characterized by the extensive use of identical or similar clauses usually presented in written agreements under circumstances of adhesion, it appears not indelicate to observe that such written agreements may control more behavior than do the ordinances of many municipalities.[10]

Historically, courts have been loath to re-write contracts or portions thereof between business participants.[11] The theory is that businessmen as such are viewed by the courts as situated on equal footing[12] with each other unless the contrary is proved,[13] and this remains true despite the fact that the less well-informed or advised businessman did not read nor understand the contract.[14] The modern trend, however, is to provide a judicial remedy[15] if the court finds the contract or certain terms contained within it to be unconscionable.[16]

In private, non-class[17] litigation, the judicial nullification or reformation of a standard contractual term which may be pervasive throughout the industry benefits only the party to that suit and not all others similarly situated who are subject to such a contractual term. This seeming inadequacy of judicial relief has spawned an articulate suggestion[18] for legislative intervention to mandate minimum standard contractual provisions in all cases. If this were to come to pass, concerns of equity would override concerns of a freely functioning market and foster what many may find to be an unwelcome solution.

Data processing management is the freely functioning market force which, by tempering its attitudes of futility toward effecting changes in and consequent complacency for standard form contracts of adhesion, can correct an imbalance in contracting within the industry. Charged by emerging trends in the law both with protecting the interests of the business committed to its care and with refraining from depriving that business of the advantage that its skill or ability might properly bring to it,[19] management may find personal liability ascribed to it by an enterprising attorney in litigation which could arise from standard form contracts of adhesion executed in the comfort of complacency and not in the best interests of the business.

Management of a company which is a buyer should be encouraged and motivated to negotiate the terms of standard form contracts of adhesion although there will be transactions where this is not possible.[20] Recognizing that a vendor wants new or continuing business as much as the buyer may need the vendor's goods or services, the management of a buyer company assertively seeking to change standard terms in the vendor's standard form contract may well be surprised at the receptiveness of the vendor to engage in negotiations if not yield altogether to the buyer's demands.

Correspondingly, management of a company which is a vendor should be alert to potentially oppressive clauses in its standard form contracts and the manner and circumstances under which these documents are presented to its customers, mindful of the increasing trend of courts to find standard contract terms unconscionable[21] and the financial trappings of litigation. One simple approach when the vendor's risk appears to significantly enlarge by a modification of terms in its standard form contract would be for the vendor and buyer to agree that the buyer will pay a modest price increase just as one might pay a higher insurance premium for increased risk to the insurer and increased coverage to the insured.[22]

This discussion cannot possibly encapsulate all of the considerations of contracting within the data processing industry. However, if management or its counsel wish to consider a path of implementing the foregoing thoughts and desire reference materials on data processing contracts, there are several good sources.[23]

The balance of this discussion will deal with several clauses usually found in standard form contracts the significance and consequence of which are oftentimes not understood or grossly understated.

LIMITATION OF LIABILITY CLAUSE

A vendor typically limits its liability to financial compensation for damages suffered directly by its customer exclusive of the claims of third parties, and then only for the amount paid by the customer under its agreement with the vendor.[24] In addition, in the case of a services contract, the vendor may seek to insure itself against the prospect of monetary liability altogether by offering to duplicate the services which were not rendered or which were rendered in a faulty manner.[25] It does not seem at all unreasonable to limit the vendor's liability only to damages sustained directly by its customer, exclusive of the claims of third parties, but the further limitation of the customer's recovery to an amount not to exceed the fees paid under the contract may well be insufficient to compensate the customer for its losses and, indeed, may bear no reasonable relationship whatsoever to the customer's losses.[26]

Consider a union payroll which is produced twenty-four hours late by a vendor of data processing services. In many construction industry union contracts, a penalty of time and a half for each hour the payroll is late is imposed upon the employer (in this case the customer). Accordingly, the customer's actual damages in this example may far exceed by multiples the amount it has paid under the contract.[27]

The failure to fully understand and appreciate the consequences of standard form limitation of liability clauses can be seen in the following excerpts from a recent address by Richard L. Bernacchi[28] to the Computer Law Association.[29]

"Commercial risk allocation through . . . limitations of liability . . . presuppose an ability on the part of both parties to the transaction to articulate [their respective expectations] and understand the risks inherent in that transaction. . . . In the simple commercial transaction the distinction between the desired ends and the risks that are inherent in the transaction is relatively easy. . . . [However,] risk allocation in the data processing environment shares a problem which is endemic to all legal documents. What appears simple and easy to apply in the abstract, becomes muddied and difficult to apply in the concrete situation. . . . Many data processing transactions lack the . . . ability of both sides to foresee the risks inherent in the transaction. . . ."

". . . to the extent that the subject matter of a particular data processing contract involves risks that the vendee will normally not be able to foresee and evaluate, and to the extent that the vendor fails to articulate those risks, there can be no bargained for allocation of risks, notwithstanding the fact that both parties are presumed to understand the meaning and implication of the contractual language used to effect that allocation. . . ."

"While it can be assumed that the vendor understands or should understand the risk of distortion, no such assumption is justified with respect to the vendee. The proper index of vendee understanding should be the contract and the amount of specific and detailed allocation contained therein."[30]

It may not be a reasonable prospect to seek an enlargement of the vendor's limitation of liability.[31] However, it may be possible to reach agreement with a vendor to accept liability for consequential damages[32] despite the traditional boilerplate in which the vendor will seek to disclaim liability for consequential damages.[33] Particular inroads here may be made if the vendor's liability for consequential damages is conditioned upon a breach by the vendor or negligence by the vendor or its employees. In this regard, a bargaining point might be to further limit vendor's liability for consequential damages to compensate the customer by making it whole. Under these conditions, the vendor may be able to obtain insurance to cover its expanded liability.

"HELL OR HIGH WATER" CLAUSE

This provision is frequently found in lease agreements and essentially provides that the customer must pay under any and all circumstances. No matter what happens to the equipment, the customer-lessee still has to pay.[34]

One now finds such clauses slowly creeping into other types of contracts in the data processing industry where, in essence, it is provided that the customer pay first and complain later. To the extent that the customer is dealing directly with the vendor, as distinguished from a third party lessee, the clause might well be unconscionable.[35]

Here the customer should seek the contractual right to offset mandatory payments with claims it has against the vendor or, alternatively, to make disputed mandatory payments to an independant third party escrow pending determination of the claims of the customer against the vendor.

MOST FAVORED NATIONS CLAUSE

Many vendors in the computer industry contractually agree to make available decreases in price to all customers in much the same fashion as they pass on price increases to all customers.[36] This is called a most favored nations clause.

What some may view as a gift horse others may see as a two-edged sword. Where a most favored nations provision is a clause in a standard form contract, a price decrease to one customer must be offered to *all* of the vendor's customers in whose standard form contracts such a clause appears. Accordingly, there is a negative incentive for the vendor to offer a price advantage to one customer if it correspondingly becomes obligated to pass that advantage on to other of its customers.

The customer would be advised to seek to have its agreement for price concessions set apart from the standard form clauses and, whether within the content of the agreement or by side letter, effectively amend the agreement so that the performance thereunder of the vendor with price concessions is different from the performance and price normally offered by the vendor to its other customers. By so doing, the negative incentive of the vendor to bargain with a particular customer on price is mooted.

INTEGRATION CLAUSE

As surely as the sun will rise tomorrow, the standard form contracts used in the data processing industry will contain a clause to the effect that the written agreement constitutes the complete and exclusive statement of the agreement between the parties superseding all prior and contemporaneous proposals, understandings, representations, conditions, warranties, and all other communications, oral or written, between the parties. There is much to commend such a clause both on behalf of the vendor and the customer provided that, and only provided that, the written agreement is, in fact, the complete and exclusive statement of the understanding between the parties. If, instead, the written agreement is the standard form contract used by the vendor and the undertakings of the parties are founded upon a relationship emanating from other understandings, the documents expressing those other understandings, whether responses to requests for proposals, communications between the parties, or otherwise, should be mandatorily incorporated by reference as exhibits with the standard form contract. Further, there should be a reference within the standard form contract stating which documents should control in the event of conflict between the written agreement and the appended exhibits.

ARBITRATION CLAUSE

Under modern systems of jurisprudence, the remedy for a contractual dispute is that of litigation in the absence of contrary provisions in the contract. As many are aware, this can be a lengthy, tortuous, and extremely expensive process. Moreover, in the data processing industry one has to consider that a judicial tribunal, whether a judge or a jury, may be incapable of understanding some of the technical niceties, complexities, and innuendoes inherent in a relationship founded on the furnishing of data processing equipment and services.

The remedy of arbitration has been utilized effectively to resolve business controversies for many years. "The past fifty years has seen arbitration growth spurred by legislation, a friendly judiciary, an over burdened court system, cost-conscious business executives, and perhaps, the realization that viable alternatives to litigation are possible in many commercial transactions."[37]

Arbitration, as a remedy, seems to have two predominant virtues. The first is that in a long-term business relationship subsidiary differences between the parties can be resolved in a speedy, expeditious, and not terribly hostile manner without affecting that long-term business relationship. Second, it can bring to bear technically competent fact-finders to resolve a dispute between the parties going to the heart of the contract which may involve matters of such esoteric dimension that a lay judicial tribunal could not possibly be expected to make intelligent findings of fact.

Both vendors and customers should be alert to the advantages of the use of arbitration and mindful also of the principal objection to arbitration in that it may tend to be too "arbitrary". This ill-founded notion seems to derive from experiences inaccurately related as to how an arbitrator may have misjudged the facts and therefore improperly decided a matter before him. Carefully drawn arbitration agreements, however, can provide for equitable methods of review by the arbitrator or arbitrators of their own decisions.

Care, however, should be taken in the drafting of arbitration agreements to have them conform to local law. For example, in California there can be no use of pretrial discovery except in a case involving injury to, or death of, a person caused by the wrongful act or neglect of another, unless specific provision is made therefor within the arbitration provision.[38] In Texas, a provision for arbitration is not binding unless the agreement has been executed by counsel for the signing parties.[39]

CONCLUSION

Management of vendee data processing companies who are presented with standard form contracts from data processing vendors should be attentive to the terms contained in such documents specifically as they do or do not relate to the proposed transaction at hand rather than complacently executing such forms with comfort, albeit ill-founded, in the notion that such documents apparently are satisfactory for all of the vendor's other customers.

Similarly, management of vendor data processing companies who use standard form contracts should be alert to the potential for oppressiveness in such documents and the manner in which they are presented, and not take comfort in the fact that similar contracts have been in general use throughout the industry with impunity.

Whether it be a judge, a jury, an administrative body, or an arbitrator who is charged with reviewing the written document which has been executed, the parties must realize that "their" transaction is set forth within the four corners of that document and any shortcomings in this regard are the fault of the parties and not that of the contract.

REFERENCES

1. Bolgár, "The Contract of Adhesion: A Comparison of Theory," 20 *Am. J. Comp. L.* 53, 55, 1972.
   ". . . the old forms of contract, based on individual bargaining and consent became altogether inadequate and, above all, time consuming, since mass marketing is predicated on mass contracting under which contractual provisions become uniform and standardized."
2. Kessler, "Contracts of Adhesion—Some Thoughts About Freedom of Contract," 43 *Colum. L. Rev.* 629, 631, 1943.
   "Once the usefulness of these [standardized] contracts was discovered and perfected in the transportation, insurance, and banking business, their use spread into all other fields of large scale enterprise. . . ."
3. *Id.*
4. 2 Computer Law Service App. §3-2b (Bigelow, ed. 1975).
5. *Id.* at App. §3-2a.
6. *Id.* at App. §3-2b.
7. Stedransky, "Unconscionability and Standardized Contracts," 5 *N.Y.U. Rev. L. & Soc. Change* 65, 1975.
8. The term as such was first used by Patterson, "The Delivery of a Life Insurance Policy," 33 *Harv. L. Rev.* 198, 222 (1919) and judicially first given recognition in *Henningsen v. Bloomfield Motors, Inc.*, 32 N.J. 358, 161 A.2d 69, 1960.
9. Saxe, "Contracts of Adhesion Under California Law," 1 *U. San Francisco L. Rev.* 306, 1967.
10. Stedransky, *supra.*
11. The doctrine of *caveat emptor* will apply where there exists some parity

or equality between the bargaining parties. *Jefferson Credit Corp. v. Mareno*, 302 N.Y.S.2d 390, 1960.

12. *Id.*
13. *Id.* at p. 394.
Due to the "lack of equality between the bargaining parties" plus clauses taking advantage of the lack of equality, the contract was unenforceable and unconscionable.
14. A. L. Corbin, *Corbin on Contracts* §107 (1952).
15. *United States v. Bethlehem Steel Corp.* 315 U.S. 289 (1942); see also *Clements Auto Co. v. Service Bureau Corp.*, 444 F.2d 169 (8th Cir., 1971), 2 CLSR 102, modifying 298 F.Supp. 115 (D. Minn. 1969), 2 CLSR 143; and *International Business Machines Corp. v. Catamore Enterprises Inc.*, 5 CLSR 1025, 5 CLSR 1060, remanded for new trial *Computer Law and Tax Report*, November, 1976, p. 7-8.
16. W. D. Hawkland, *Sales and Bulk Sales*, p. 23, 1958.
"The basic test of unconscionability is whether, in light of the general commercial background and the commercial needs of the particular trade or case, the clauses involved are so one-sided . . . under the circumstances existing at the time of the making of the contract."
Boucher, "Unconscionability: Uniform Commercial Code Section 2-302", 36 *Albany L. Rev.* 114, 141.
"The person claiming the unconscionability should be arguing that the purpose of the clause was to oppress or unfairly surprise. . . . A clause which would be perfectly conscionable and legal in other circumstances, will be held to be unconscionable if its only purpose . . . is to oppress."
17. Class litigation is the adjudication of the claims of multiple parties in one judicial proceeding rather than multiple plaintiffs going forward in separate suits. *Amalgamated Workers Union of Virgin Islands v. Hess Oil Virgin Islands Corp.*, 478 F.2d 540 (CA. Virgin Islands 1973).
18. Kornhauser, "Unconscionability in Standard Forms," 64 *Calif. L. Rev.* 1151, 1976.
19. W. E. Knepper, *Liability of Corporate Officers and Directors*, p. 1, 1969.
20. Kessler, *supra*, at p. 632.
"The weaker party in need of goods or services is frequently not in a position to shop around for better terms, either because the author of the standard contract has a monopoly (natural or artificial) or because all competitors use the same clauses. His contractual intention is but a subjection more or less voluntary to terms dictated by the stronger party; terms whose consequences are often understood only in a vague way, if at all."
See also *Blair v. Pitchess*, 5 Cal.3d 258, 275-6.
21. Hawkland, *supra*.
22. Saxe, *supra.*, at p. 320.
23. Bernacchi, R. L. and G. H. Larsen, *Data Processing Contracts and the Law*, 1974.
Bigelow, R. P. and S. H. Nycum, *Your Computer and the Law*, 1975.
Brandon, D. H. and S. Segelstein, *Data Processing Contracts*, 1976.
24. 2 Computer Law Service App. §3-2a, *supra*.
25. *Id.*
26. Sweet, "Liquidated Damages in California," 60 *Calif. L. Rev.* 84, 93, 1972.
Actual damages must still be proved, "but the designated sum [in a clause limiting liability] operates as a ceiling on the performing party's accountability."
"In standardized contracts . . . which are made by parties of unequal bargaining strength, the California courts have long been disinclined to effectuate clauses of limitation of liability which are unclear, unexpected, inconspicuous, or unconscionable."
Steven v. The Fidelity and Casualty Co., 58 Cal.2d 862, 879, 1962.
27. *Clements Auto Co. v. Service Bureau Corp.*, *supra*; cf. *Farris Engineering Corp. v. Service Bureau Corp.*, 1 CLSR 902, affd. 1 CLSR 905.
28. See Footnote 23.
29. Address by Richard L. Bernacchi titled "Proper Allocation of Risk in Data Processing Contracts" to the Computer Law Association, March 3, 1976.
30. *Id.*
In Bernacchi's address, he offers the following examples:
"If the contract contained or incorporated by reference a detailed set of functional specifications, including response times, display formats and the like, then the inference is strong that either the vendee was aware of the risk of distortion and made certain that he actively participated in the technical articulation of his needs, or the vendor was concerned about the risk of distortion and attempted openly to articulate the system design in a manner calculated to allow the vendee the opportunity to recognize any distortion which may have occurred. If, on the other hand, the contract merely provides for a 'system' which contains certain hardware referred to by model number and certain software consisting of applications referred to by title, then the inference is strong that the risk of distortion was inarticulate at the time the parties entered into the agreement. . . ."
"A similar situation exists with respect to 'system' implementation, which usually consists of the delivery phase, the conversion of the data base and the existing applications, and the development of new applications. With new applications or with old ones that must be converted to new equipment, the implementation may turn out to be in the order of years rather than the months that the parties expected, or at least that the vendee expected. The particular difficulties in implementation may effectively change the vendee's expectation of the *purchase* of a computer 'system' into a contract for the *development* of that 'system'. If the contract either provides a specific time frame within which implementation is to occur, or specifically indicates that the amount of time required for implementation is unknown, then the inference of articulate risk and bargained-for allocation is strong. If the contract simply provides that the vendor will install the system, or that he will provide a certain amount of man-hours to that end, then the inference is strong with the risk of a lengthy implementation due to technological difficulty was inarticulate at the time the parties entered in the agreement. . . ."
"Similar problems exist in the area of system performance, which generally boils down to system acceptance tests, including both simulated and live tests. Even when you have a detailed set of functional specifications, it is the general nature of data processing 'systems' to exhibit inadequacies in live operations that do not appear in simulated testing. If the contract provides that the vendee's obligations are contingent upon successful system performance under acceptance test criteria which include live testing, then the inference of articulate risk and bargained-for allocation is again strong. If the contract simply provides for payment upon installation and certification that the system is 'operational', then the inference is strong that the risk of system inadequacy due to circumstances peculiar to the vendee's environment was inarticulate at the time the parties entered into the contract. . . ."
31. "There does not appear any way that the company can fairly price its services unless it does limit its liability in some way, because the efforts that it otherwise takes in order to protect against those anticipations of what the risks might be, will price the product right out of the market." *Aetna Casualty & Surety Co. v. Eastman Kodak Co.* 10 UCC Reporting Service 53, 56 (Dist. of Columb., 1972)
32. Hawkland, *supra*, at p. 139.
"Consequential damages are those which could not have reasonably been prevented by the customer and are for those injuries that the vendor had reason to foresee as a probable result of his breach when the contract was made. If the injury is one that follows the breach in the usual course of events, there is sufficient reason for the . . . [vendor] to foresee it; otherwise it must be shown specifically that the . . . [vendor] had reason to know the facts and to foresee the injury."
33. Hawkland, *supra.*, at p. 162.
The Uniform Commercial Code "recognizes the general validity of agreements limiting consequential damages . . . but it states that such agreements are to be tested in terms of unconscionability . . .", i.e., the limitation of consequential damages may be ineffective if such limitation is found to be unconscionable.
34. In *Leasco Data Processing Equipment Corp. v. Starline Overseas Corp.*, 74 Misc.2d 898, 346 N.Y.S.2d 288 (New York, 1973), the court upheld such a clause against the contention that it was unconscionable.
35. *Id.*
36. 2 Computer Law Service App. §3-2b, *supra*.
37. Aksen, "Legal Considerations in Using Arbitration Clauses to Resolve Future Problems Which May Arise During Long-Term Business Agreements," *The Business Lawyer*, January, 1973, p. 595.
38. The California Arbitration Law (California Code of Civil Procedure, §§1280 through 1295, specifically §§1283.05, 1283.1); Texas General Arbitration Act (Vernon's Ann. Civ. St., Art. 224 through 238-6, specifically Art. 224).

Small computers and small investors

by GEORGE KIM JOHNSON
Baton Rouge, Louisiana

ABSTRACT

The availability of low-cost hardware and the activity of turnkey developers has significantly expanded the number of potential computer users. Limited capital has been a major inhibiting factor in the growth of this marketplace. This paper proposes a method whereby small investors can use leveraged leasing to inject the capital necessary for expansion in this area. Investors may, after weighing potential risks and costs, obtain income as well as significant tax advantages. Developers can grow without requiring additional capital. Small end users can lease rather than purchase. Computer professionals can invest money, as well as time and effort, in their profession. Included is an analysis of a typical procedure for creating such an arrangement. Financing, tax analysis, contracting, and negotiations are specifically addressed.

The computing industry today is in a position similar to that of the automobile industry shortly after the turn of the century. We are witnessing a transition from an industry with its emphasis on technological marvels to a business based upon mundane, even household, uses of computing equipment. At the same time, we are observing the commencement of a transformation in the type of people involved in computing and the data processing industry. We are seeing the decline of the "computer priesthood" and the rise of a new entrepreneurial class. That is: the people who work with computers are beginning to invest their money, as well as their time and knowledge, in their profession.

Technological change is, or course, an important factor in this shift. For example, in the past few years we have seen the minicomputer market boom, the microprocessor market develop, and the leading edge of the personal computing phenomenon appear. So-called "mini" computers now have up to one megabyte of memory, and support data base management systems, high-level languages, and a wide variety of applications software. At the same time, the cost factors involved are dropping rapidly.

We could expect this change to be even more explosive were it not for certain limiting factors. Two of these are of immediate relevance. First, many of the developers of small systems (as well as potential users of these systems) have definite credit limits. Difficulty in obtaining capital, coupled with the traditional unavailability of leasing arrangements for small developers and users, inhibits growth to a considerable degree.

The second factor of importance is that the manufacturers of most of the smaller systems which would be suitable for financially limited users are incapable of providing "hand holding." Programming and maintenance support usually come from "local" system developers and turnkey system suppliers who are also limited financially.

This paper will present one approach which may be useful in surmounting these problems. It attempts to bring together the small investor, the OEM or turnkey system developer, the manufacturer of minicomputer systems, and potential users who, either through limited credit or concern over lack of expertise and technological obsolescence, do not or cannot purchase their data processing equipment. Essentially, this approach involves the use of leveraged leasing by an investor, who purchases a small computer and then leases it to a minicomputer turnkey system developer, who in turn will lease to an end user. Many variations of this arrangement are possible, but for discussion purposes, let us consider the ramifications of the set of agreements just described.

It should be noted at the outset that this opportunity may prove particularly attractive to small investors having a computer background, as they are in a unique position to spot opportunities for development. That is, they are likely to be aware of software/hardware package developments which are not effectively marketed due to a lack of financing. Furthermore, they are often able to understand and evaluate both the technological and marketing aspects of such a situation. Finally, it should be stressed that the benefits of this approach can apply to people who have a relatively small amount to invest and who are on fixed incomes. Tax shelters, for instance, are available to investors at any level, not just to the "wealthy."

Before looking at the specific procedures to be followed in setting up an arrangement of this type, let us quickly review some of the benefits, risks, and costs which are likely to be encountered.

There are two major benefits for an investor in a situation such as the one under consideration. First, and most obviously, there is an opportunity for income. Such income can provide a very useful supplement to a basic salary, since once the arrangements have been made, the expenditure of time by the investor is minimal. Second, a wide variety of tax shelter benefits may exist, depending upon the particular financial situation of the individual investor. It should be reiterated that these shelters are available to any investor, no matter what his level of financial involvement. Finally, although it was pointed out earlier that such an opportunity would be particularly appealing to an investor with a computer background, such knowledge is not essential. There is no requirement for particular expertise, nor for a large overhead. Of course, this is an advantage for the developer as well, since it considerably broadens his potential range of investment sources.

For the developer, it provides an opportunity to obtain the financing necessary to fully develop and effectively market his system. This is often quite difficult for the small software developer acting alone. Even for those already established as business entities, this method can provide an opportunity for expansion, after capital credit limits have been reached.

There are also benefits for the users. First, such an arrangement permits a user with a limited availability of capital to lease instead of purchase. It also provides flexibility, and enables the first-time user to get into data processing without having to acquire an in-house staff. It will also benefit all users in the long run, by encouraging the development of a greater number and wider variety of user-oriented software/hardware packages.

Finally, there will be benefits to the computer industry as a whole. It may be expected that increased investments at this level would encourage a considerable growth in the total number of users, particularly at the small end of the spectrum. It should furthermore encourage broader diversification within the manufacturing and software industries. Last, there should be an increase in the development of new application areas, since the availability of financing will encourage such development by individuals who were previously discouraged by the difficulty of financing the start-up of a small business.

As in any investment situation, there are risks and costs which must be recognized and evaluated. The costs are fairly obvious: some capital must be provided; in the case of leveraged leasing (that is, leasing by an investor who puts up a part of the capital outlay—say 20 percent to 30 percent of the total cost—and finances the remainder) there will be interest charges; there will be fees for the professional services of accountants, financial advisors and attorneys; and finally, there will of course be taxes to pay.

The major risks may be listed rather simply. They are: default by one of the parties; changes in tax regulations after the investment has been in place for some time; a failure on the part of one or more of the suppliers; problems of coordination between the parties involved; a failure to address all potential problems during the negotiating phase; to some extent, technological obsolescence (though this is less troublesome where the investor has a computer background); and unanticipated steep rises in costs (for example, for insurance or maintenance).

Let us now consider the basic procedures which would be followed in arranging an investment package of this type. It should be pointed out at the outset that as such an arrangement is developed, all of the factors should be individually explored, evaluated, and agreed upon in negotiations, prior to executing any contractual commitments relative to any single part. Furthermore, of necessity, the procedures outlined here are only general in nature. Each particular situation, depending upon the status of the individuals and parties involved, will require considerable "tailoring." It will be necessary in almost every case to obtain financial, tax, accounting, and legal advice prior to making any commitments.

The following analysis will be based upon a hypothetical situation structured as follows:

> Two system specialists have jointly developed an industry application package to operate upon a particular computer configuration. They have entered into an OEM agreement with the manufacturer of that hardware. They are prepared to undertake the marketing and maintenance of this package if they can obtain financing through a lease-back arrangement. The parties involved therefore will be the minicomputer manufacturer, the software developers, the investor, and the end-user.

The first step will be to consider financing arrangements. Naturally, it will be necessary for the investor to put up at least a portion of the purchase price in cash. Generally speaking, the minimum percentage will be ten percent, with more common figures ranging from 20 percent to 30 percent. Thus, for a typical minicomputer system in the $50,000 to $100,000 range, the investment amount required might be between $5,000 and $30,000.

In shopping around for a financing organization, the most important considerations on the part of the investor will probably be the interest rate and the schedule of payments. For tax reasons the length of the financing term may also be important. In some cases, it may be possible for the investor to take advantage of an OEM discount which would be available to the developer. For example, a developer might be able to purchase from the manufacturer at a discount and resell to the investor at that discount price. The financing organization may then take into account the market value (or list price) of the system in determining the percentage of the total investment which they will finance. Thus, an investor with ten thousand dollars might be able to obtain a $100,000 system if he received a 20 percent discount and put up his $10,000 in cash, thereby needing to finance only $70,000 of a "$100,000 purchase."

Of course, the business form of the investor (sole ownership, partnership, corporation) may affect the financing. Almost certainly the financing organization will require some sort of security interest, such as a mortgage on the property. They may also require the investor to assume personal liability for the loan.

Once the availability of financing has been determined, and the various options identified, it is necessary to undertake a tax analysis. As was mentioned earlier, such an analysis will vary with the financial status and requirements of the investor. There are in any case a number of general considerations which should receive attention in such analysis. (Keep in mind that this is an area in which expert advice is almost essential, since the complexity and rapid change of tax regulations makes effective analysis by lay people extremely difficult.)

There are several sources of potential tax advantage in an arrangement such as the one under consideration. First, where a portion of the purchase price has been financed, it may be possible to deduct the interest paid on the amount financed. Such a deduction is generally available, although there are certain significant conditions under which it may be limited.

Second, the Tax Reform Act of 1976 extended the availability of the investment tax credit. This credit, which is also subject to certain limitations as to amounts and availabilities, is a one-time credit of up to 10 percent of the total purchase price of the hardware. This credit is particularly advantageous if the investor has an income peak which he wishes to offset in the year in which he acquires the equipment.

Third, the equipment purchased (and in some cases a certain amount of associated software) may be depreciated. There are basically two types of depreciation applicable to personal property such as a minicomputer. The first is regular, or "straight-line" depreciation; the second is accelerated depreciation, which may take any of several forms. This is an area of considerable complexity, since accelerated depreciation is often treated as a tax preference item, and since determination of useful life can be a significant factor.

Finally, deductions are allowed for any legitimate business expenses and taxes (such as property and sales taxes) which may arise in the various transactions.

There are, on the other hand, some possible tax liabilities which must be evaluated. First, such an investment will in most cases generate income which will be subject to tax. Second, the equipment (and possibly the software in some jurisdictions) may be subject to property taxes or sales taxes. Of course, as was indicated, these taxes may be a source of deductions, but they must nonetheless be paid. Third, an improperly structured transaction may result in the recapture, at a future date, of taxes avoided through accelerated depreciation. Finally, if the arrangement is not set up properly, the Internal Revenue Service may find that the transaction is in fact a sale rather than a lease, and certain leasing advantages or tax advantages (e.g., the investment tax credit) may be diminished or lost.

The next step is the contracting phase. It is particularly important to remember that the purpose of a contract is not merely to serve as a tool for the resolution of disputes, should they arise. Rather, its most important purpose is to serve as a medium for the formalization of all the aspects of the working agreements between the parties. In a situation such as the one presently being considered, there will be multiple contracts. There will be contracts for: the original sale of the equipment; the sale of the hardware by the developer to the investor; the lease from the investor back to the developer; the lease from the developer to the end user; financing arrangements; insurance coverage; and maintenance. Each agreement will have some unique requirements. There are, however, some general principles which may be applied in all of them.

One of the most important considerations in structuring the agreement of sale is insuring that advantage is taken of all possible tax benefits. Specifically, responsibility for sales taxes, and pass-through of investment tax credits (if appropriate) should be delineated. Contracts of sale should also spell out very clearly the responsibilities of the parties, and any liabilities which may attach. For example, the time at which title transfers should be specified. Responsibilities for maintenance, delivery costs, and documentation should be enumerated. Warranties should be clearly specified, as should effects of liens, and provisions for failure to deliver. It is often desirable to include guarantees of performance and certifications of originality. Of course, detailed hardware specifications and configurations should be attached to, and made part of, the formal agreement.

As the owner of the equipment, the investor must enter into certain peripheral agreements himself, and must insure that other parties also fulfill their responsibilities in such matters. This is particularly true in the case of insurance and maintenance agreements.

Of course, the investor himself will be responsible for any financing contracts which he may enter into. These agreements will probably provide a security interest in the equipment on the part of the financing organization, and therefore may have to be referenced in other contracts.

The next agreement which should be considered is the lease to the developer. It should be pointed out that it is quite possible for a developer with adequate capital to form his own organization to purchase the computer, and then to lease that computer to a separate business entity of which he is a part, which would handle marketing and maintenance. A common arrangement in this regard is for a group of developers to form a partnership, which purchases equipment for lease to a corporation, the stock of which is held entirely by the same individuals who comprise the partnership.

Other contract provisions should define responsibility for the installation of the system, and for the maintenance of the software and the hardware. Provision should be made for upgrading hardware and software components, for insurance coverage, and for payment of other costs such as shipment. The lease contract, like those relating to the original sale, should address any applicable warranties, including guarantees of performance, guarantees of originality, and provisions for the lessee to be an attorney-in-fact of the lessor in relationship to the vendor, in order to obtain advantage of any vendor warranties.

It is also desirable for the lessor to insure that he receives treatment at least equal to that accorded any other customer of the developer. It is quite possible that the developing organization may also develop or market other software packages, and deal with other investors. It is, therefore, important for each investor to insure that a more profitable arrangement does not displace him in leases to end users.

It is also essential to consider what happens upon the termination of the agreement. There are three types of termination which must be dealt with. The first of these is a temporary termination. This occurs in the case where an end user (for any reason) terminates his agreement. The responsibilities of the developer as lessee and the investor as lessor must be clearly specified, particularly with regard to who will bear the burden of carrying the cost of the equipment until it can be replaced in another end user organization. Second, provisions must be made for the orderly, planned termination of the contract at the end of its specified term. This will be influenced considerably by tax considerations. For example, the lessee may wish to have the option to purchase the equipment at the end of the lease.

Finally, attention must be directed to the eventuality of an unexpected business termination by any of the parties. This is particularly important in the case of the developer, since the termination of the developer as a business entity could severely affect the investor's ability to continue supporting installed end users. This is especially true in the case where the investor is not knowledgeable in the computer field, or where the investor does not have the time to market and maintain the equipment. At the very least, provision should be made for escrow of the source code and the documentation of the application programs.

In the case where multiple business entities are involved (that is, where the developing organization is a subsidiary of a larger organization), it may be desirable to include provisions for cross guarantees.

In addition, there are a number of other standard contractual provisions which should be considered. Among these are limitations on assignments, methods of payment, and availability of financials of all the parties. These provisions can be fully explained and evaluated in each situation by legal counsel.

While the investor may not have a direct relationship with the third party or end user, he nonetheless should be aware of the contractual arrangements between the lessee and the end user, especially with regard to costs and liabilities in the event of termination. This agreement should clearly specify the rights, relationships, and responsibilities of the parties in such a way that the investor is reasonably protected.

The final area for consideration is that of negotiations. This is the phase in which all of the factors are put together into a workable arrangement. Prior to entering into negotiations, it is particularly important for each party to specify formally its own requirements and objectives. This is the part of the process in which the use of specialists such as CPA's or attorneys is particularly important. It should be reiterated that the entire package should be formalized before commitments are made with respect to any portion of it.

Just as in good system design, it is important that negotiations be both comprehensive and formal. Each factor should be carefully considered, and a specific agreement reached as to the terms relating to that factor. These should then be documented fully.

At first glance, such a project may seem somewhat complex. It is in fact a unique opportunity which is quite feasible if approached with care and understanding. The members of the data processing community are presented with an opportunity both for the development of new systems and markets, and for personal investments in a familiar business environment. By being watchful for development opportunities; by being aware of one's own resources, situation, and objectives; and by following the basic procedures outlined herein relative to financing, taxation, contracting and negotiation, small investors, system developers, and end users can obtain significant financial and business advantages.

The utilization of this approach can considerably increase the availability of computer systems to a wide variety of business organizations. It offers significant opportunities for entrepreneurs, developers, and investors. Finally, it provides a unique means whereby computer professionals can invest in their own profession.

Nondedicated interprocessor communications discipline

by DAVID J. BASTYR
Recognition Equipment Incorporated
Irving, Texas

ABSTRACT

When designing an interprocessor communications discipline, the problem of controlling data transaction on the interprocessor communications links has been traditionally solved by designating one processor as the central controller (the Master). All other processors are designated Slave processors which cannot utilize the interprocessor links unless requested by the master processor.

An alternative approach is to allow the interprocessor link to determine the Master/Slave relationship based on which processor has requested and received control of the link. An interprocessor link is in the neutral state (neither processor is Master or Salve) until a request for control is received from one of the processors. This processor is notified via hardware interrupt that it is the Master processor. Conversely, the other processor is notified that it is the Slave processor and is also informed of the nature of the pending transaction. When the interprocessor transaction has been completed, the Master processor releases control of the interprocessor link. The Slave processor is then notified that the interprocessor link is neutral again. Either processor can now request control of the interprocessor link.

INTRODUCTION

This paper describes an interprocessor communications discipline in which all processors in the interprocessor network have an equal opportunity to gain control of the interprocessor communications links. The interprocessor links in this system are Direct Memory Access (DMA) channels which are capable of transmitting buffered or unbuffered data. Traditionally, the problem of controlling data transmissions on an interprocessor channel has been solved by making one of the processors responsible for initiating all interprocessor transactions. This processor is referred to as the Master processor. All other processors in the interprocessor network are Slave processors that cannot utilize the interprocessor communications links unless requested to do so by the Master processor.

The nondedicated interprocessor communications discipline permits the master/slave relationships for each interprocessor channel to be determined at the time a specific interprocessor transaction is being initiated.

This technique requires a hardware unit in the interprocessor link which responds to requests for control of the interprocessor communications link and from which the current status of the link can be determined. A set of software modules comprising the Interprocessor Supervisor provides the interface between the interprocessor unit and the application programs.

INTERPROCESSOR UNIT (IPU)

The Interprocessor Unit (IPU) resides between the processors DMA channels as illustrated in Figure 1. Each interprocessor link requires an IPU. The IPU responds to the commands listed in Table I by activating channel interrupts and setting the appropriate IPU status conditions listed in Table II. The IPU command format is illustrated in Figure 2. Two state controllers determine the current status of the IPU. The Master/Slave Controller (Figure 3) determines which processor has control of the interprocessor link. The Transfer Controller (Figure 4) sets the operational sequence required for an interprocessor data transfer.

INTERPROCESSOR CHANNEL CONTROL

A processor must gain control of the interprocessor channel before it can initiate an interprocessor transaction. If the channel is free, the IPU status will indicate that "Neither Processor has Control." The IPU Master-Slave Controller (Figure 3) is in state X. If the request for control has been successful, the IPU status will indicate that "This Processor is Master." Conversely, the IPU status on the other side of the interprocessor channel will indicate that "This Processor is Slave." The IPU Master-Slave Controller returns to state X when the Master processor executes the IPU command to Release Control.

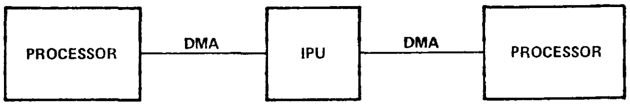


Figure 1—Interprocessor unit (IPU)

TABLE I—Interprocessor Unit Commands

Processor Requests Control
Processor Releases Control
Initiate Single Word Transfer
Initiate Block Output Transfer
Initiate Block Input Transfer
Send End-of-Transmission

INTERPROCESSOR TRANSACTIONS

This interprocessor communications discipline consists of three different types of transactions:

- Single Word Transfer
- Block Input Transfer
- Block Output Transfer

The IPU must be in state A when an interprocessor transaction is initiated and must be returned to state A when the transaction is completed (see Figure 4).

SINGLE WORD TRANSFER

The Single Word Transfer can be used to transmit a coded message which can be entirely contained in a single word. In Figure 4 the transaction in the IPU transfer controller is A-B-C-A.

After obtaining control of the interprocessor channel, the processor executes an IPU command to Initiate Single Word Transfer. When the IPU acknowledges the command, the IPU then activates an interrupt at the Slave processor. While processing the interrupt, the Slave processor must execute an instruction to input the data word and then execute the IPU command. This command then sends an End-of-Transmission status to the Master processor thus completing the Single Word Transfer. The IPU activates an interrupt at the Master processor and becomes ready for the next transaction. At this time the Master processor may initiate another transaction or release control of the interprocessor channel.

BLOCK INPUT TRANSFER

In Figure 4, the transaction in the IPU transfer controller is A-B-C-D-A. The Block Input Transfer can be used to transfer a data block from the Slave processor to the Master processor. The Master processor executes the IPU command to Initiate Block Input Transfer. The IPU responds to the command by activating a Slave interrupt. The Slave processor responds to the block input request by preparing the data for transmission and then executing an instruction to initiate a block output transfer. When the block transfer has terminated, the Slave processor executes the IPU command to Send End-of-Transmission to the Master processor, which completes this transaction. Again, the Master processor has the option to initiate another interprocessor transaction or to release control of the interprocessor channel.

TABLE II—Interprocessor Unit Status

Vertical Parity Error
This Processor is Master
This Processor is Slave
Neither Processor has Control
End of Transmission
Block Transfer
Block Transfer from This Processor

IPU COMMAND CODE	TRANSACTION CODE

Figure 2—IPU command format

BLOCK OUTPUT TRANSFER

In Figure 4, the transaction in the IPU Transfer Controller is A-B-C-E-C-A. The Block Output Transfer can be used to transfer a data block from the Master processor to the Slave processor. The Master processor executes the IPU command to Initiate Block Output Transfer. The IPU responds to the command by activating the Slave interrupt. The Slave processor responds to the block output request by readying an input buffer and then executing an instruction to initiate a block input transfer. When the block transfer has terminated, the Master processor executes the IPU command to Send End-of-Transmission to the Slave processor. The IPU activates the Slave interrupt which the Slave processor acknowledges with the IPU command to Send End-of-Transmission to the Master processor completing this transaction.

INTERPROCESSOR SUPERVISOR

The Interprocessor Supervisor provides the software services necessary to maintain an orderly flow of interprocessor transactions. The Interprocessor Supervisor also has the capability to manage several interprocessor links simul-

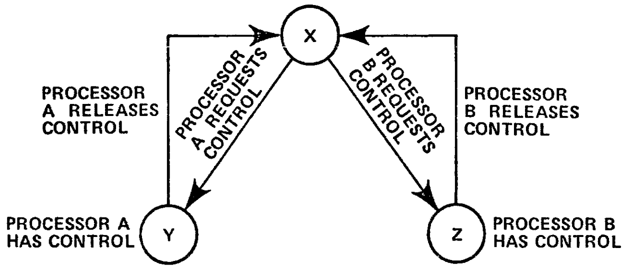

X
PROCESSOR A REQUESTS CONTROL
PROCESSOR B REQUESTS CONTROL
PROCESSOR A RELEASES CONTROL
PROCESSOR B RELEASES CONTROL
PROCESSOR A HAS CONTROL
Y
Z
PROCESSOR B HAS CONTROL

Figure 3—Master/slave controller

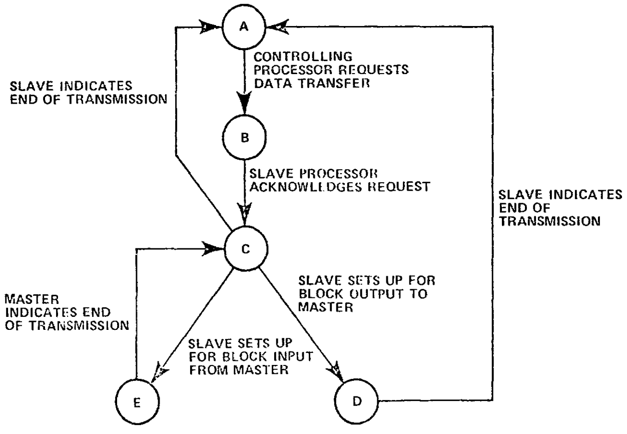

A
CONTROLLING PROCESSOR REQUESTS DATA TRANSFER
SLAVE INDICATES END OF TRANSMISSION
B
SLAVE PROCESSOR ACKNOWLEDGES REQUEST
SLAVE INDICATES END OF TRANSMISSION
C
SLAVE SETS UP FOR BLOCK OUTPUT TO MASTER
MASTER INDICATES END OF TRANSMISSION
SLAVE SETS UP FOR BLOCK INPUT FROM MASTER
E
D

Figure 4—Data transfer controller

taneously and to provide for both foreground and background interprocessor communications. Interprocessor transactions may be initiated by foreground events from their interrupt processors or from background tasks. A foreground to foreground transaction is accomplished through specific application-oriented extensions of the IPU interrupt processing.

The Interprocessor Supervisor provides general input-output services for queuing caller requests, executing IPU commands, processing IPU interrupts, and processing transactions errors.

CONCLUSIONS

The nondedicated interprocessor communications discipline was developed for a multiprocessor application involving the distributed processing of high-speed asynchronous real time events. A single event might involve the execution of routines in several processors and therefore necessitate several interprocessor transactions. Whereas this method of interprocessor communications can be used in multiprocessor networks executing batch-type applications, the more conventional dedicated Master-Slave interprocessor communications is ordinarily sufficient.

An approach to address identification from degraded address data

by VIRESH SETH

Recognition Equipment Incorporated
Irving, Texas

ABSTRACT

Today's Optical Character Recognition (OCR) technology does not read characters with 100 percent accuracy. Thus, the data string read by OCR may have one or more of the following deficiencies.

- Unrecognizable characters
- Incorrectly read characters
- Added characters
- Missing characters
- Subclass characters

This degradation then poses special problems in performing contextual analysis on address data strings comprising the identification of relevant address elements and the comparison of these address elements with entries in an address directory. Having some knowledge of the nature of the data, context analysis first attempts to correct one or more of the above mentioned deficiencies. Next a search for known keywords such as street designators (Avenue, etc.) is performed on the data. Finding keywords helps identification of the position of the other address elements and the determination of the type of address element. A search of the relevant files in the address directory then yields a definite address identification. However, due to the degraded nature of the data, keywords either escape detection or are erroneously found which creates confusion in contextual analysis.

The comparison of the address data with entries in the directory is performed in the hardware primarily because of real time considerations. The algorithm used is based on a "weighting" technique which compensates for the deficiencies in the data. Generally, the algorithm is successful in reducing the comparison of ten or less potential candidates from the directory. Then, contextual analysis in the software attempts to isolate the unique candidate yielding the correct sort information for this mail piece.

INTRODUCTION

Optical Character Recognition (OCR) technology creates a set of unique problems that must be dealt with. One of these problems evolves from the fact that the characters established by today's character determination algorithms are not totally reliable. The reliability of the data depends not only upon the sophistication of the character reading algorithms but also upon the quality of the printed characters. This problem is further aggravated if the character reader is attempting to process several or all of the various type fonts that are used by today's typewriters, line printers, etc. However, the problem of OCR processing with unreliable character data varies in severity with each application.

In a mail sorting application, the OCR challenge is probably the greatest. The envelopes on which the addresses are printed have a great variety of print quality characteristics, paper color, and font type. Therefore, the address recognition algorithms which process the address and determine a destination for the respective mail piece must be cognizant of the fact that character data may range from totally correct to grossly inaccurate.

SYSTEM OVERVIEW

An overview of the OCR system used to sort French mail pieces is illustrated in Figure 1. Mail pieces are fed into the mechanical transport by the Mail Feeder at rates up to twelve letters a second. As the envelopes pass the Optics Modules an image of the address is digitized and temporarily stored in the Mass Storage device. Each line of the address data is then presented to the Character Reader which converts the digitized image into characters and outputs the resulting address string to the computer. The address recognition algorithms in the computer then compare the address against an Address Directory stored in the Address Recognition Unit (ARU). The resulting destination code of the respective mail piece is then sent to the Mechanical Transport. The transport then routes the mail piece into one of the sort pockets based upon this destination code. The address recognition algorithms are logically divided into the string comparator and context analysis. The string comparator verifies the address string by comparing it with the various entries maintained in the address directory. Since this process is very time consuming, this function is implemented in the Address Recognition hard-

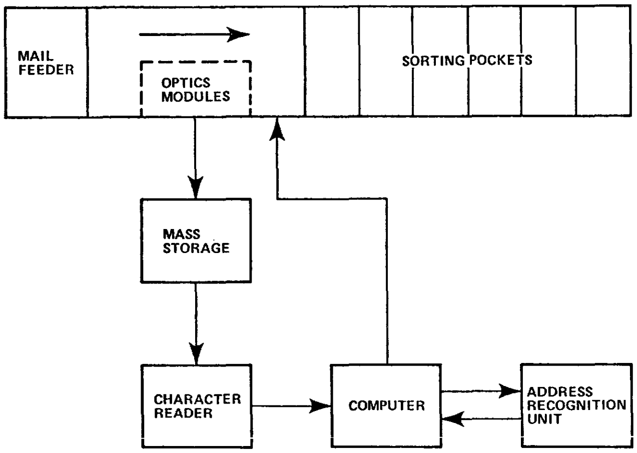


Figure 1—Optical character recognition mail sorting system

ware. The context analysis portion of the algorithm isolates logical components of the address. As this function is format dependent, it is best suited for implementation in the software. This division of processing has the further advantage that while contextual analysis is being performed on one mail piece in the address recognition software, the ARU can be performing comparisons on another. This parallel action results in faster processing speeds.

THE PROBLEMS AT HAND

The degradation in the address character data that is input into the computer may be classified into five categories. For example, consider the following address string in its correct form.

75015 PARIS

In this address string, 75015 is the ZIP code and PARIS is the city name.

(a) The Character Reader may be unable to recognize certain characters and the string may be received as:

75015 PAR??

where the question marks indicate the unrecognized characters. This type of degradation results in confusion in the city name being recognized as either PARIS or PARLY

(b) The second possible degradation in the address string is incorrectly read characters. The same string may appear as:

75015 PAPLS

where the characters RI have been incorrectly read as PL

(c) The third possible degradation is reading a character when no such character exists in the address. This string may be received as

75015 PARIIS

(d) The fourth category is missing characters which may make the same string read as:

75015 PARS

(e) The fifth type of degradation is a subclass. This situation occurs when the Character Reader is unable to uniquely identify the character but isolates it down to two to five characters. In the following address string:

75(DOØ)15 P(A4)RIS

DOØ is the subclass determined by the Character Reader for the character O and A4 for the character A.

It is not difficult to imagine what the address string will look like if all five possible types of degradation appear in the same address string. Therefore, the problem then becomes, to determine the correct city name from a character string that may be either totally correct or may have any combination of the five previously mentioned degradations.

COMPARISON ALGORITHMS

The comparison algorithms are implemented in the ARU comparator unit. The address directory containing the names of all the cities, streets, building names, and other relevant sort information is maintained in the ARU mass storage unit. The comparator reads entries from the directory (one at a time) and compares them with the address string received from the address recognition software. This comparison is performed with every entry in the specific directory record or file. At the end of each comparison operation the best directory matches are returned to the address recognition software.

The first step in determining the correct directory match is to minimize the number of comparisons to be performed. To accomplish this task, the directory is arranged into several files, each file being further subdivided into records. A sample organization of the directory follows:

a. ZIP code file
b. City file
c. Street file
d. Building name file
e. Keyword file

The ZIP code and the city files are further subdivided into records based upon the first two digits of the ZIP code, while division in the street file is based upon the geographical zone within the city. The decision on the file to be searched (or more specifically a record within a file) is made by the context analysis algorithms in the address recognition software.

The comparison of an entry from the directory (from hereon called the *memory string*) and the string read by the Character Reader (from hereon called the *read string*) is performed by means of the weighting algorithms described in the following procedure.

1. The memory string must not be longer than the read string. No comparison is performed with the current memory string if this criterion is not satisfied.
2. The comparison is performed on a character-by-character basis, starting with the rightmost character and proceeding to the left.
3. A score of six is given for each character that matches.
4. No score is added or subtracted when the character in the read string is "unknown."
5. A score of three is given for each subclass character matched.
6. On a mismatched character a left angle comparison is performed, i.e., the mismatched character of the memory string and the next character of the read string are compared. If they are found to compare, then the remaining parts of the two strings are positionally realigned so the two characters matched by an angle comparison are in the same relative position.
7. If a left angle comparison fails, then a right angle comparison is performed in a similar manner.
8. Two points are added for each angle comparison successfully performed and no more than two angle comparisons are allowed for each memory string.
9. Next the Normalized Match Score is calculated by the following formula:

$$\text{NMS}=\frac{(\text{AMS}*100)}{(6*\text{CC})}+(2*\text{CC})$$

where:

AMS=Actual Match Score (Score calculated by the comparator)

CC=Character Count (Number of characters in the memory string)

This is the final score that is assigned to the current memory string.

The comparator algorithm will be best understood by means of an example. Consider the following:

D A L L A S T E X A S — Memory String

P ? L L A X S T E X (S5) — Read String

The parens indicate the S and 5 form a subclass, while the question mark indicates an unknown character. Notice that all the five types of degradation are present in this string.

- The D in Dallas has been incorrectly read as P
- The A in Dallas has not been recognized by the Character Reader
- An X has been erroneously inserted
- The A in Texas has not been read
- A subclass 'S5' has been read for S

The comparator first matches the 'S5' subclass with S and gives a weight of three. Next, the X from the read string is matched with the A in the memory string. Since these two characters do not match, a left angle match is performed which yields a match. Therefore, a weight of two is added for the angle comparison. The remainder of the string will now be aligned as follows:

D A L L A S T E X. . . . . Memory String

P ? L L A X S T E X. . . . . Read String

A weight of six will now be added for each of the matched characters E, T, blank, and S. The X and A will not compare and a left angle comparison will be performed, i.e., the X will be compared with the L. Failing this comparison, a right angle comparison will be performed, i.e., an X with an S. Failing this, a weight of six will be subtracted from the total weight and the next character comparison performed. Since the A will not match the L, a left angle comparison will be performed and since this will not match a right angle comparison will be performed. As this will be successful a weight of two will be added for the angle comparison. Now the remainder of the string will be aligned as follows:

D A L L A. . . . . Memory String

P ? L L A. . . . . Read String

A weight of six will be added for both L's, no weight will be given for the unknown character while the mismatched characters D and P will cause a weight of six to be subtracted from the total.

This comparison technique will yield an accumulated match score of 31. The length of the memory string is 12; therefore, the Normalized Match Score will be:

$$\text{NMS}=\frac{(31*100)}{6*12}+(2*12)=67$$

Note that the left angle comparison technique adjusts for the characters that were not read by the Character Reader while the right angle comparison takes care of characters erroneously present in the read string. An unrecognized character does not add or subtract from the accumulated weight while an incorrectly read character contributes a negative weight. In searching a file, it is not necessary that the memory string which yields the highest Normalized Match Score is the correct address since the degradation may sufficiently modify the read string so that it matches better with another memory string. Unless a match yields a perfect score, or a match yields a score much higher than the second highest score, several memory strings are selected as the possible matches. Then it is the function of context analysis to isolate the correct address from these several possible addresses.

CONTEXT ANALYSIS

Contextual analysis is address format dependent. The procedure for addressing mail in the United States is different from the French or German addressing procedures, i.e. the physical location of the ZIP code, the city or street name and the specific keywords which aid in address identification are different for each country. For example a typical United States address would be:

DALLAS TEXAS 75006

while a typical French address would be:

75006 PARIS

The major differences are that in a French address there is no state specified and the ZIP code appears on the left. However, in spite of all the different formats the basic technique of contextual analysis is the same. This paper describes the contextual analysis used to process French Mail.

GENERAL CONTEXTUAL ANALYSIS PROCEDURE

The general contextual analysis procedure for an address without any degradation in the address will be described first. Then the specific problem cases will be addressed and techniques used to cope with them. Consider the following address:

78 AVENUE WASHINGTON

75006 PARIS

The street number is 78, WASHINGTON is the street name, 75006 is the ZIP code, PARIS is the city name and AVENUE is the street keyword.

As was mentioned earlier, one of the files in the directory is a keyword file. This file contains common words found on French addresses that provide some information about the address elements. When every word of the above address is matched against the keyword file, AVENUE will be identified as the street keyword.

This will then help identify WASHINGTON as the street name and 78 as the street number. On the bottom line no keywords will be found except the five character numeric word which will be identified as the ZIP code. Since the city name is the only other address element that appears on the bottom line, the remainder of the string other than the ZIP code will be identified as the city name.

Having identified all the address elements the next step is to compare them with the memory strings in their respective files in the directory and obtain a destination code for the mail piece. Since this address is not degraded, a single unique match will be found for the city and also for the street. On the city file search the ZIP code on the read string can be verified by comparing it against the ZIP codes for PARIS that are obtained from the directory. Similar verification can be performed for the street number.

CONTEXTUAL ANALYSIS OF DEGRADED ADDRESS DATA

The context analysis procedure is fairly straightforward as long as the data is not degraded. Let us first examine the bottom line and determine the various degradations that create the problem situations. The degradation in the city name is handled by the comparator algorithm described earlier. However, context analysis must process the degradations in the ZIP code which may be in one of the following forms:

a. Unrecognized characters . . . . 7500?
b. Added characters 750016
c. Dropped characters 7506
d. Incorrectly read characters 75005

Now the task of verifying the ZIP code for PARIS from the read string with those obtained from the directory is not straightforward. In the first case the technique employed is to treat the unrecognized character as an universal character, i.e., it is allowed to match any character. In the second case the technique is to remove one character at a time and to use the resulting five characters in the comparison. Thus, for the incorrect ZIP code 750016 the five ZIP codes used in the verification will be 75001, 75006, 75016, 70016, and 50016. It is likely that more than one of these five possible ZIP codes will match the ZIP codes in the directory for PARIS in which case an "unresolvable confusion" results. In such cases the mail piece cannot be sorted. The same technique can be used for the case of a dropped character. An unrecognizable character is added in each of the five possible positions and the resulting five ZIP codes are used in the verification.

The incorrect ZIP code on the read string occurs not only due to a reading error, but also due to an incorrect ZIP code being printed on the envelope. The problem is then to decide whether there are characters in the ZIP code that are in error or whether the wrong memory string was matched by the string comparator in the ARU. Thus, in comparing the ZIP code it becomes necessary to determine the number of characters that match and the number that do not match. Using the Normalized Match Score that was obtained from the city match a heurestic technique must be used to decide whether this is the correct match. The method is to establish minimum Normalized Match Score thresholds which are required for each combination of the number of mismatched and matched characters in order to accept the match. The higher the number of mismatched characters and the lower the matched character count, the higher will be the Normalized Match Score that must be required to accept the match. This graduated scale of Normalized

Match Score thresholds must be determined by trial and error in the particular situation.

An important note here is, when these thresholds are kept high, the percent of the incorrectly sorted mail pieces will be low but the percent of the correctly sorted mail pieces will also be low. On the other hand when the thresholds are kept low, then the sort rate will rise but so will the rate of missorted mail. Therefore, the tuning of these thresholds will depend on what is more important to the system: to keep the sort rate high or to keep the missort rate low.

The entry in the city file that is matched by the ARU comparator may have associated with it several ZIP codes. In this case the ZIP code comparison procedures described must be performed individually with each ZIP code. If more than one ZIP code satisfies the criteria, then an "unresolvable confusion" occurs, in which case the mail piece cannot be sorted. Further, several possible matches may be returned from the city file search, each of which may have one or more ZIP codes associated with it. In this case each ZIP code of each match must be similarly processed.

Considering the second address line, it was mentioned that the required address elements (street, building name, etc.) are identified by searching for keywords. For example: AVE (abbreviation for Avenue) in the following address line identifies WASHINGTON as the street name

78 AVE WASHINGTON

Therefore, it becomes important to recognize the keyword and herein lies the problem. The first of this set of problems occur when the keyword AVE is sufficiently degraded as not to yield a match when searched against the keyword file. The result of this is that no address element is identified on the address line. This forces the searching of all the files in the directory for an entry similar to the second address line. Depending upon the degradation in the street name, matches may be found in both the building name and the street file or only in the street file. When matches are found in both files, the only alternative left to identify the correct match is to look for a street number on the address line. In this case, since the street number 78 will be found, and knowing that no number appears on a building name address element, the match from the street file will be chosen.

The second problem that must be dealt with is the multiple matches on keywords. Consider the following address line:

78 ?(UV)E WASHINGTON

The avenue keyword has been degraded so a match of equal score will be obtained when the ?(UV)E is matched against the two street keywords in the directory AVE and RUE. Often the two keywords matched will be in conflict; therefore, identifying the same string as two different address elements. The problem then is the same as if no keyword were found. However, when both keyword matches identify the string to be the same type of address element, then the problem is solved unless two street entries exist in the directory of the same name, one with a RUE and the other with AVE. Again to resolve this confusion the street number must be used. If one of the streets has 78 as a legitimate street number, while the other does not, then the problem is solved. However, if both have 78 as legitimate street numbers, then once again an "unresolvable confusion" occurs.

The third type of problem occurs when the space between the words is either not read or is read as an unknown character. Consider the degraded string:

78?AVEWASHINGTON

in which the space between the 78 and the AVE has been read as an unrecongizable character and the space between the AVE and WASHINGTON not read at all. A keyword search on this string will not yield AVE nor will the 78 be recognized as the street number. The solution to this type of problem is to search all the files with the address line. For each match that is found the number of characters matched are stripped and the remainder of the string is sent for a keyword match. Thus, in the example WASHINGTON will match when searched with the street file. Now the string 78?AVE can be sent for a keyword search. On identifying AVE as a street keyword, 78 can now be established as a street number.

CONCLUSION

The techniques employed for address recognition when the address data is degraded are mostly heuristic in nature. It is not necessary for every address presented to the address recognition algorithms to be uniquely and correctly identified. The success rate depends upon both the simplicity of the possible address formats and inversely upon the amount of degradation in the data. However, it is possible to tune the algorithms to yield a high percentage of sorted mail pieces, but when this is done the percentage of incorrectly sorted mail pieces will also rise. Conversely, reducing the number of incorrectly sorted mail pieces will also reduce the sort rate. Thus, an acceptable trade off must be achieved between the two.

Signature and facial image compression by boundary encoding

by DAVID P. HIMMEL

Recognition Equipment Incorporated
Irving, Texas

ABSTRACT

Storage, retrieval, and transmission of signature and facial images is important to the problem of personal identification for monetary transactions or security. Compression of such images is necessary for efficient and economical storage and retrieval, and for fast transmission of the information. This paper explains a method, based on boundary encoding, of compressing signature or facial images by extracting only necessary edge information. The techniques were developed and prototype equipment was built in the research laboratories of Recognition Equipment Inc. Specific examples of signatures and facial images used in experiments are described.

Signatures are scanned from cards by a solid state self-scanned array photosensor and the images are processed by a two-stage compression algorithm. The digitized signature image is first thinned to a one-cell stroke width "skeleton", then the skeleton is transformed into a vector chain code for storage. The stored reference signature is then available for later retrieval, transmission, and display for visual comparison with a "suspect" signature. The average compression ratio for signatures is observed to be about 25:1. Retrieval and display response time is shown to be on the order of one second.

A variation of this technique was used to accomplish facial image compression. Digitized gray-shade images are first divided into as many binary images as there are bits in the digital gray level. That is, a four-bit gray level image is separated into four binary images, representing the least significant bits, the next least significant, and the next, up to the most significant bits. Then the edge details in each of these images is extracted with the same boundary encoding technique used for signatures. Compression ratios ranging from 4:1 to 7:1 have been observed.

SIGNATURE IMAGE COMPRESSION

Overview of a signature storage and retrieval system

One of the most important and effective ways of validating personal identity for monetary transactions or secure entry is through verification of signatures. Signature verification is the act of comparing a known, valid signature example with one written by the person conducting a transaction or desiring entry. When a large number of potential customers are involved, maintaining a file of example signatures and retrieving the signatures is a very real problem. This is especially true in the banking and retail credit industries where hundreds of thousands of people may hold credit cards or possess accounts at one bank. This report presents a technique which makes feasible centralized storage of large banks of signatures, from which signatures can be transmitted to remote sites quickly and economically. For example, it is feasible to envision a system whereby a signature could be requested from across town, retrieved from a disk file, transmitted via telephone lines, and displayed on a CRT within a period of one second.

Figure 1 illustrates such a Signature Storage and Retrieval System. This system is comprised of three main subsystems: the Composer Subsystem, the Storage Subsystem, and the Display Subsystem. The Composer Subsystem consists of an optical scanner and a digital compression unit; the purpose of this subsystem is to lift an example signature from paper and compress it into compact digital form for Storage. Signatures would appear on paper within a one-inch by four-inch clear area which, if scanned at approximately .008" resolution, can be represented by a digital matrix of 128×512 cells. The compression unit processes the 65,000 bits containing the original signature and encodes all essential information into an average of about 2500 bits. The 2500 bits are then stored on a disk mass-storage device for later retrieval. The Storage Subsystem consists of the disk and a retrieval computer that accepts requests for signatures, determines where the proper signature is stored, addresses the disk to retrieve the data, and transmits the compressed signature to the correct remote display terminal. The Display Subsystem is a number of identical remote terminals each of which has a CRT, keyboard, and signature regenerator unit. The Display Subsystems can be connected with the retrieval computer through dial-up telephone lines or possibly a faster hard-wired communication channel. A signature request can be made simply by entering an account or credit card number

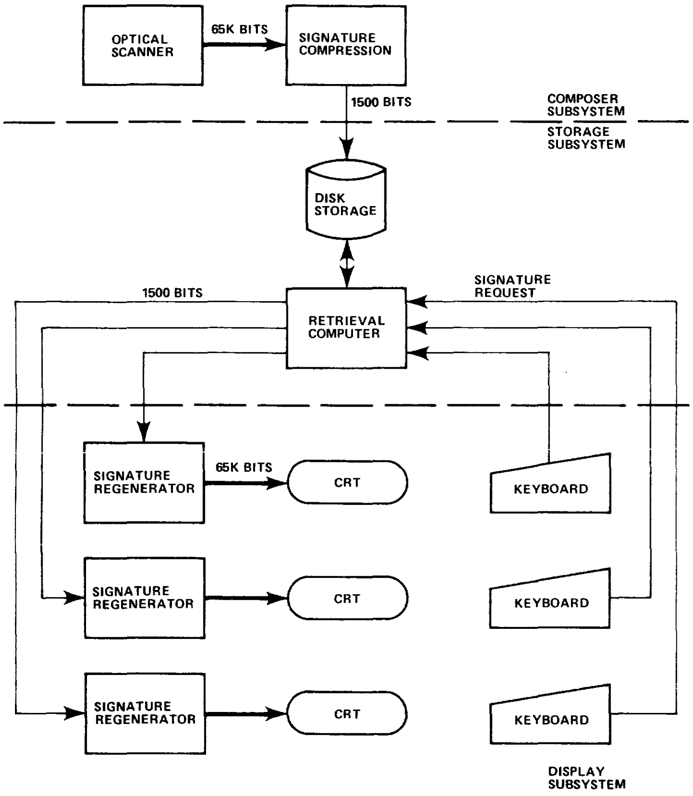


Figure 1—Signature storage and retrieval system

on the numeric-only keyboard. Upon receiving the compressed signature data, the regenerator unit reassembles the signature to its original form in a 65,000 bit refresh memory which drives the CRT display.

Signature compression

Signature compression is accomplished in two stages; thinning, and vector encoding. The signature image is thinned by tracing the boundary and peeling off layers of black cells until the lines are one cell thick. The encoding phase of compression consists of boundary tracing the image one more time and encoding the sequence of boundary points into a string of vectors which represent the signature, and which can be compactly stored in a digital form.

Figure 2 is an image of a portion of a signature after optical scanning and digitizing; the whole image is stored in a 128×512 bit binary matrix in which the thinning and encoding operations are accomplished. A raster scan function is used to find an initial boundary point on the signature, from which boundary tracing will commence. The image boundary is traced by stepping from one black/white boundary point to the next adjacent one. This is illustrated in Figure 2 by the sequence of vectors extending from starting point one at the left of the image. A simple set of algorithmic rules govern the progress of boundary tracing, and certain tests determine when to stop and also how to find separated image pieces as well as the inside areas of the signature. Image thinning occurs simultaneously with boundary tracing; as we progress around the boundary, the outer layer of cells is stripped away until all strokes are only one cell thick. Done correctly, this procedure produces the center "skeleton" of the signature image. During the thinning process, it is important to remove only the desired cells so that lines aren't shortened, breaks are not introduced, and undue distortions do not occur. This is achieved by applying another simple set of algorithmic rules to each cell encountered in the boundary sequence which govern whether or not the cell is to be removed. The thinning process is illustrated in Figure 2 by the "*" cells, which denotes those cells that are removed from the image by the thinning rules. Solid black cells denote the single-cell skeleton that results from boundary tracing and thinning.

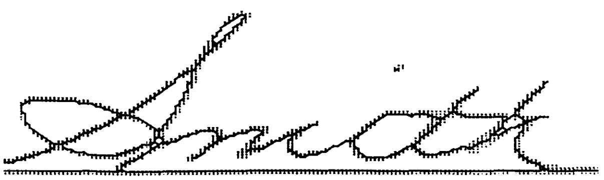


Figure 2

The final step in signature compression is that of encoding the skeleton image. This is accomplished by boundary tracing the image once again to generate a string of vectors defining the links between adjacent cells. There are eight neighbors to each cell of the skeleton, so a three-bit digit gives the location of the next adjacent cell. The entire signature skeleton can be encoded in the form of a number of X, Y starting point locations, followed by a sequence of three-bit vector numbers.

As explained earlier, the encoded signature information can be stored on a digital mass medium for later retrieval. Upon retrieval, the signature can be reconstructed by "redrawing" the skeleton from the starting point and vector information and then performing certain smoothing operations to remove unwanted irregularities or quantization effects.

Examples and results

Figure 3 shows four examples of signatures that were processed; the original signature appears on the left and the compressed signature, after reconstruction, appears on the right. Obviously, because of the nature of this compression technique, the reconstruction is not 100 percent; however, as can be seen, the quality of the reconstructed image is quite good. The table in Figure 4 is a list of 10 signatures that were compressed, showing the compression ratios and the number of bits required to store each signature. In each case, the original image required 65 thousand bits to display the signature as a 128×512 bit binary image.

The amount of compressed storage required is a variable quantity depending on the size and complexity of the signature. For a data set of 50 signatures, the minimum storage was found to be 1571 bits from "Larry Canny," and

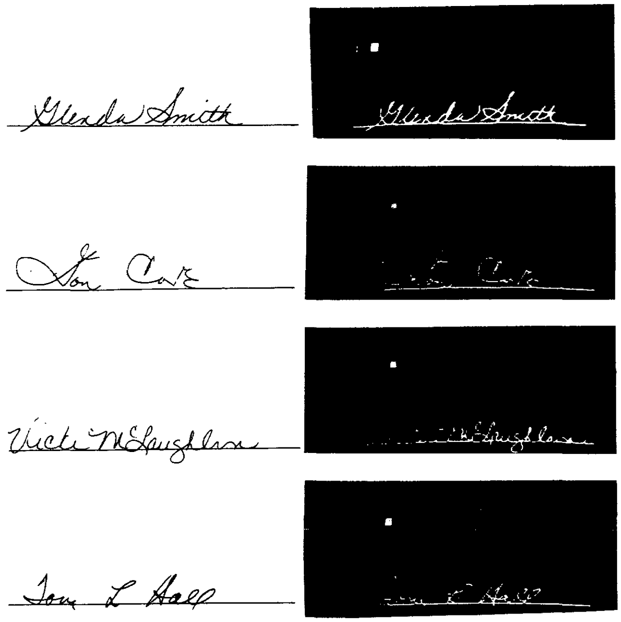


Figure 3—Original and compressed signatures

the maximum was 4407 bits for "Virginia A. Bradford." The average compression ratio for this data was 25:1 corresponding to an average storage requirement of 2666 bits.

FACIAL IMAGE COMPRESSION

The facial compression method

The technique for compressing facial images is a variation of the scheme described earlier. It consists of the following basic steps: (1) decompose the gray-level image into four binary images, each consisting of one bit of the four-bit gray representation, (2) perform a raster scan of each of the four binary images to locate the image boundaries, and upon such location, (3) trace the boundary of each separate image area in order to (4) generate a vector sequence that defines the boundary. These steps are almost identical to the methods used for signature compression, the major difference being that thinning is not done; instead, the first step is the decomposition operation.

Figure 4—Signature compression examples

	Storage requirement (bits)	Compression ratio
1 Glenda Smith	3064	21:1
2 Don Cave	2154	30:1
3 Vicki McLaughlin	3105	21:1
4 Tom L. Hall	2320	28:1
5 Charles R. Carmichael	3666	18:1
6 Jean Helms	2513	26:1
7 Jean Peak	2097	31:1
8 Dorothy Hall	2596	25:1
9 Clifton S. Hayley	3241	20:1
10 Douglas L. Bromfield	2578	24:1

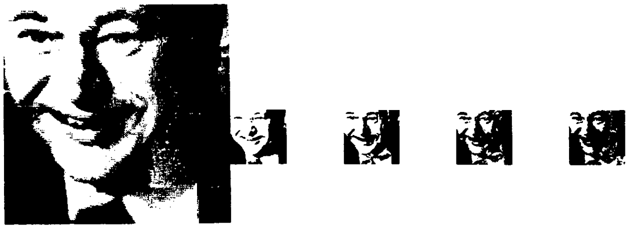


Figure 5—Separation of gray-level image into four binary images

The original image, after scanning and digitizing, consists of a matrix of four-bit numbers, one for each picture element, which defines one of sixteen gray levels for that element. The first step in data compression is simply to form four binary images from the original gray-level image by separating the four bits of each picture element. Figure 5 illustrates this operation; the binary images are labeled from the most significant bit of the gray level (MSB) to the least significant bit (LSB). As can be seen in Figure 5, finer detail is contained in the lesser significant bits. Boundaries that are quite small are ignored; that is, they are just not encoded. This is equivalent to deleting very small black and white areas from the image which are hardly noticeable, but which would contribute significantly to the number of bits required to store the image.

The encoded image consists of x and y coordinates for each boundary starting point, and a series of vectors that describe the boundary. In addition, one bit must be used to designate whether the boundary encloses a "white" or "black" area, since either can occur. This data comprises a fairly compact storage scheme for the image, and when the image is desired, the data can be retrieved and the image reconstructed in a 4-bit digital matrix. The image can be reconstructed from the above information by first retracing the boundary paths and setting the boundary cells for each binary image in the appropriate matrix bit plane. Then the white and black areas within the boundaries can be filled in with one pass to a raster scan. In this way, the correct gray level for each picture element of the image is reconstructed.

Results and conclusions

Figure 6 shows an example of a facial image after digitizing and after compression and reconstruction. The

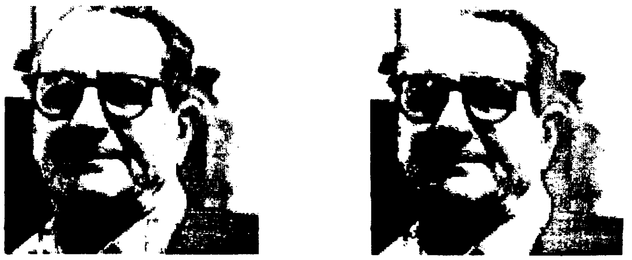


Figure 6—Original and compressed images

Figure 7—Facial image compression examples

Image #	# Bits boundary encode	Comp. ratio boundary encode	# Bits run length	Compression ratio run length
1	15,264	4.30		
2	9,122	7.18		
3	14,836	4.41	24,900	2.62
4	12,008	5.46	19,117	3.43
5	16,030	4.10	26,570	2.46
6	15,238	4.30	22,013	2.97
	———	——	———	——
Total	82,498		92,600	
Avg.	13,750	4.78	23,150	2.84

image was digitized in 16 shades of gray from a two inch photograph; the scanner resolution was .016″ which yields an image size of 128×128 pixels. Picture quality at this resolution is only fair, but it is adequate to accomplish facial recognition. Figure 7 shows the compression ratios and the number of compressed bits for six facial images and a comparison with run length coding for four of them. The average compression ratio for the boundary coding technique is 4.78 compared with 2.84 for the more standard run length coding.

Evaluation of images produced at three different resolutions (.008″, .016″, .032″) afforded an opportunity to see the relation between image resolution and compression ratios. The number of bits required to store the three different size images after compression with the boundary coding technique was approximately 8000, 15,000, and 40,000 respectively for the 64, 128, and 256 cell images. These numbers suggest a linear relationship between the number of bits required to represent the compressed image and the number of cells on a side. This postulate is logical because the boundary coding technique codes only the boundary vectors, and the boundary circumference (number of vectors) increases linearly with the picture size.

A second observation was that the background scene of the photographs contributed to the compressed data so that the compression ratio could be further improved by insuring that facial photographs are taken against a plain monochrome background.

Although the boundary coding yields a better compression ratio for facial images than run-length coding, it would amount to a more expensive hardware implementation. The boundary coding processor requires a large random access buffer memory equal in size to the number of cells in the image.

In conclusion, it has been shown that it is feasible to compress a 16 gray-level facial image of size 128×128 points into approximately 16,000 bits (compression ratio greater than 4:1) with good quality reconstruction.

An interactive text-editing system in support of Russian translation by machine

by DAVID A. LUTHER
USAF/RADC
Rome, New York

and

CRISTINE MONTGOMERY and RONALD M. CASE
Operating Systems, Inc.
Woodland Hills, California

ABSTRACT

An interactive, text-editing system was designed and built to support the pre- and post-processing of machine translated scientific and technical Russian literature.

Sixteen independent CRT/keyboard terminals are supported on five small processors with a distributed data base and distributed processing.

A "free-form" text editor has been provided for the creation or modification of textual files. Unusually powerful functions for technical editors have been implemented such as "restore text" and "alternate word list." Software character generation is used to display any of a mix of 256 symbols from four logical alphabets.

INTRODUCTION

The USAF Foreign Technology Division (FTD) and Rome Air Development Center (RADC) have collaborated for ~~some time in the development and application of computer~~ or machine translation systems. They have been specifically applied to the conversion of foreign scientific and technical literature into English. The quality of the current translation system is under constant improvement and has reached a satisfactory level. However, an analysis of the current operational environment accented the need for total system improvement, with focus on automating the supporting functions of input and output, which contribute most heavily to total machine translation costs.

In this paper we describe an interactive, text-editing system designed and built to support the keyboarding of cyrillic text, and the editing and composition of machine translated English. Sixteen display/keyboard terminals have access to sophisticated text manipulation functions and can display any of a repertoire of 256 symbols. A high level of interaction is maintained by a unique system of five small distributed processors.

FUNCTIONAL DESCRIPTION

The machine-translation process divides into three functional areas: input or keyboarding of foreign material, translation by computer software, and editing and recomposition. The results of a technical analysis served to document the feasibility of a machine-aided editing system oriented to the use of CRT display devices used to interact directly with the machine-translated output.[1] A later study reported on user reaction to an experimental text-editing terminal installed within the FTD operational environment.[2] In conjunction with the ultimate users, requirements were identified and specifications developed.

The resulting edit system was designed to provide the means for 16 independent users at different editing stations to create, view and edit different documents at CRT/keyboard terminals. Transfer to and from the translation system computer, an IBM 360/65, is via magnetic tape. The software required to support all editing and text manipulation is organized on two levels. At the bottom level, the majority of the editing functions are supported on four identical IMLAC PDS-4 processors; each processor supports four terminals. Above them is a PDP 11/05 which supports complex editing functions, file handling and peripherals.

FUNCTIONAL CAPABILITIES

The edit system is logically organized such that one of the terminals is designated as the system operator. While so designated, the user at this terminal can have access to system level commands such as assigning individuals to terminals, for instance. This function, like all others, is completely terminal oriented; that is, all commands and edits, all actions and interactions take place through the terminal.

End-user requests and commands at a terminal fall into three modes: command string, edit and review modes. Command string mode deals with start-up and utility functions. Its syntax has a traditional form, i.e., a Command Word and a parameter(s).

The Edit mode is the most interesting and powerful of the three. It is provided for the creation and modification of textual files. The CRT display is a 1000 character "window" on a scroll of text. The text is displayed on 80 character lines in "free-form," characteristic of technical documents. All changes are made directly and immediately on the text displayed in the window using the keyboard for cursor positioning (pointing at the text), function selection and literal input.

In Review mode a specified document will be displayed at the requesting terminal but no editing will be permitted. More than one terminal may be reviewing the same document, and each terminal may be reviewing a different portion of the document.

EDIT MODE

As originally envisioned, the edit system was to be used primarily to edit and correct text material produced by the machine translation process. Its use here by the human operator or translator would be necessarily restricted to scanning and correcting functions. However, the projected use of the system evolved and grew to include human translation and cyrillic input; both applications stressing text input or creation. The combined functional requirement grew thereby to cover a gamut of editorial and composition functions.

When in Edit Mode, the user will normally find his terminal in an "insert" status. That is, any literal key that is struck will put a character on the display at the position indicated by the cursor. The cursor is positioned by way of cursor control keys on the keyboard. If the cursor is in the midst of existing text, the file will be opened up and the characters to right of the cursor will be pushed to the right. Full word wraparound (preservation of word integrity) will occur at the right margin; the word pushed off will be inserted at the beginning of the next line, and so on.

Characters and words can be deleted by pointing at the character or any part of the word, respectively, followed by pushing the appropriate function key. Another character deletion function, called rubout, will remove the character to the left of the cursor. This is extremely useful for the occasional slip of the finger while creating text. Any material deleted by any method is replaced by a special null character, displayed as a "bullet." Upon execution of the close-up function, the nulls will be removed and the text file closed-up from the bottom. This approach is much more satisfying than a text file which is changed for every occurrence of a delete.

A special capability to restore a line of text following the execution of an editing function is available. The line will be restored to what it was just prior to the last edit. This capability is intended for recovery from an inadvertent edit or for that user who suddenly changes his mind.

SPECIAL FUNCTIONS

Two powerful functions have been included that are particularly useful for editing machine translations. They are the Select and Alternate Word List functions.

By positioning the cursor under one word of a series of embedded alternate words supplied by the machine translation algorithm, and striking the select key, the user can select one word to remain and cause all the others to be deleted. This capability puts the human translator back into the loop to make difficult choices which may be beyond the capability of an automatic translation algorithm to perform.

The alternate word list allows each individual to set up a special dictionary consisting of word pairs, one of which is the string to be replaced and the other is the replacing string. During an editing session, a user need only point to an occurrence of a string to be replaced, strike the alternate word function key and a swap will be made.

HARDWARE/SOFTWARE IMPLEMENTATION

The programs which support all of the highly interactive tasks are running on four identical IMLAC PDS-4 processors. On any one IMLAC the program is shared by four keyboard/display units. Each unit has one-fourth of the display refresh capacity as well; display refresh in the IMLAC is handled by a separate processor. One significant reason the IMLAC was selected is the fact that characters are generated in software. This made possible the definition of 2^8 or 256 separate symbols. They are divided into four groups: roman, cyrillic, Greek and math/technical. A mode key on the keyboard switches the way in which a key strike is interpreted. Combinations of the four alphabets may be mixed on the display.

Each IMLAC is connected by a DMA interface to the PDP 11/05. The PDP is called upon to load and unload files, for execution of complex functions such as move and copy, and for mass memory storage. A scroll at the IMLAC beyond its buffer limit will cause a request to be sent to the PDP for data.

This configuration is a true example of division of labor. Considering the tasks to be supported, the processors are very small and are being used to their maximum. However, under no circumstances to date has there been anything less than immediate response, that is, *no* delay apparent to the user.

REFERENCES

1. Conti, E. and N. Demuth, "Study of Machine-Aided Editing," RADC-TR-67-390, February 1968.
2. Evans, E. A., "Machine-Aided Post Editing," RADC-TR-72-227, September 1972.

Computer generation of conference presentations

by CHARLES A. BELOV
Aetna Life and Casualty
Hartford, Connecticut

ABSTRACT

There are various subjective factors which interfere with the usefulness of the typical conference presentation to the individual listener. This satire does not attempt to name and categorize these factors; this would in itself interfere with understanding. Instead, a hypothetical processing system for creating conference presentations is described, supposedly by the system itself. The system has a number of interfering factors as well as positive factors built into it. Areas of study are the use of acronyms; methods of developing the introduction, main topic, and conclusion; graphs and tables; and small details which tend to distract. Contributing to the satirical purpose, the system is an example of computer overkill—an entire system is developed to create one conference presentation. A gentle plea is made to future writers to view the listener as the most important consideration when editing their product.

INTRODUCTION

Computers and speed have been allies since the turn of the century. Ever since Herman Hollerith devised a tabulation system for the United States census of 1890, it has been recognized that, while computers may not always be the cheapest way of doing things, they are certainly the fastest. Through the years, the speed of the computer has increased to the point where millions of calculations can be performed every second. Even the common hand-held calculator can give answers to problems essentially the instant that the equals key is pressed. This great speed advantage, combined with a potential savings in labor costs, has permitted the computer to be utilized as a jack-of-all-trades, in manufacturing, finance, education, communications, and just about any other application imaginable. Therefore, it should come as no surprise that this author should turn to the computer not merely as a subject, but indeed as a source for a live conference presentation.

A DEADLINE TO BE MET

On November first, 1976, this author was informed that the American Federation of Information Processing Societies was seeking papers for presentation at its 1977 National Computer Conference. At the same time, we found that Aetna Life and Casualty, the company at which this author is employed, was encouraging its data processing employees to submit papers. We were interested in presenting a paper in the primary area, "The Individual and Computing," but were awed by the deadline date of December first.

We have become acutely aware that we as individual programmers must view the computer from both ends, as one who both causes actions and feels the effects of one's programming. Our programming experience thus could be used to devise a system which would generate a presentation. Our consumer selves would then be able to take advantage of a system which would supply us with a paper prior to the December first deadline.

This seemed to be the only reasonable step. The alternative would be to rely on creative juices to come up with a presentable paper in the short span of one month's time. The choice was clear. Time was of the essence, and that very factor pointed to the computer as the only feasible means to meet the deadline which had been set.

BASIS FOR A SYSTEM

Before designing the system, it was necessary to determine the characteristics of a typical conference presentation. Reference for this study was the AFIPS Conference Proceedings, 1975 National Computer Conference. The typical paper had certain readily identifiable characteristics. We shall study the abilities of the system with regard to each of these characteristics as we identify each one.

ACRONAMING THE SYSTEM

Characteristic one: When a system is involved in the presentation, it has an acronym for a name

When a new system is presented, it is customary that the system be named with an acronym. As most of us know, an acronym is a word made up of the first letter or letters of other words which actually describe the system or other entity which is assigned the acronym. Our presentation generator had to be capable of making key words entered by the author into acronyms. The author would have to specify the amount of contrivedness used in creating the acronym. With **L**ow **A**cronym **C**ontrivedness **K**eying (LACK), the acronym is sought to fit the description. A good example of LACK specification is COBOL, or **CO**mmon **B**usiness **O**riented **L**anguage. This author, however, chose **H**igh **A**cronym **C**ontrivedness **K**eying (HACK) in which a description is sought to fit the acronym. Entering the key words Presentation, NCC, and Computerized, we were supplied with the acronym PROCEEDINGS, for **P**resentation **R**igmarole **O**ptimized **C**omputerized **E**laboration **E**ditor (**D**eveloped and **I**ntended for the **N**CC) **G**enerating **S**ystem. This acronym is displayed in Table I. Thus, our system for presentation generation was given the name PROCEEDINGS.

BEGINNING WITH AN INTRODUCTION

Characteristic two: The presentation begins with an introduction

An introduction usually serves two purposes. One is to summarize what has gone on in the past. The other is to lead into the topic to be presented.

The presentation author can specify as to how much past information he wishes to present. He can limit it to a sentence or two, this being known as giving history the short shrift. Conversely, he can spend the first half of his talk in this area, and this is called dwelling in the past. PROCEEDINGS permits the author to choose either of these extremes or anywhere in between.

Leading from the past into the topic to be presented is a rather simple process for PROCEEDINGS. In the typical human-composed introduction, sentences are ordered logically to lead into the main topic. Since computers and logic go together like bread and peanut butter, this portion of the presentation is a piece of cake for the PROCEEDINGS system.

However, the introduction is sometimes used to summarize the entire presentation. This is not recommended in the PROCEEDINGS system, because this involves the use of a pre-post-processor. This method of processing the data before it is available has not yet been proven accurate in our generator.

TABLE I—Words Represented by the PROCEEDINGS Acronym

Letter in the PROCEEDINGS acronym	Word(s) represented by the initial letter
P	Presentation
R	Rigmarole
O	Optimized
C	Computerized
E	Elaboration
E	Editor
D	(Developed and
I	Intended for the
N	NCC)
G	Generating
S	System

PRESENTING THE PERTINENT FACTS

Characteristic three: The presentation involves the presenting of pertinent facts

A number of options are available in the PROCEEDINGS system by which the author can present his main body of material. He or she can specify to PROCEEDINGS that the length of sentences, that is, the basic group of words which are strung together, at least in our English language, word after word until a basic thought or thoughts is or are completed and possibly reiterated until the listener is totally unsure of the overall meaning despite understanding perfectly the meaning of individual phrases within such a sentence, shall be long. PROCEEDINGS sentences can also be short. Sentences will be understandable if a low fog index is requested. Conversely, and in fashion detrimental to the cognizance of the conferees, it can be specified that the fog index shall be high. Buzz words, idioms, and both formal and informal language can be mixed in proportions to suit the author.

The facts themselves are another matter. The author may, if so desired, enter the facts concerning the subject to be reported upon. The computer will rearrange and augment these facts so that they comprise a presentation. However, recall that at the start of this talk it was said that the major advantage of computers is their great speed. It is much faster to let the computer write its own presentation using the facts it already has. The author would simply make his specifications of sentence length, buzz word content, and other criteria which would alter the computer output to his style of writing. Such an approach was used for the writing of this presentation. The facts which the PROCEEDINGS system had, and could therefore use, were facts concerning the PROCEEDINGS system itself. The result, logically, was this presentation on the PROCEEDINGS system. Admittedly, such a technique limits the scope of future presentations. On the other hand, we must expect to make such minor sacrifices for the sake of speed and accuracy.

To permit some variety in the presentation, yet observe a logical progression of ideas, a new method of file organization had to be developed. Using random access storage devices, we created the Random Sequential method of file organization. We won't go into the technical aspects of Random Sequential access, but it is rather like dropping the tone arm of your phonograph onto a long-playing record and letting the record continue to play. Diversity, yet continuity.

HEADING THE SUBTOPICS

Characteristic four: The publication copy of the presentation has headings for each subtopic covered

When a presentation is published it is customary to provide subject headings throughout the work. This is to enable a reader to skip sections which the reader thinks will not be interesting, and to aid in locating items of special interest. The PROCEEDINGS system can produce short headings, long headings, descriptive headings, humorous headings, and more. In this paper, for instance, the headings of all sections relating to the abilities of the PROCEEDINGS system have the letters "ing" completing the first word of the heading. This can be seen better by studying the printed version of this report.

LISTING WITH TABLES

Characteristic five: The presentation utilizes one or more tables for listing data

Few presentations are complete without a table of facts which would be boring or confusing if read aloud during the presentation. PROCEEDINGS can compile lots of facts for any length desired, complete with footnotes. This can be seen in Table II. The tables can even be informative, as can be seen by referring once again to Table I.

SHOWING WITH DIAGRAMS

Characteristic six: The presentation has one or more impressive looking graphs and/or drawings

Just as tables are an essential part of the well-dressed presentation, so are diagrams, graphs, and pictures. PROCEEDINGS diagrams can be simple or complex, well documented with explanatory captions or cryptic and ambiguous. They are all impressive, however, which is the reason for using a diagram in the first place. In this particular presentation, the author specified that he wanted two graphs. This was an unwise choice at the time, simply because we don't have all the bugs worked out of the graphing subsystem; the graph output is egotistical and certain of its creative abilities. This can be seen by referring to Figures 1 and 2.

TABLE II—Abilities of PROCEEDINGS to Produce Tables of Desired Lengths

Desired length of table		Within ability of PROCEEDINGS
1	entry	Yes@
10	entries	Yes
100	"	Yes
1,000	"	Yes
10,000	"	Yes
439,376,421,121	"	Yes*

@ Rather silly, but easy to produce.

* Not recommended, as printing takes a while.

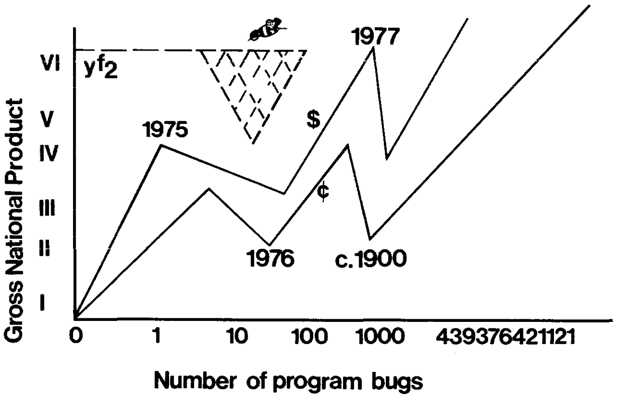


Figure 1—A meaningless yet impressive graph

SUMMING UP*

Characteristic seven: The presentation has a summary or conclusion

Almost as important as the introduction is the summary or conclusion. While the introduction sets the stage, so to speak, the summary reviews that which was too complicated to understand the first time around. This is also true of a conclusion, but a conclusion usually also encourages a course of action or serves to say "I told you so." The PROCEEDINGS system can produce on command a con-

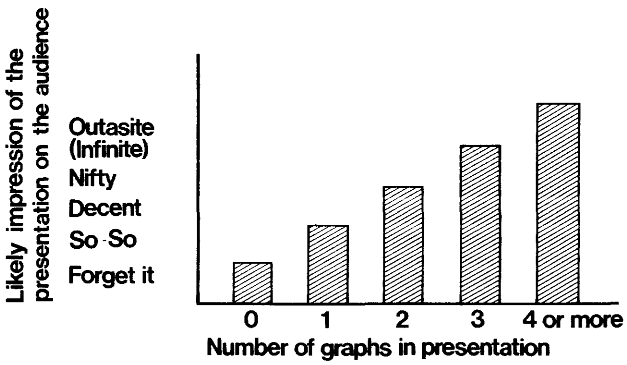


Figure 2—Impressiveness of presentations

* Note: The presentation is not over. We are merely describing the conclusion.

clusion or summary that is a brilliant review of the points which were covered in the equally brilliant presentation. Conversely, it can cut a report short with a "Let's get the heck out of here" attitude. The choice of degree is up to the individual author, just as it is with almost every other aspect of a PROCEEDINGS-produced conference presentation.

PROVIDING REFERENCES

Characteristic eight: The presentation has a list of references

It is common, following the conclusion of an article, to provide a list of references used in the production of a presentation. If the entire report was fabricated by the PROCEEDINGS system, as this one was, this could conceivably pose a problem. But, PROCEEDINGS makes it surprisingly easy. Depending upon user specifications, PROCEEDINGS can provide legitimate references which were actually used to supply PROCEEDINGS with data; it can provide real references which have nothing to do with the presentation; and it can provide fictitious references, which are useful because they are difficult to check on. For illustrative purposes, we have included all three types in the list of references for this presentation.

EXHIBITING IDIOSYNCRACIES

Characteristic nine: Except during acknowledgments, use of personal pronouns in the first person singular is avoided

The PROCEEDINGS presentation can be extremely personalized. While most prefer to do their own "uh" 's, "ah" 's, hand wavings, nervous tics, and "ahem" 's, these are not beyond the capabilities of PROCEEDINGS. However, speech conventions such as avoidance of personal pronouns in the first person singular can easily be worked in by our system. This very presentation utilized "this author," "we," and circumlocutions such as "it was necessary to determine." PROCEEDINGS can be invaluable by providing variety in this awkward situation.

CONCLUSION**

There are many "ingredients" which go into a talk on computers, or, indeed, a talk on any subject. All of these so-called ingredients have a potential of being useful to the listener. The PROCEEDINGS system can provide all of the ingredients, but it is still up to the individual author to set the generator specifications so that an understandable report is presented. This system was designed to run only once, to produce a presentation for the 1977 National Computer Conference. However, I hope that I have programmed it well enough that the principles behind it are clear, and that these principles will be used in the future by more people than use them today. A poorly programmed machine can make a talk such as this totally useless, regardless of whatever vital facts the talk may contain.

If we may leave this fantasy which I have presented to you and return to reality, I could not have had the time or resources to plan and implement an actual PROCEEDINGS system in one month. Therefore, I was totally unprepared to write on it. The only thing I could do was to resort to the use of a computer to prepare my entire presentation instead of writing it myself as I probably should have done. My apologies to you all.

REFERENCES

1. *AFIPS Conference Proceedings, 1975 National Computer Conference*, AFIPS Press, Montvale, New Jersey.
2. Belov, Charles A., "Computer Generation of Conference Presentations," *AFIPS Conference Proceedings, 1977 National Computer Conference*, AFIPS Press, Montvale, New Jersey.
3. Goldstine, Herman H., "Billings, Hollerith, and the Census," *The Computer from Pascal to von Neumann*, 1972, Princeton University Press, Princeton, N. J.
4. Raskin, Jef, "Errata," *Journal of Irreproducible Results*, 20, 2, December 1973, pg. 30.
5. Wrentchler, Polly and Nicholas Teck, "A General Reference for Conference Presentations," *Journal of Reference Citations*, May 1975, pp. 376-427.

ERRATUM

(Added by the author—not computer generated)

As mentioned before by the PROCEEDINGS, there are still a few bugs in the system. In the original computer-produced manuscript, page five follows page three. It is not known whether there was any loss in text because our control counters all contained a value of pi. Apparently, the computer was hungry and had stepped out for a byte to eat.

** Note: The real ending, this time.

Design of a diagnosable and fault-tolerant input/output controller

by A. K. BOSE and S. A. SZYGENDA
The University of Texas-Austin
Austin, Texas

ABSTRACT

The paper describes the design of a diagnosable and fault tolerant input/output controller. The approach used is to follow an initial design effort by a detailed analysis of the organization of the system and its environment. Based on this analysis, modifications are made on the system and its environment to get the desired performance.

For the interface controller under consideration, it is seen that its diagnosability and fault-tolerance can be improved significantly by introducing a minor amount of redundancy into the system. The environment which in this case is predominantly a CPU or a channel is modified only to the extent that a nominal amount of software is added and the existing software modified.

The overall effect of these modifications results in a system which achieves the reliability comparable to that of duplication but for a redundancy of approximately 30 percent.

INTRODUCTION

Massive redundancy techniques, applicable in general to most systems or subsystems, can be applied to input/output controllers. Reliability can always be purchased by paying for it in terms of redundancy. But, in most instances, the application of a general technique does not lead to the most cost-effective solution.

The answer to the problem of efficient acquisition of reliability invariably lies in the internal organization of the system and its environment. A detailed analysis of the system and its environment, following an initial design effort, may provide an economic way for making the system reliable.

This paper discusses the design of a diagnosable and fault-tolerant input/output controller. A preliminary design is followed by a detailed failure analysis based on which a final system is synthesized. Using only a nominal amount of additional hardware and a minor increase in software, the final system achieves a level of fault tolerance which is comparable to that of a massively redundant system.

PRELIMINARY DESIGN

In the first phase of the design effort, an asynchronous communications interface controller is designed. These controllers are commonly used for interfacing the CPU or the channel with peripheral devices requiring a serial data format such as CRT terminals and teletype printer/readers. The controller is very similar to the commercially available Universal Asynchronous Receiver Transmitter (UART) and thus only a brief description of its operation is provided.

A block diagram of the controller is shown in Figure 1.[1,2] The system is capable of full duplex operation and can handle multiple baud rates (receiving-transmitting) simultaneously. It can receive 8 bits of data in parallel from the CPU and communicate it in serial to the peripheral, or it can receive 8 bits of data in serial from the peripheral and format it for parallel read for the CPU. The transmitter part adds start, parity and stop bits to the serial output data while the receiver part checks the serial input data for the same. Error registers in the receiver indicate parity error, overrun error (when a new character wipes out the previous character before it has been read) and framing error (when valid stop bits are not received). Several control and flag lines which are used by the CPU to control the operation of the controller are also available and are listed below.

The timing diagrams for the transmitter and receiver for the data, control and status lines are shown in Figures 2 and 3.

ANALYSIS OF THE INITIAL DESIGN

A study of Figure 1 shows that a certain amount of symmetry exists between the input and the output paths for the data flow between the CPU and the peripheral. The format changes that are required for data flow in one direction are exactly opposite to the format change required for data flow in the other direction. If the peripheral were to be treated as a reflector, that is, whatever appears at its input is reflected back to the output, a closed loop is established for the data flow. The data that is put out by the CPU should be received back by the CPU after having

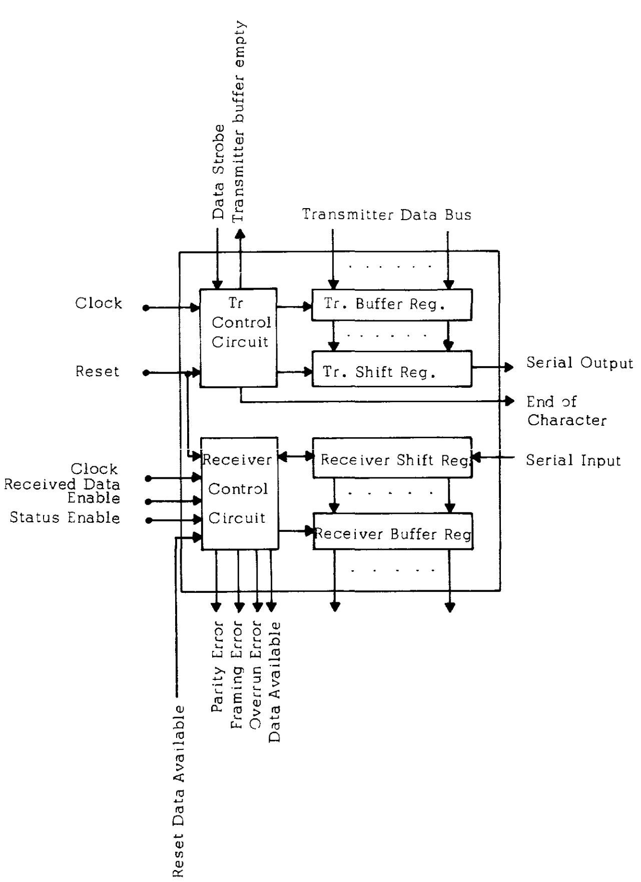


Figure 1—Block diagram of an asynchronous communications interface controller

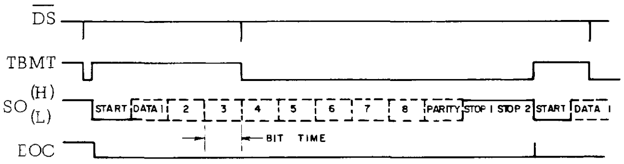


Figure 2—Transmitter timing

propagated in both directions through the interface controller. Thus, if enough hardware is added such that the peripheral may be made to appear as a reflector when desired, it should be possible for the CPU to check the data paths of the controller by matching the incident data put out by it with the reflected data received after having propagated in both directions through the controller.

The method discussed could be used for both off-line and on-line testing of interface controllers. On-line testing, which is the primary objective, would require an indication to the CPU of a suspected malfunctioning of the controller. Periodic testing would disrupt normal operation and be very inefficient in terms of CPU time. Only on suspicion should the CPU stop its normal operation and execute the controller test routine.

To provide the CPU with an indication of the malfunctioning of the controller, a certain amount of concurrent checking of the data propagating through the controller is required. Sufficient checking capability could be built into the controller at a nominal additional cost, whereby an interrupt signal to interrupt the CPU could be generated by the checkers, if it is detected that bad data is being put out

TABLE I

Symbol	Name	Function
$\overline{DS}$	Data Strobe	A strobe on this line will enter the data bits into the transmitter buffer register. The line is low active.
TBMT	Transmitter Buffer Empty	This flag goes to a logic one when the transmitter buffer is empty and a new character can be loaded.
SO	Serial Output	This line will provide, serially by bit, the entire transmitted character. It will remain at logic "1" when no data is being transmitted.
EOC	End of Character	This line goes to a logic "1" each time a full character is transmitted. It remains at this level until the start of transmission of the next character.
TCP	Transmitter Clock	This line requires a clock whose frequency is 16 times the desired transmitter baud rate.
$\overline{RDE}$	Received Data Enable	This line is the tri-state controller. A logic "0" places the received data onto the receiver shift register *output lines* while a logic "1" leaves the output lines in a high impedance state.
PE	Parity Error	This line goes to a logic "1" if a parity error is detected in the received data. Tri-state output.
FE	Framing Error	This lines goes to a logic "1" if two valid stop bits (logic "1") are not received. Tri-state output.
OR	Overrun Error	This line goes to a logic "1" if the previously received character is not read (DAV line not reset) before the present character is transferred to the Receiver Buffer Register. Tri-state output.
DAV	Data Available	This line goes to a logic "1" when an entire character has been received and transferred to the receiver buffer register. Tri-state output.
$\overline{RDAV}$	Reset Data Available	A logic "0" on this line will reset the DAV flip-flop and return the DAV line to a logic "0".
SI	Serial Input	This line accepts the serial bit stream of received data. A "1" to "0" transition is required for initializing data reception.
RCP	Receiver Clock Pulse	The receiver clock whose frequency is 16 times the desired baud rate has to be provided on this line.
$\overline{SE}$	Status Enable	Tri-state controller for the lines DAV, OR, PE and FE.

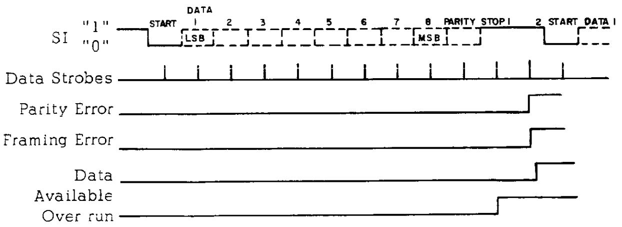


Figure 3—Receiver timing

by the controller. This interrupt signal could start the CPU's diagnosis of the controller.

Since the CPU check could be used to achieve some degree of fault isolation, it should be possible, that in certain situations, a degraded but tolerable functioning of the controller could be permitted. Most controllers have some error detecting capability. If the CPU check isolates the fault to this part of the controller and at the same time verifies that no faults exist in the data paths, the CPU could then continue to use the controller while ignoring error messages put out by it.

In the situation where it becomes impossible to use the controller anymore because of the presence of fatal faults, it should be possible for the CPU to perform spare switching, whereby it would switch out the faulty controller from the system and switch in a good one. The CPU could check the new controller and establish that it is fault free. Normal operation could then be resumed. Since several similar controllers are normally used in a system, one spare could be used to cover all of them.

The techniques discussed in this section have been arrived at from considerations of the internal organization of interface controllers, their function and their environment. Interface controllers always operate under the control of a CPU or a channel which have the capability of performing the operations mentioned above. As long as the involvement of the CPU or the channel is kept low, the cost incurred is very nominal.

ADDITIONAL HARDWARE FOR IMPROVING TESTABILITY

Based on the discussion of the preceding section, additional hardware in the form of three switches is added to the controller to improve its testability (Figure 4). The three switches, operated under CPU control, would provide a connection between the serial output line of the transmitter and the serial input line of the receiver and would enable the use of a common clock for the transmitter and receiver.

FAILURE ANALYSIS

The entire controller circuit was simulated to determine the tests required to detect all single "stuck at one" and "stuck at zero" faults. The simulated circuit has 111 elements. Considering every individual input and output line of each element as a possible site for a stuck-at-one (S-A-1) or stuck-at-zero (S-A-0) fault led to a possibility of 894 possible stuck-at faults in the circuit. The simulator reduced this to 721 faults after considering that several faults could have identical effects (ex., a S-A-1 fault at the input of an inverter is identical to a S-A-0 fault at its output).

As shown in Figure 4, the 8 "Data Input Lines," the transmitter "Data Strobe" line and the receiver "Reset Data Available" line were defined as the inputs while the 8 "Data Output Lines," the 4 "Status Lines" and the "Transmitter Buffer Empty" lines were defined as the detection points. This representation is consistent with the normal CPU-controller interconnection.

With the intention of detecting S-A-0 and S-A-1 faults in the data paths, two obvious input patterns, "11111111" and "00000000." were tried. Signals required for proper operation were provided on the Data Strobe line while one of the "Reset Data Available" signals was deliberately skipped to test the overrun error detection hardware. The input data pattern was also changed back to "11111111" after the second strobe to detect faults in the strobe inputs. The detection points were strobed after the Data Available line in the receiver went high signifying the end of data propagation in the controller.

This test resulted in the detection of 485 faults of 67.2 percent of the total faults. An analysis of the remaining 236 faults showed that 96 of them were "don't care" faults. These did not affect proper functioning of the controller and thus could not be detected. Most of them were S-A-1 faults on inputs that were connected to the power (logical 1) lines. Since the effect of both are the same, these faults could neither be detected, nor could they introduce any error in the operation of the circuit.

Of the remaining 140 faults, 89 could not be detected by the simulator but would be detected in the physical system. These faults were mainly the stuck-at faults on the reset, strobe and initializing lines in the controller. If an element is not properly initialized, the simulator treats its output as

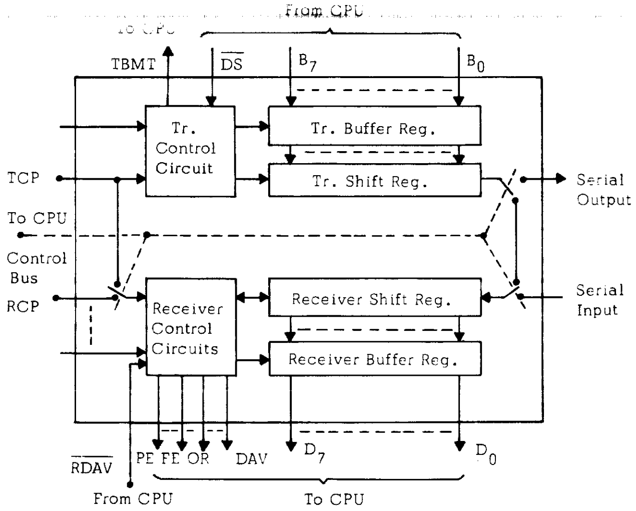


Figure 4—Hardware to improve testability

indeterminate "X." This makes the outputs of all the elements through which this signal propagates also an "X." The simulator does not consider "X" for fault detection since their value could be either "1" or "0." In the physical system, however, an improperly initialized element would have an output of either a "0" or "1." Normally following power up, the distribution of 1's and 0's is random and would be detected by a pattern of all 1's or all 0's. Even in the situation where, following power up, the outputs of the faulty elements are such that they are not detected by one pattern, they would definitely be detected by the other.

The two test patterns "11111111" and "00000000" thus cover 670 of the 721 possible faults. Of the remaining faults, 4 are detected through the use of the input pattern "11111000." This is required to detect S-A-0 faults in the parity generating and checking hardware since the first two patterns are such that the parity bit is zero in both cases.

Detection of the remaining 47 faults require complicated test sequences rather than different input patterns. Seven faults are detected by providing an idle period for the controller when no data is being propagated. These faults are such that they make the controller operate even during this idling period. Of the remaining 40 faults 24 more are detected by using a different strobe period for looking at the detection points. These faults result in improper timing in the controller thus making them operate faster or slower than normal.

The remaining 16 faults cannot be detected in the existing setup. They are located in the stop bit generation circuit of the transmitter or the framing error detection circuit of the receiver, and their existence is suspected by the repeated detection of framing errors by the receiver or the peripheral. The reason why these faults cannot be detected is that the CPU cannot affect their operation in any way. The detection of S-A-0 faults in the framing error detection hardware of the receiver requires a deliberately introduced error in the stop bits put out by the transmitter. Since this is not possible for the CPU to realize, these faults cannot be detected. If, however, the serial input and output lines of the controller are placed directly under CPU control, detection of these errors would be possible. This is unnecessary because these faults are not fatal faults and a degraded operation of the controller (as discussed in a following section) is possible in their presence.

Since a large number of the possible faults (93 percent) are detected by just two input patterns while the remaining few faults require several complicated tests, greater testing efficiency is achieved by properly sequencing the tests. A flow diagram which shows an optimum testing scheme is shown in Figure 5.

HARDWARE FOR CONCURRENT CHECKING

Concurrent checking hardware is introduced with the primary objective of detecting the propagations of erroneous data due to faults in the controller. Situations where erroneous data arises out of faults external to the controller

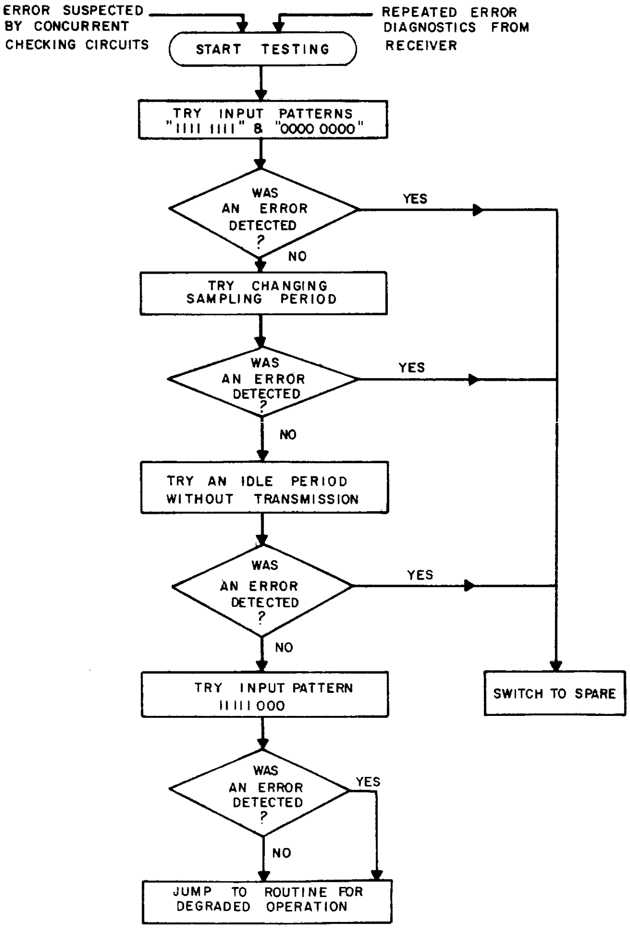

ERROR SUSPECTED BY CONCURRENT CHECKING CIRCUITS
REPEATED ERROR DIAGNOSTICS FROM RECEIVER
START TESTING
TRY INPUT PATTERNS "1111 1111" & "0000 0000"
WAS AN ERROR DETECTED ?
YES
NO
TRY CHANGING SAMPLING PERIOD
WAS AN ERROR DETECTED ?
YES
NO
TRY AN IDLE PERIOD WITHOUT TRANSMISSION
WAS AN ERROR DETECTED ?
YES
NO
TRY INPUT PATTERN 11111 000
SWITCH TO SPARE
WAS AN ERROR DETECTED ?
YES
NO
JUMP TO ROUTINE FOR DEGRADED OPERATION

Figure 5—Flow diagram of optimum testing scheme

will not be considered. These are normally checked by the existing error detecting hardware inside the controller which puts out error messages for the CPU. No attempt will be made to correct detected errors or to detect a fault that does not produce erroneous data.

A general technique to realize concurrent checking is duplication, which can be done at the system level or at any lower level. The two outputs are matched and any error in one unit produces a mismatch resulting in detection. As mentioned earlier, this approach is expensive and a cheaper solution is found by closely examining the internal hardware.

For convenience, the internal organization of the controller can be considered as three separate functional units: the data circuit, which consists of the data paths and the data storage facilities; the control circuit, which provides the timing signals; and the auxiliary circuit which includes the flag and error flip-flops. Each of these circuits will be considered separately to determine their role in the propagation of bad data.

Flow of data in both the transmitter and the receiver involves a parallel path and a serial path. Faults in the parallel paths of the transmitter can be checked by introducing a parity bit at the input and verifying it at the output.

Since the normal operation of the transmitter requires a parity encoder, the additional hardware involved would only be that of the checker which has to be located at the output. Similarly, the receiver has a parity checker at the output. To check for faults in the parallel path of data flow, an additional parity checker can be introduced at the input and the outputs of the two checkers compared. A mismatch would indicate a fault in the parallel paths of the receiver. Thus the addition of two parity checkers over the simplex system provides detection of all errors introduced in the data due to a single fault in the parallel paths of the data. This is shown schematically in Figure 6a and b.

The paths through which data flows serially pose a bigger problem for concurrent checking. No convenient techniques exist for checking the parallel to serial or serial to parallel conversions required. A single fault could introduce multiple errors in the data. For eight bits of data, an error detecting code would be required to detect up to eight errors. The extent of additional hardware makes duplication attractive. This part of the system is thus duplicated.

The control circuitry in the transmitter provides timing pulses for shifting the data. The start of the control operation follows a high to low transition on the EOC line while the end brings about a low to high transition. Since the period between the two transitions is predetermined by the clock rate, a check of this period would detect a fault in the control hardware. This is easily realized using a monostable. The control circuit in the receiver can be similarly checked.

The auxiliary part of the controller cannot introduce any error in the data and is thus ignored for the purpose of concurrent checking.

Concurrent checking can be achieved by the techniques discussed. The different error signals generated would have to be ORed and used to set an error flip-flop. Some of the error signals generated would require gating with strobe pulses generated by the existing control circuits to insure that they are sampled only at specified times. The output of this flip-flop would provide an interrupt signal to the processor if an error is detected and this would start a CPU check sequence. Repeated error indications from the receiver flags can also be used to initiate the CPU check.

Concurrent checking techniques discussed are not exhaustive but provide significant protection for the nominal amount of additional hardware required.

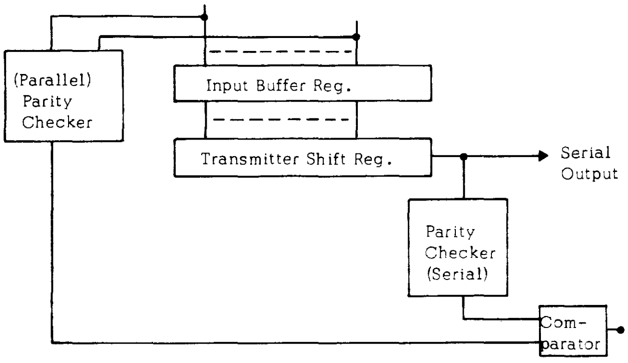


Figure 6a—Transmitter

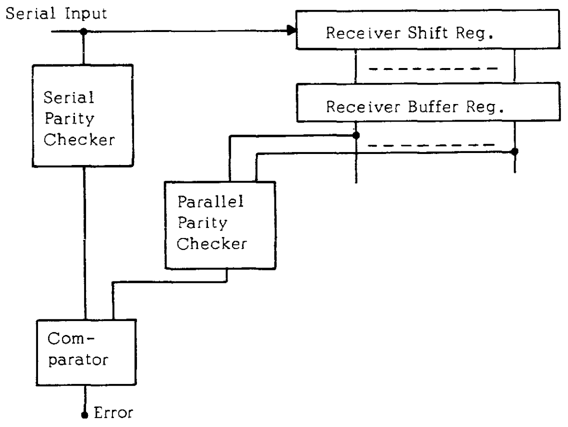


Figure 6b—Receiver

DEGRADED OPERATION AND SPARE SWITCHING

When the CPU test detects a fault in the controller, two possible corrective actions can be taken. These are discussed individually in the following paragraphs.

Degraded operation

The failure of the CPU to detect any faults after proper symptoms had initiated the check, indicates a fault in the stop bit generation circuit of the transmitter or the framing error detection circuit of the receiver. The fault could also be external to the controller residing in the peripheral. Step 4 of the test sequence isolates the fault to the parity hardware of the controller. For all these situations, the remaining controller circuit is established as fault free and a degraded operation is possible.

To realize this degraded operation, slight modifications in the software are required. The CPU normally reads the status register of the receiver and checks the error flag positions for errors in the data. Once the CPU test establishes the possibilities of degraded operation of the controller, this step of the program has to be disabled. A flow diagram which shows one possible way of doing this is shown in Figure 7.

Spare switching

Faults occurring in the control circuit or in the data paths make further use of the controller impossible. In this situation the defective controller can be switched out of the system and a new one switched in. A specific situation where this is realized mainly through software modifications is shown in Figure 8.

When several interface controllers are used in a system, a memory mapped I/O mode is frequently used. The CPU or

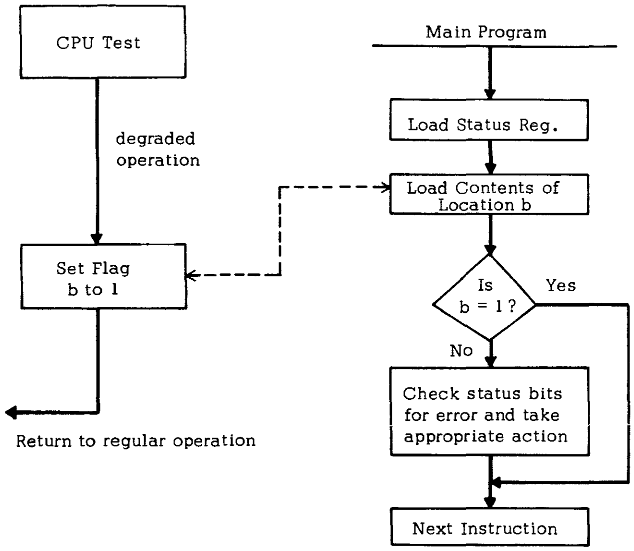


Figure 7—Flow chart for degraded operation

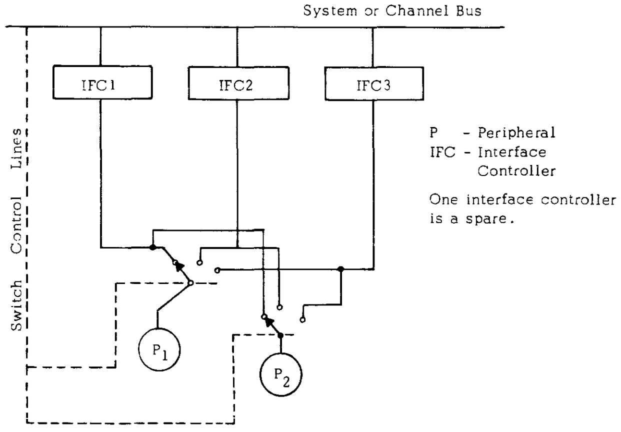


Figure 8—Hardware for spare switching

the channel hardware thus recognizes the controllers as particular address locations and spare switching can be performed in the following manner.

(1) Use of a routine to change the address of the defective controller to that of a spare.
(2) Switching of the peripherals connected to the defective controller to the spare.

The switching hardware required in step 2 is quite nominal since the serial communication between the controller and the peripheral require very few lines.

CONCLUSION

The use of hardware and software techniques in the proposed approach makes it, in several ways, more advantageous than some of the existing approaches.

Most existing systems use software diagnostics to detect faults. This has the disadvantage that detection is not achieved until the diagnostic tests are run. Failures are thus allowed to propagate if they occur between tests. Since peripheral devices are often electro-mechanical, these failures may cause severe damage to the equipment.[3]

The proposed approach incorporates concurrent checking to detect the propagation of erroneous data. Since the detection is accomplished prior to the end of transmission of a complete character, effective steps may be taken, if an error is detected, to make the peripheral abort the data. This can be done by transmitting invalid marker bits which follow the data or by inhibiting the flag which indicates the end of transmission to the peripheral. In either case, the peripheral will not act on the data and any damage that may have been caused by the erroneous data is prevented.

The use of concurrent checking also results in saving of CPU time. Diagnostic tests are no longer required to be run at regular intervals. The tests need only be run when a fault is suspected. This is indicated to the CPU through the use of interrupts, and the processor is thus completely freed to execute its normal functions. The concurrent checking hardware, however, adds to the cost of the simplex system, but this increase is very nominal.

Another commonly used approach to improving reliability is that of duplication. A duplicated system costs at least 100 percent more than the simplex system and often even higher due to the cost of the matching circuits. This approach guarantees the detection of all single faults, as soon as they occur. Isolation of the fault, however, requires software diagnostic tests or additional hardware.

An analysis of the final system showed that the increase in hardware due to the concurrent checking circuits is approximately 30.8 percent over the simplex system. Additional hardware required to facilitate testing is equivalent to 0.9 percent. The total redundancy proposed in the system is thus approximately 31.7 percent. For a redundancy of about 30 percent, the proposed approach provides detection of all failures in the data and most of the control hardware which could affect the propagation of the data. The auxiliary hardware is not checked since it cannot result in any serious failure.

Since the detection of all data failures is achieved and since corrective steps may be taken to prevent the peripheral from acting on the data, the advantages to this approach are comparable to that of duplication. The amount of redundancy in the proposed approach is much less.

The proposed approach becomes even more attractive than duplication when fault correction is considered. As pointed out earlier, several controllers are normally used in a computer system. The duplication procedure requires each individual controller to be duplicated, while this approach requires about 30 percent redundancy in each controller and one spare controller to cover all the working controllers.

REFERENCES

1. Signetics, *Integrated Circuits Data Book*, Signetics Corporation, 1972.
2. Motorola, *M6800 Microprocessor Applications Manual*, Motorola Inc., 1975.
3. Usas, A. M., *Error Management in Digital Computer Input/Output Systems*, Ph.D. Dissertation, Stanford University, Stanford, California, 1976.

Modular redundancy without voters decreases complexity of restoring organ

by P. T. DESOUSA
Rockwell International
Richardson, Texas

and

F. P. MATHUR
Wayne State University
Detroit, Michigan

ABSTRACT

Fault-tolerant modules have usually been implemented through the use of static fault-masking or dynamic spare-switching. But a new class of MR (Modular Redundancy), the Responsive schemes, promises higher reliability levels and more efficient implementations for medium to high degrees of redundancy. In particular, Siftout Modular Redundancy (SMR) does not use voters and provides a 2-out-of-N redundancy with a very simple restoring organ. The complexity of implementation is analyzed for several MR's and reliability figures are compared for three 2-out-of-N schemes. SMR is shown to have the best performance.

INTRODUCTION

Fault-tolerant digital systems are usually achieved by the use of Modular Redundancy (MR) techniques. The module to be made fault-tolerant is replicated a number of times. Each one of the replicas will be called a channel. The number of channels is the degree of redundancy. These identical channels constitute the "executive organ." The "restoring organ" is made up of the additional circuits necessary to perform the functions of fault-masking and/or recovery over the executive channels.

Fault-masking and spare-switching are the two best known fault tolerance techniques.

- Static MR provides fault-masking. All channels are on-line throughout the mission time. The failing of a channel is "masked" by the good channels, keeping the overall structure output correct.
- Selective MR provides spare-switching. There is a functional core of on-line channels and a standby bank of spare channels. Whenever an on-line channel fails, one of the spare channels replaces it.

Responsive MR schemes do not quite fit in any of the two mentioned categories. There are no spare channels and all channels start the mission on-line. But upon the occurrence of a failure, the structure reconfigures itself. The contribution of the failed channel is reduced or eliminated. Pierce[1] was the first to propose a scheme of this type, using an adaptive restoring organ. The system output is a weighted vote of the channel outputs. The weight depends on the error probability of the correspondent channel. A weighted-input vote-taker is implemented with linear threshold elements, that perform "linearly separable Boolean functions."[2] Adaption circuits estimate the error probability of each of the channels and use the estimate to set the vote-weight. The threshold vote-taker and the adaption circuitry make up the restoring organ, called "decision element" by Pierce. Three questions arise:

(1) What vote-weight to use?

   (a) Continuous, in proportion to the error probabilities;
   (b) Quantized. The vote-weight is either 0 (the channel is disconnected) or 1 (the channel is connected).

(2) How to estimate the error probability?

   (a) Using reliability information generated in the same source that generates the digital information;
   (b) Counting the errors occurred in a time cycle and setting the weight accordingly;
   (c) Counting the errors periodically and incrementing the previous cycle data;
   (d) (for quantized vote-weights) Disconnect a channel whenever the error count exceeds a given threshold.

(3) How to detect errors?

(a) From conditions in the channel itself. Correctly functioning circuits can be arranged to display properties different from properties of circuits with faults;
(b) By comparison with an externally supplied correct answer;
(c) By comparison with the output of the restoring organ (feedback of information). The output of the restoring organ is assumed to be correct.

Pierce analyzed all these alternatives. Answers 1(b), 2(d) and 3(c) have been the most appealing. Goldberg, et al.,[3] and Losq[4] designed implementations for schemes with those answers. Alternative 3(b) is used, for example in the model-assisted BMR of Devaney.[5]

Siftout MR[6] answers question 3 with a new alternative:

(d) By comparing the outputs of the channels with one another.

Adaptive redundancy uses threshold-rule in the restoring organ. Other Responsive configurations (Monitored Majority structures) use majority-rule. NMR/Simplex schemes[7] are examples of Monitored Majority Redundancy.

Siftout MR does not use a vote-taker. The restoring organ merely discards any channel that does not agree with the majority. The module output is thus equal to the output of any of the channels that remain on-line.

SIFTOUT MODULAR REDUNDANCY (SMR)

When using a Siftout configuration the system is organized into N identical channels, where N is any integer. The channels are synchronized with one another and perform simultaneous operations. Each channel is active as long as it is fault-free. Whenever one of the channels fails, its contribution to the system output stops. The system becomes an (N−1) redundancy scheme. Upon the occurrence of a new failure, the process repeats itself.

SMR has a fault tolerance F=N−2. (N−2) channels can fail and the module will still operate correctly. When the module is reduced to two channels and one of them fails, the system is unable to detect which one failed. SMR is a 2-out-of-N structure, or more emphatically, an N-down-to-two redundancy.

SMR IMPLEMENTION

To implement a Siftout redundant structure, a Checking Unit is placed at the outputs of the N channels. The Checking Unit compares the output signals. If one of the signals disagrees with the others, the correspondent channel is "sifted out." The signal of the "good" channels is selected without need for voting. The diagram of Figure 1 shows the main parts of the Checking Unit: the Comparator, the Detector and the Collector.

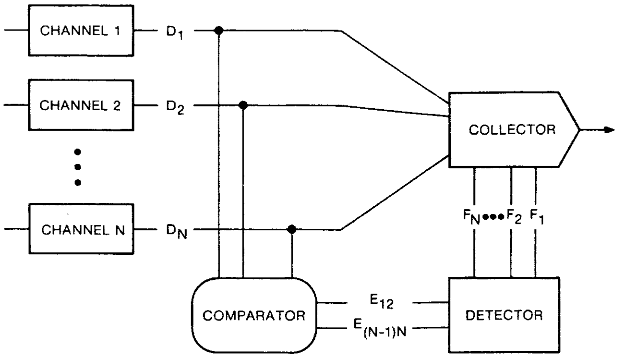


Figure 1—Siftout redundancy

The Comparator is a set of $\binom{N}{2}$ EXCLUSIVE OR gates, that checks the N channels against each other. It is depicted in Figure 2 for the case of N=4.

The Detector (Figure 3) is a sequential circuit with $\binom{N}{2}$ OR gates and N AND gates. The signal F_i is equal to 0 when channel i is fault-free. F_i is equal to 1 when channel i has failed. For example let channel 1 be the first channel to fail. It will disagree with the other channels, causing lines E_{12}, E_{13} and E_{14} to hold a logical value 1. Line F_1 will then be set to 1 and the feedback loop will force it to stay that

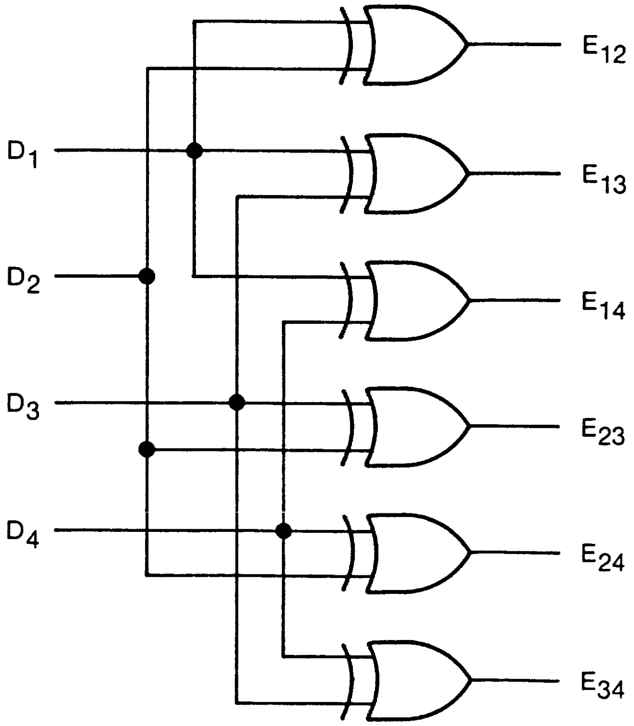


Figure 2—Comparator for a 4-channel siftout redundancy

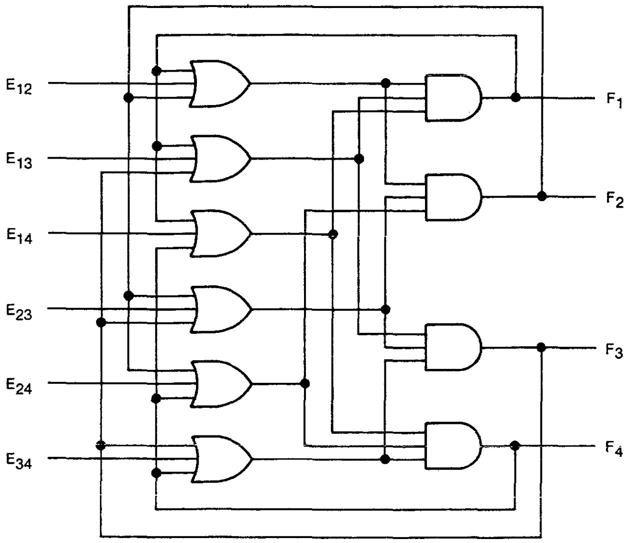


Figure 3—Detector for a 4-channel siftout redundancy

way. A flip-flop can be added in the feedback loop, if a reset-retry procedure is desired. Such a flip-flop would make the structure tolerant to transient failures and would facilitate initial checkout.

The final step is the Collector, with N OR gates and one AND gate. Each good channel feeds one input to the AND gate. Each bad channel provides a logical value 1 as input to the AND gate. The output of the AND gate is the correct output of the system, provided that at least two channels are good. Figure 4 shows the Collector when N=4.

OTHER MR TECHNIQUES

SMR is now compared with other redundancy techniques that have been used to provide ultra-reliable digital systems.

Static MR

TMR (Triple Modular Redundancy)

In the basic TMR configuration, the system is organized into three identical channels that feed a voting element. The voting element compares the output signals of the three channels and selects the signal on which the majority of the channels agree.

The TMR organization is one of the oldest forms of redundancy and has been considered the most promising for universal application.[8] However, the process that makes TMR fault-tolerant also makes it difficult to maintain. To analyze the performance of a malfunctioned system, error detection and fault isolation are necessary. The TMR majority voting mechanism masks a bad channel but at the same time complicates the detection of the error. To

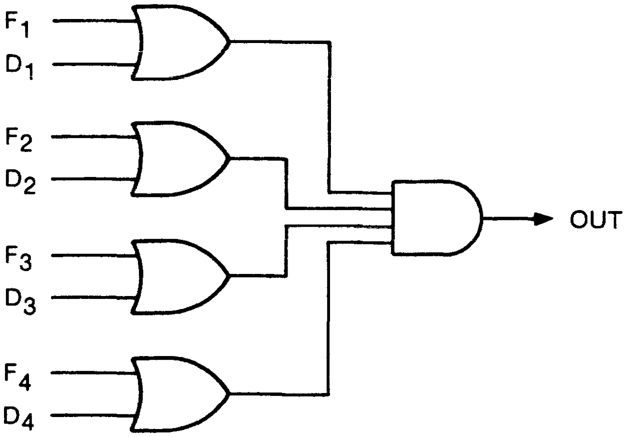


Figure 4—Collector for a 4-channel siftout redundancy

overcome this difficulty, extra hardware has been incorporated into TMR organized computer systems.[8,9]

A Siftout configuration with three channels (Figure 5) has the same fault tolerance as a TMR configuration. It already has the built-in capability of automatic error detection and fault isolation. The value of the variables F_i provides immediate information about the state of channel i ("good" if $F_i=0$, "bad" if $F_i=1$). This is an important advantage when redundancy is considered for easing maintenance operations and improving availability.

NMR (N-tuple modular redundancy)

In an NMR system each nonredundant module is replicated an odd number (N) of times. The N identical channels feed a majority voting element. The structure works as long as a majority of the channels is fault-free.

The fault tolerance of an NMR configuration is only $F=(N-1)/2$. The fault tolerance of a Siftout configuration with the same number of channels is $F=N-2$. Comparing the NMR voting unit with the Siftout checking unit, the voter is found to be less complex than the checker for small values of N, but the situation inverts as N increases. (See Table 1). In addition, NMR has the same disadvantages of TMR, of which NMR is a generalization. However, NMR

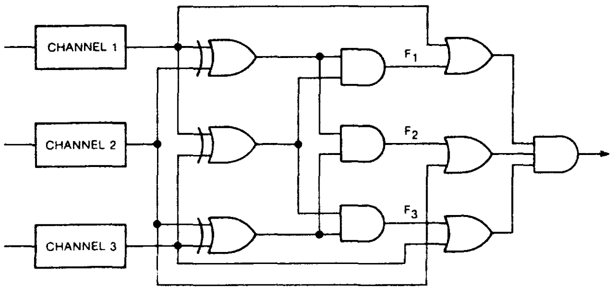


Figure 5—Siftout redundancy with three channels

TABLE I—Equivalent Number of Gates for Restoring Organs

		HMR				
		Majority Voter		Threshold Voter		
N	NMR	TMR Core	5MR Core	TMR Core	Self-Purging MR	Siftout MR
3	4	—	—	—	34	13
4	—	71	—	57	47	21
5	13	91	—	80	63	31
6	—	111	144	102	78	43
7	41	131	172	126	95	57
8	—	151	200	151	113	73
9	145	171	228	177	132	93

can mask some multiple failures, while SMR requires that no more than one channel fails at a time. If the channels have been dormant, sufficient for several failures to have developed, then several of the signals may be erroneous when the system becomes active. Under this circumstance, voting becomes valuable, for systems with fivefold or higher redundancy.

Selective MR

HMR (hybrid modular redundancy)

Hybrid redundancy has been developed as a means to achieve greater reliability and longer times of failure-free operation than those achieved by TMR or NMR systems.[10] It consists of an NMR core and S standby spare channels. The restoring organ includes besides the NMR vote-taker, a disagreement detector and a switching network. If the disagreement detector finds that the output of a channel in the NMR core does not match the output of the vote taker, the switching network replaces it by one of the standby channels.

Hybrid redundant systems have the advantages of NMR systems (instant internal fault-masking) and Standby systems (increased reliability for long time missions). They yield a more efficient hardware utilization than the NMR systems, due to a greater fault tolerance. The implementation of the restoring organ of a Hybrid system is not straightforward and requires a fairly complicated switch.[11]

Siftout Redundancy appears as a real challenger. It has a fault tolerance as high or higher than Hybrid Redundancy. And it has a simpler implementation. HMR is able to use dormant spare channels, but that capability does not provide a significant increase in reliability.

Responsive MR

Self-purging redundancy

In the Self-Purging MR,[4] there are N on-line channels feeding a threshold vote-taker with threshold equal to 2. The vote-taker output is compared with the channels outputs for disagreement detection. If a disagreement is detected, the faulty channel output is forced to a logical zero.

Like Siftout MR, this is a 2-out-of-N strategy. The Self-Purging restoring organ requires N flip-flops, that increase substantially its complexity. However, these flip-flops can also be used to handle transient errors and for restart procedures.

COMPLEXITY OF RESTORING ORGANS

The complexity of the restoring organs for the MR's discussed in the paper are displayed in Table I. The following assumptions were used:

(a) A gate is considered to be any one of the following logic functions: AND, OR, NAND, NOR, Exclusive OR, and Inverter.[12]

(b) A J-K or R-S flip-flop is equivalent to 8 gates.[12]

(c) NAND/NOR gates are available with up to 8 inputs.

The equivalent number of gates were calculated using the following expressions:

(i) *NMR* (all-NAND implementation):

$$\binom{N}{M}+1 \tag{1}$$

where $M=(N+1)/2$.

(ii) *HMR* (iterative cell array implementation[11]):
TMR core ($S=N-3$):
Majority voter:

$$\left[\binom{3}{2}+1\right]+(3S+8)+(9S+27)+(8S+12)=51+20S \tag{2}$$

Threshold voter:

$$\left[\binom{3+S}{2}+1\right]+(3+S)+(9S+27) \tag{3}$$
$$+(7S+3)=34+17S+\binom{3+S}{2}$$

5MR core ($S=N-5$):
Majority voter:

$$\left[\binom{5}{3}+1\right]+(5S+17)+(9S+45)+(14S+43)=116+28S \tag{4}$$

(iii) *Self-Purging* [4]:

$$(8+2)N+\binom{N}{2}+1=(N^2+19N+2)/2 \tag{5}$$

(iv) *Siftout MR*:

$$2*\binom{N}{2}+2N+1=N^2+N+1 \tag{6}$$

TABLE II—Applicability Bounds for 2-out-of-N MR's

N	Minimal R_0	Minimal R_R
3	0.5	0.8889
4	0.2324	0.7248
5	0.1311	0.6028
6	0.0836	0.5137

RELIABILITY ANALYSIS

Reliability of 2-out-of-N systems

Out of the schemes shown in Table I, Siftout MR, Self-Purging MR, and Hybrid MR with TMR core are all 2-out-of-N strategies. Regarding the restoring organ as a series element in the reliability block diagram, the reliability of a 2-out-of-N MR is:

$$R=R_E{\cdot}R_R$$

$$R=\{1-[(1-R_0)^N+N(1-R_0)^{N-1}R_0]\}R_R$$

$$=\{1-(1-R_0)^{N-1}[1+(N-1)R_0]\}R_R \qquad (7)$$

where R_E is the reliability of the executive organ, R_0 is the reliability of a single channel and R_R is the reliability of the restoring organ.

Applicability bounds

The crossover point is the minimum value of the reliability of a nonredundant unit for which there is improvement in the reliability using a redundant system. It is geometrically interpreted as the point where the curves for the redundant and the nonredundant system cross each other.

The reliability of a nonredundant unit (simplex system) is equal to the reliability of a channel R_0. The crossover point (R_{cp}) of a 2-out-of-N system is the nontrivial root of the equation:

$$1-(1-R_{cp})^{N-1}[1+(N-1)R_{cp}]=R_{cp} \qquad (8)$$

R_{cp} gives the lower bound of applicability of a 2-out-of-N system. The reliability cannot be improved by using redundancy when $R_0<R_{cp}$, whatever the value of R_R.

Similarly, there is a value of R_R below which the reliability cannot be improved over the simplex design, whatever the value of R_0. This lower bound for R_R is the minimum of the function R_0/R_E. Table II shows this minimal R_R and the minimal $R_0(R_{cp})$ for several values of N.

TABLE III—Reliability of 2-out-of-4 Systems with 1000 Gates/Channel

		R		
R_0	R_E	Siftout	Hybrid	Self-Purging
0.5	0.6875	0.6776	0.6590	0.6655
0.7	0.9163	0.9095	0.8966	0.9011
0.9	0.9963	0.9941	0.9899	0.9914
0.95	0.9995	0.9984	0.9964	0.9971
0.99	0.999 996	0.9998	0.9994	0.9995
0.999	0.999 999 996	0.999 98	0.999 94	0.999 95

TABLE IV—Reliability of 2-out-of-5 Systems with 1000 Gates/Channel

		R		
R_0	R_E	Siftout	Hybrid	Self-Purging
0.5	0.8125	0.7952	0.7676	0.7778
0.7	0.9692	0.9586	0.9413	0.9477
0.9	0.9995	0.9963	0.9909	0.9929
0.95	0.999 97	0.9984	0.9958	0.9967

Reliability comparison

In order to compare the performance of the three 2-out-of-N MR's, an analysis was made based on the values from Table I.

If r is the reliability of a single gate and each channel has G gates,

$$R_0=r^G \qquad (9)$$

If g is the number of gates in the restoring organ,

$$R_R=r^g=R_0{}^{g/G} \qquad (10)$$

Tables III to V present values of the reliability R for the three MR's discussed, with selected values of R_0. It was assumed G=1000, that is generally considered as a typical value.[13] The HMR implementation considered was the TMR core, threshold voting. The numbers show that, given a fixed R_0 and a fixed G, there is for any MR a maximum number of channels N_{max} to consider. Increasing the degree of redundancy above N_{max} will not increase the system reliability. This result has long been known for the HMR.[13] Although there are no drastic differences among the three MR's discussed, Siftout presents the best reliability performance.

CONCLUSIONS

Responsive schemes, and in particular SMR, have been shown to have significantly better performances than Static or Selective schemes. They provide higher fault tolerance than Static schemes, and have the added value of fault detection capability. Their implementation is much simpler than the Selective schemes, enabling higher limits of reliability.

SMR does not use voters, and has a very straightforward

TABLE V—Reliability of 2-out-of-6 Systems with 1000 Gates/Channel

		R		
R_0	R_E	Siftout	Hybrid	Self-Purging
0.5	0.8906	0.8645	0.8287	0.8438
0.7	0.9891	0.9740	0.9530	0.9619
0.9	0.999 95	0.9954	0.9890	0.9918

implementation. It was favorably confronted with the older TMR, NMR, HMR, and Self-Purging. SMR is particularly suitable for systems with high availability requirements and systems with high reliability requirements over a long period of time.

The SMR reliability was compared with two other 2-out-of-N schemes, Self-Purging and the iterative cell array, threshold voter implementation of TMR + spares. Lower bounds were presented for the channel and the restoring organ reliabilities. There is a maximal number of channels for each instant of time, above which the reliability of any scheme starts to degrade. This means that there is an upper limit for the reliability that can be achieved with an MR over each mission time, irrespective of the degree of redundancy.

REFERENCES

1. Pierce, W. H., "Adaptive Vote-Takers Improve the Use of Redundancy," In *Redundancy Techniques for Computing Systems*, pp. 229-50. Edited by R. H. Wilcox, and W. C. Mann. Washington: Spartan Books, 1962.
2. Pierce, W. H., *Failure-Tolerant Computer Design*. New York: Academic Press, 1965.
3. Goldberg, J., K. N. Levitt, and R. A. Short, "Techniques for the Realization of Ultrareliable Spaceborne Computers," Final Report—Phase 1, Stanford Research Institute Project 5580, Menlo Park, California, September, 1966.
4. Losq, J., "A Highly Efficient Redundancy Scheme: Self-Purging Redundancy," *IEEE Transactions on Computers,* Vol. C-25, June 1976, pp. 569-578.
5. Devaney, M. J., "Fault Diagnosis and Self-Repair in Operational Synchronous Digital Systems," Ph.D. Dissertation, University of Missouri-Columbia, June, 1971.
6. deSousa, P. T. and F. P. Mathur, "Sift-out Modular Redundancy," submitted for publication.
7. Mathur, F. P. and P. T. deSousa, "Reliability Models of NMR Systems," *IEEE Transactions on Reliability*, Vol. R-24, June 1975, pp. 108-113.
8. Ball, M. and F. Hardie, "Self-Repair in a TMR Computer," *Computer Design*, Vol. 8, February 1969, pp. 54-57.
9. Hight, S. L. and D. P. Petersen, "Dissent in a Majority Voting System," *IEEE Transactions on Computers*, Vol. C-22, February 1973, pp. 168-171.
10. Mathur, F. P. and A. Avizienis, "Reliability Analysis and Architecture of a Hybrid-Redundant Digital System: Generalized Triple Modular Redundancy with Self-Repair," *AFIPS Conference Proceedings (Spring Joint Computer Conference)*, Vol. 36, May 1970, pp. 375-383.
11. Siewiorek, D. P. and E. J. McCluskey, "An Iterative Cell Switch Design for Hybrid Redundancy," *IEEE Transactions on Computers,* Vol. C-22, March 1973, pp. 290-297.
12. "Reliability Prediction of Electronic Equipment," Military Standardization Handbook MIL-HDBK-217B, Department of Defense, U.S.A., September 1974.
13. Ogus, R. C., "Fault-tolerance of the Iterative Cell Array Switch for Hybrid Redundancy," *IEEE Transactions on Computers,* Vol. C-23, July 1974, pp. 667-681.

A study of intermittent faults in digital computers

by ÖMÜR TAŞAR and VEHBÍ TAŞAR
University of Detroit
Detroit, Michigan

ABSTRACT

Definition of intermittent faults in digital computer systems and their possible causes are given. The effects of intermittent faults on the performance of systems and alternatives to overcome these effects have been examined. Present attempts to cope with the intermittent failures and future research areas have been explored.

INTRODUCTION

The advanced system architecture of the current computers has had an impact on reliability and maintainability concepts. Although there have been evolutionary inventions in hardware and software built into computers, the complexity of the whole system makes the problem of fault isolation difficult. It is of vital importance to keep a system running as well as producing the system with the utilization of the best engineering knowledge. Hence one should be concerned with providing high availability and easy maintainability at the design stage. Due to insufficient attention devoted to this aspect, many of the machines in the field today have been suffering from poor reliability and maintainability.

In the so-called space age, it is hard to believe that systems can go down unexpectedly or cause interrupts for no apparent reason and groups of people wander around for many hours, even days to find the problem. All the maintenance procedures embodied in the machine do not give any hint. The field engineer does his best and the machine does not respond. At some point in time, the machine chooses to run due to an unknown action and there is almost no information on why, how, where and when this happened. This dramatized description is quite realistic for a number of current installations.

The problem causing the above situation is called the intermittent fault. In most systems 80 to 90 percent of faults are estimated to be intermittent. These faults account for more than 90 percent of total maintenance expense because they are difficult to detect and isolate.

INTERMITTENT PROBLEM DEFINITION

Intermittents were first defined as faults over which the user had little or no control. This definition still holds true. Today some people define intermittents as failures that are not reproducible since the conditions causing or surrounding an intermittent are often not known. The inherent inconsistency in the occurrence of such faults is a major hindrance to a detailed analysis of these conditions. Intermittents can be defined as random failures that prevent the proper operation of a unit for a short period implying that the duration of failures is not long enough for the application of a test procedure designed for permanent faults.

Some intermittents are known to occur due to external effects such as temperature, humidity, vibration, power fluctuation, pollution, pressure, and electromagnetic fields. Assuming that the system runs under specified external conditions, this class of faults constitutes a small portion of intermittents.

The nonenvironmental intermittents, namely the faults that are within the system, seem to be the basic source of irritation to the user. There is a difference of opinion as to the behavior of this class of intermittents. Some believe there actually exists a permanent failure which is stimulated only under a certain sequence of events; consequently, it appears as intermittent. The other opinion is that portions or components of the system malfunction intermittently. For example, loose connections, resistance variations and partially defective components may cause such faults. Examples of both opinions have been observed in practice. There are also intermittents that eventually become permanent failures. Such intermittents are caused by deteriorating or aging components. Once they become permanent, they can be detected by existing procedures. However this transition can take from a few minutes to several months during which the frequency of intermittents increases intolerably. To speed up this transition, environmental conditions can be drastically changed to unfavorable levels. Sometimes this technique, known as the stress technique, works but it may inflict new damage in other parts of the system.

The source of nonenvironmental intermittents can be software as well as hardware. It has not yet been established which causes the major part of such intermittents. However, the general trend is to treat them as hardware oriented.

REVIEW OF LITERATURE ON INTERMITTENT FAULTS

The first experiments on the effects of intermittent faults were conducted by Ball, Hardie and Suhocki in 1966.[1,2] A highly sophisticated logic simulator was developed for the purpose of normal and/or fault simulation of the Saturn V Launch Vehicle aerospace computer which was being designed by IBM. The computer is a binary, fixed point, serial machine employing triple modular redundancy to provide very high reliability. The simulator was capable of analyzing single and multiple, permanent and intermittent faults. Nearly 800,000 intermittents were simulated to obtain a reliable statistical sample. The duration of these intermittents varied between 500 nanoseconds (one clock time) to five milliseconds. They were injected at randomly selected points of combinational and sequential logic circuits in the arithmetic-instruction and multiply and divide units, at the execution time of the simulator.

The analysis of the intermittents included a record of the time of occurrence, time and number of detections and the number of intermittents that caused a difference from the correct output. The probability of detection was calculated based upon this information.

The authors revealed interesting as well as important conclusions from the stimulation output. These can be summarized as follows. Only 8.3 percent of the total intermittents injected caused the computer to perform incorrectly. The combinational logic was less sensitive to intermittents. In other words, the probability of detection in combinational logic was smaller compared to sequential logic. A single intermittent fault of one clock period was almost undetectable. Single intermittents of longer duration had very low probabilities of preventing the correct operation. The probability of detection was directly proportional to the duration of intermittents and highly dependent on computer modules.

The authors experienced that field failures in aerospace computers tend to be intermittent. In commercial computers, intermittents have been estimated to constitute 90 percent of all field failures. Hence, maintenance cost is mainly due to intermittents. The authors believed that most intermittents were due to insufficient testing at the factory. They proposed to develop better detection procedures as a partial solution to the intermittent problem. The other approach was to design equipment insensitive to intermittents introducing retry and/or redundancy.

IBM has also tried to deal with intermittents in systems 360 and 370.[3,4] Here the objective was to reduce the number and length of unscheduled maintenances and to minimize the impact of intermittents on system availability. Automatic error recording at the instant of discovery followed by error recovery was proposed as a feasible solution. The so called Recovery Management of IBM follows several steps to achieve the above objectives.

Instruction retry can be affected in the I/O area, central processor and main storage areas. Selective termination enables the system to examine the failing environment while all other jobs are running. These functions are incorporated in Recovery Management in four steps. First, a functional recovery is attempted. If the retry of the interrupted operation is successful, the fault becomes transparent to the user. If not, the next function is System Recovery where a selective termination is effected to analyze the failure. Then the system-supported restart is tried without stopping for repair. If a stop is required, system repair utilizes all the detailed error analysis records. To perform all these steps, Recovery Management has I/O Device/Unit Recovery, Channel Recovery, I/O Recovery Management, CPU/Processor Storage Recovery, System Associated Recovery and Error Record Retrieval facilities.

Honeywell's proposal to overcome the intermittent problem was also retry. The Honeywell 6000 was implemented as a retryable processor. Maestri discusses the problems encountered and the design proposal to avoid such problems.[5] Some instructions cause a destructive read of a memory location. Therefore, it is necessary to restore that memory location before retry can be attempted. For this purpose, a buffer register has to be added. Data can be held in this register until error detection and correction codes report correct data recovery. Instruction retry for a MOVE may cause problems if a data block overlays another block, partly destroying the latter. To avoid this, snapshot registers are added where at the occurrence of a fault the state of the cycle control flags and address register could be saved. Snapshot registers can also be used as a diagnostic aid. To retry instructions with indirect addressing, several methods are proposed. One way is to obtain a pointer to the first indirect word. Restoring should take precautions against parity errors in the updated word and double updating. The second approach requires a scratchpad memory where the state of sequence control flags and memory addresses are saved for every cycle of an instruction. Software can then handle restoration of indirect words assuring no error. Multicycle instructions change the contents of registers with the speed of the adder cycle. In order to retry such instructions, intermediate registers are needed to protect the contents of primary registers and to keep up processor speed. These can be placed at the inputs of the adder on lines from memory and primary registers. Data can be held here until error checking is completed.

To solve problems due to instruction overlap, four instruction counters are added for simultaneously executed instructions. If an error occurs, the instruction counter of the failing cycle can be identified either selectively or by a program utilizing a failure flag included in the scratchpad memory.

The addition of the above mentioned registers increases the cost of implementation. However, the new processor is almost 100 percent effective in doing instruction retry. The cost advantage can be justified bearing in mind that instruction retry is 80 to 90 percent successful. Honeywell now even states, in sales literature, that the 6000 system is capable of handling 95 to 97 percent of all intermittent failures.

The other approach to coping with faults in digital computers has been the introduction of redundancy. Research has shown that Triple Modular Redundancy (TMR) can mask solid as well as intermittent faults. If TMR is applied at the module level, then the effect of a single intermittent can be permanent in a sequential circuit.[6] In case an intermittent induces a faulty state, it must be corrected with a resynchronization sequence before a second failure occurs. TMR has been used in some space projects; however, there has been no commercial application.

Totally self-checking check circuits have been designed. Such circuits are capable of detecting failures in themselves during normal operation.[7] If intermittents last long enough, they affect the output in such a way that they can be detected.

Theoretical modeling of intermittent faults has been attempted in recent years. Breuer presented a two-state first order Markov model to represent the intermittents.[8] Figure 1 shows the two states: the faulty state (N1) and the normal state (N2).

Since the Markov model is probabilistic, each state and every transition is associated with probabilities. With respect to intermittents, a circuit either operates normally or it possesses an intermittent. Let p be the probability of having an intermittent, then the normal state will have a probability of (1-p). To apply this model to any circuit, first of all the parameters, namely p, r, s, have to be estimated.

A time interval (T,T') is considered for fault analysis. Fault patterns are defined over this interval as follows: $d=(d_Q, d_{Q-1}, \ldots, d_2, d_1)$. The subscripts refer to the number of clocks in this interval. $d=(0, 0, \ldots, 0, 0)$ implies a normal circuit. $d=(1, 1, \ldots, 1, 1)$ represents a solid fault. $d=(0, 0, \ldots, 1, 0, 0)$ is the fault pattern of an intermittent occurring in the third clock period. The probability of each fault pattern can be calculated using the Markov model. A complete test set will be obtained if test sets are generated for all possible fault patterns. A modified d-algorithm is proposed for test generation that makes use of two types of gates called d-OR and d-AND. d-OR has the property that its output is a d, if and only if at least one of its inputs is a d or $\bar{d}$; otherwise its output is 0. In other words, if any input of d-OR receives a fault, it can propagate it. The output of d-AND is a d, if and only if all of its inputs have a d; otherwise, it is a 0. Test generation requires that the combinational logic be evaluated the same number of times as the number of fault patterns present. In sequential logic analysis, the circuit evaluation should be repeated for every fault pattern in every clock period within the time interval (T,T'). Hence, even for a small circuit the test generatiofi is a very lengthy procedure.

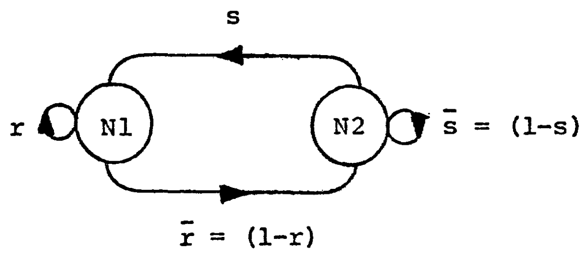


Figure 1—Markov model

Every test generated has a certain probability of detection. If a confidence level can be specified, then a lower bound for the number of tests that are required to detect an intermittent can be calculated.

Kamal's model is also probabilistic.[9] It is based on pattern recognition techniques. The analysis covers well behaved signal independent single intermittents in nonredundant circuits. A state space includes the state of being faultfree—the states possessing an intermittent that do not affect the output and the states possessing an intermittent that affect the output. The test set is the set of tests that are generated to detect permanent faults. The proposed solution is the repeated application of these tests. It is proven in the paper that an intermittent can be detected by infinite number of repetitions of a test. The probabilistic model is utilized to find the finite number of repetitions satisfying a confidence level. The model requires estimation of the probability that an intermittent occurs and the conditional probability of the output being affected given an intermittent has already occurred. The probability of detection given an intermittent is calculated according to Bayes' rule. After each application of a test, this figure is updated. This probability approaches 1 with an infinite number of applications. However, the procedure can be stopped once the confidence level is reached or the number of repetitions can be calculated in terms of the confidence level and the other probabilistic parameters. In most cases this number is very large. Kamal provides an optimization procedure by the use of integer programming.

Diagnosis procedure follows the same approach.[10] Here it is assumed that an intermittent has been detected. Using the test sets and the set of permanent faults, a coverage table is generated. From this table a test set is chosen that covers the permanent faults. These tests are repeatedly applied until one fails. This reduces the fault table and the possible faults. The above cycle is repeated until the fault set consists of a single fault. If there exists no test set to cover all possible faults, the diagnosis experiment is again terminated. The length of the experiment is calculated making use of the model and is shown to be finite. The fault table is also analyzed to obtain minimum diagnostic resolution. Optimization of the diagnosis procedure is not offered.

There are two major difficulties in the application of Breuer's and Kamal's proposals. First, it is very hard to obtain realistic estimates of the model parameters using the current available data. Even if an extensive experiment is conducted on a certain system, the parameters would apply to this particular system. The unpredictable behavior of intermittents may also impose an update on the parameters. The second problem is the huge number of tests required to detect and diagnose intermittents. In this respect, the proposed procedures may not be economically feasible. Even the detection of solid faults are far more expensive then desired.

Intermittent faults have been analyzed at the circuit level. Yen discusses intermittents due to noise in four phase MOS gates[11] which are widely used in LSI circuits. The problem is a charge redistribution problem. It is possible that the charge at some output node will decrease to critical levels causing intermittents. The situation does not fit the stuck-at model and testing will not be successful in detecting this type of failure. Yen proposes the addition of prechargers to avoid the problem. The number of the prechargers can be optimized by the use of computer-aided circuit simulation in order to decrease the additional hardware cost.

ALTERNATIVES FOR SOLUTION

No ideal method has been established to diagnose intermittent faults so far. Academic work offers solutions on restricted models that do not realistically represent the intermittent space. Industrial efforts concentrate on increasing the availability, hence they are mainly concerned with the minimization of the effects of intermittents. Up to now, the former has found no application. On the other hand, commercial manufacturers introduced retryable systems that recover from intermittents in a very short period. Although retryability is not a method to diagnose or to avoid the intermittents, it seems to be the only feasible way of recovering from such faults with minimum impact. Indeed, it is difficult to detect and diagnose something that goes on and off unpredictably, and retryability seems to be the only solution at the systems level currently and in the near future. Intermittents at the circuit or card level should be viewed as a completely different problem. At factory testing, intermittents account for only 30 percent of all faults, whereas in the field they are estimated to constitute 90 percent of all faults. Furthermore, the consequences of intermittents occurring at the system level are far more serious and costly compared to their effects at the testing level. Therefore, our efforts should concentrate on methods dealing with system intermittents. Here we discuss retry and other possible alternatives and where and how they can be effective in fighting intermittents.

Instruction retry is not a new concept. It has been a regular procedure that is carried out whenever an error occurs in the I/O area in reading or writing a tape. The new attempt is to extend this feature to central processor and memory units. IBM has been working on this since 1967; Honeywell since the early 70's. Burroughs has recently expended efforts on making its large scale processors retryable. The outcome indicates that these systems can survive 95 percent of intermittents. 100 percent effectiveness is also possible. Since the chance of success in retrying is .8 to .9, the system will have 75 to 85 percent less faults that are apparent to the outside world. In other words, instead of giving an interrupt, the system continues to operate successfully with a few microseconds or milliseconds of delay, caused by the execution of retry. Recovery alone does not help to isolate the intermittent, but an immediate remedy is achieved with respect to the operation of the system. Adequate error recording serves as a diagnostic aid during scheduled maintenance. Utilization of these records toward a test program will be discussed later.

Retry requires both hardware and software support. Intermediate and snapshot registers and scratchpad memories have to be added in order to be able to save the status of the system prior to an intermittent. Only then can the system go back and retry the failing state. Even with the addition of the sophisticated retry software, the implementation is not costly compared to the maintenance cost due to intermittents.

Today it is common practice to use stress testing to change an intermittent to a solid failure. Marginal voltage and timing conditions can be set or thermal stress can be applied such that slow rise times, low switching thresholds and race conditions are amplified. If test and diagnostic programs are run during the application of mechanical stress, faults due to loose connections, defective connectors or printed circuits can be caught. The disadvantage of this method is the dedication of the processor to maintenance procedures as well as the possible infliction of new damage due to thermal and mechanical stress. By all means retry is a better approach than this technique.

Retry is aimed at continuing the operation of the system. Whether it is successful or not, a detailed error recording is a major part of this scheme. If retry is not successful, the analysis of the error recording prior and at the instant of error occurrence would assist the personnel to determine the cause of the error in an easier and faster way. If retry is successful, then the error recording dump would be analyzed during scheduled maintenance to determine the cause of the intermittent. It may be advantageous to replace or repair the part causing the intermittent to avoid future possible intermittents. Although retry requires a short amount of time, it is more desirable not to have any interruptions. If this policy is continued persistently, the rate of interruptions due to intermittents may gradually decrease.

Another important outcome of detailed error recording is the possible generation of a test program that diagnoses intermittent faults. The accumulated error recording of a certain period may well include information about the most frequent intermittents such as the rate of occurrence, the effect of the intermittent, the symptoms and the fastest correction procedure. Similar to the detection procedure, given an intermittent and its symptom(s), we can run the above mentioned program and come out with an intermittent or a set of possible intermittents that can be examined. Isolation of the intermittent will be an easy matter if it actually belongs to the group of the intermittents under consideration. If not, the intermittent can be located by existing means and the test program can be expanded to include the new situation. As time progresses, one may come up with a complete intermittent diagnostic routine.

Realization of a diagnostic routine for intermittents requires extensive error recording on a retryable processor. Diagnosis can be achieved by another practical method called dynamic monitoring. As the name implies, dynamic monitoring involves a continuous scan of the machine to be able to detect a fault at the instant of occurrence. Otherwise, there is practically no way of reproducing the same event for purposes of detection and diagnosis. This scheme can be realized by placing test points and interrogating the values of these test points at every clock period. If the machine has retry capability, dynamic monitoring can be very successful in isolating intermittents. The values of test points can be continuously stored in a monitor memory, the size of which depends on the number of test points and the number of clock periods that have to be considered. In case of an interrupt, one of the test points will have a faulty value. These values can be stored in a memory for further reference. If the retry is successful, the particular test point will have the correct value. Comparing the two maps of test points before and after retry, the faulty test point can be isolated. To locate the fault causing component, it would suffice to examine the components feeding that test point.

Although dynamic monitoring requires test points and a memory of considerable size, the advantages are apparent. The fault is actually isolated during the operation of the system. All intermittents are detected regardless of their cause. If statistics are maintained, deteriorating components can be isolated and future faults predicted.

The feasibility, development and application of this technique has to be considered in conjunction with retry capability. If retry is not available, the technique can be implemented by straight dumping of the recorded memory. The basic problem in realizing this scheme is the optimal location of the test points.

The above discussion refers to the solution of intermittents at the systems level. Dynamic monitoring can also be employed in testers that are used to screen circuits and cards. At the circuit level, the models presented by Kamal and Breuer may also be applicable. Current automatic testers operate at a very high speed. Hence, a very large number of tests can be applied in a short period. If careful factory screening is desired, these techniques can be incorporated with the testers. Although the cost of testing will increase, less faults will escape the factory screening. Economic feasibility of these two models can be studied by the use of computer-aided simulation.

ACKNOWLEDGMENT

The authors wish to thank Mr. R. E. Stackhouse for his guidance and comments during the preparation of this paper. This work was supported by the Burroughs Co. through an internship program.

REFERENCES

1. Ball, M. and F. Hardie, "Effects and Detection of Intermittent Failures in Digital Systems," 1969 Fall Joint Computer Conference, *AFIPS Conference Proceedings,* Vol. 35, Montvale, N. J., AFIPS Press, 1969, pp. 329–335.
2. Hardie, F. H. and R, J. Suchocki, "Design and Use of Fault Simulation for Saturn Computer Design," *IEEE Trans. Computers,* Vol. EC-16, August 1967, pp. 412–429.
3. Carter, W. C., H. C. Montgomery, R. J. Preiss and H. J. Reinheimer, "Design of Serviceability Features for the IBM System/360," *IBM Journal*, April 1964, pp. 115–126.
4. Droulette, D. L., "Recovery through Programming System/360-System/370," 1971 Spring Joint Computer Conference, *AFIPS Conference Proceedings,* Vol. 38, Montvale, N.J., AFIPS Press, 1971, pp. 467–476.
5. Maestri, G. H., "The Retryable Processor," 1972 Fall Joint Computer Conference, *AFIPS Conference Proceedings,* Vol. 41, Montvale, N.J., AFIPS Press, 1972, pp. 273–277.
6. Wakerly, J. F., "Transient Failures in Triple Modular Redundancy Systems with Sequential Modules," *IEEE Trans. Computers,* Vol. C-24, May 1975, pp. 570–573.
7. Anderson, D. A., and G. Metze, "Design of Totally Self-Checking Check Circuits for m-out-of-n Codes," *IEEE Trans. Computers*, Vol. C-22, March 1973, pp. 263–269.
8. Breuer, M. A., "Testing for Intermittent Faults in Digital Circuits," *IEEE Trans. Computers*, Vol. C-22, March 1973, pp. 241–246.
9. Kamal, S., and C. V. Page, "Intermittent Faults: A Model and Detection Procedure," *IEEE Trans. Computers*, Vol. C-23, July 1974, pp. 713–719.
10. Kamal, S., "An Approach to the Diagnosis of Intermittent Faults." *IEEE Trans. Computers,* Vol. C-24, May 1975, pp. 461–467.
11. Yen, Y. T., "Intermittent Failure Problems of Four-phase MOS Circuits," *IEEE Journal of Solid-State Circuits,* Vol. SC-4, June 1969, pp. 107–110.

A "calibration-prediction" technique for estimating computer performance

by C. A. ROSE

Naval Ocean Systems Center
San Diego, California

ABSTRACT

One possible use of an analytical model of a computer system is to predict the increase in performance that results from a proposed modification of the existing system. With this aspect in mind, experiments were conducted that hypothesized situations in which a manager was considering upgrading his system and was interested in estimating the performance improvements to weigh against the anticipated costs. The basic concept was to run a benchmark program on the baseline system and to validate the model parameters. After validation, the model was used to predict CPU and I/O channel utilizations for the upgraded or reconfigured system. Lastly, the benchmark was rerun on the reconfigured system and measured utilizations were compared with predicted values from the model. There was good agreement between predicted and measured utilizations when adding a channel to an IBM 370/155, when reallocating files and adding two channels to an IBM 370/155-2, and when adding a channel to an IBM 370/168-1.

INTRODUCTION

There have been many queueing models of computer systems published in the literature, but relatively few studies have been validated on actual systems. Studies which have provided validations include Moore,[9] Baskett and Gomez,[1] Sekino,[12] and Bhandiwad and Williams.[3] Hughes and Moe[8] used a model to predict new CPU and channel utilizations, and then performed the reconfiguration to compare measured utilizations with predicted values. Their experiments consisted of reallocating files and increasing main memory on a UNIVAC 1108. Bhandiwad and Williams changed memory size on an IBM 370/145.

The concept of using an analytical model to predict performance after a reconfiguration, then actually performing the reconfiguration and comparing model and measured utilizations was the basic theme of the project reported in this paper. Results will be presented which show the applicability of the model for predicting computer performance in situations of reallocating files and increasing the number of channels. The experiment for increasing the number of channels has not been previously reported, and may be of interest to managers of systems which are currently I/O bound.

MATHEMATICAL SOLUTION OF CLOSED QUEUEING NETWORKS

The solution shown in this section is due to Chandy, Herzog and Woo.[6] Assume a class of closed queueing networks with exponential servers, with fixed N customers and with M queues indexed 1, 2, . . ., M. The service rate for the ith queue when there are k customers in the ith queue is $U_i(k)$, $i=1, \ldots, M$ and $k=1, \ldots, N$. The service discipline for all servers is first come-first served (FCFS). When a customer finishes service in queue i, he joins queue j with probability p_{ij} independent of the current state of the system, $i, j=1, \ldots, M$. The states of the system are m-tuples $(n_1, \ldots, n_M)$, where n_i is the number of customers in queue i including any in service, $i=1, \ldots, M$. Note that $n_1+n_2+\cdots n_M=N$. There will be $\binom{M+N-1}{N}$ different system states. Let $P(n_1, \ldots, n_M)$ be the probability that the system is in state $(n_1, \ldots, n_M)$. Gordon and Newell[7] showed that

$$P(n_1, \ldots, n_M)=g(n_1, \ldots, n_M)/G \qquad (1)$$

where

$$g(n_1, \ldots, n_M)=\prod_{i=1}^{M} x_i(n_i) \qquad (2)$$

and G is a normalizing constant. Buzen[4] derived an extremely efficient algorithm for the solution of G.

The quantities $x_i(n_i)$ are defined recursively as follows:

$$x_i(0)=1,\ x_i(k)=x_i(k-1)\cdot y_i\cdot 1/U_i(k) \qquad (3)$$

where y_i is the relative frequency of arrivals at queue i and $i=1, \ldots, M$; $k=1, \ldots, N$. The eigenvector equations which must be satisfied are:

$$\sum_{i=1}^{M} y_i\cdot p_{ij}=y_j \qquad j=1, \ldots, M \qquad (4)$$

The set of y_i's is unique to a normalizing constant, and may be regarded as a set of stochastic balance equations.

Let $p_{ij}(r)$ be the probability that a customer of class r

joins queue j after finishing service in queue i. Let $y_i(r)$, $i=1, \ldots, M$ and $r=1, \ldots, R$ be a set of numbers such that

$$y_j(r)=\sum_{i=1}^{M} y_i(r)\cdot p_{ij}(r) \text{ for all } j, r \tag{5}$$

Let the event that there are $n_i(r)$ customers of class r in queue i, $r=1, \ldots, R$ and $i=1, \ldots M$ be represented by the matrix $[n_{ir}]$, whose i, rth element is $n_i(r)$. The matrix $[n_{ir}]$ is feasible if $n_i(r)$ is non-negative and if $\Sigma_{i=1}^{M} n_i(r)=N(r)$, where $N(r)$ is the total number of customers of class r in the network. Of course, $\Sigma_{r=1}^{R} N(r)=N$.

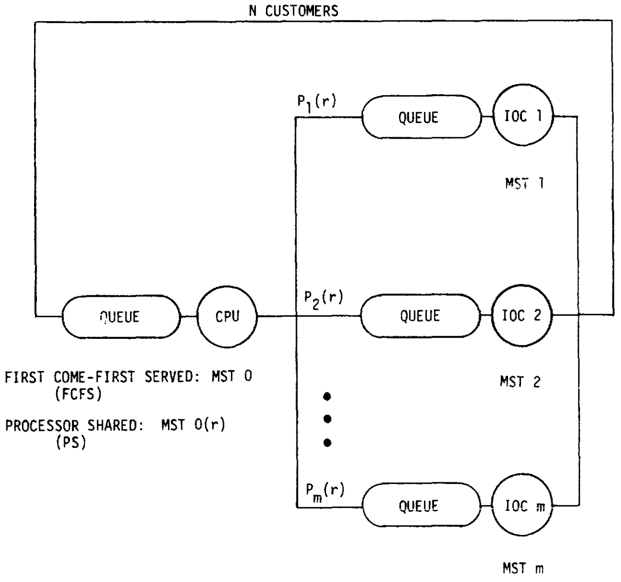
Baskett, Chandy, Muntz and Palacios[2] extended Equations (1) and (2) to the multiclass case and accommodated Processor Shared (PS) CPU service disciplines as well as FCFS. Thus:

$$P([n_{ir}])=\frac{1}{G(R)}\prod_{i=1}^{M} x_i(n_i(1), \ldots, n_i(r)) \tag{6}$$

$$g([n_{ir}])=\prod_{i=1}^{M} x_i(n_i(1), \ldots, n_i(R)) \tag{7}$$

Chandy, Herzog and Woo[6] derived an extremely efficient algorithm for the solution of $G(R)$.

Figure 1 depicts the model, assuming that a fixed integer number N of customers (processes) traverse a closed network consisting of the central processor (CPU) and the I/O channels (IOC). A customer alternately receives service from the CPU and one of the channels. After completing service at the CPU, a customer branches to a channel according to a probability $P_i(r)$ associated with that channel and which may be class dependent. CPU scheduling discipline is either FCFS or PS. Channel scheduling discipline must be FCFS. It turns out to be more convenient when automating the solution of equations (6) and (7) to use Mean Service Time (MST), the reciprocal of the mean service rate.



CPU MST ≡ (TOTAL CPU SERVICE TIME)/(NO. OF CPU PROCESSING PERIODS)

MST m ≡ (TOTAL IOC m SERVICE TIME)/(NO. OF IOC m PROCESSING PERIODS)

Pm (r) ≡ PROBABILITY THAT A JOB OF CLASS r WILL BRANCH TO CHANNEL m AFTER A CPU PROCESSING PERIOD

Figure 1—Queueing network model

Previous papers which reported model validations often provided few details as to how the parameters for the analytical model were measured, so a significant amount of time during the project was devoted to this phase. It was considered highly desirable as a part of this project to develop a measurement methodology which would be within the capabilities of computer staff personnel. Using this methodology and the model of (6), computer staff personnel could validate the model for their system, then carry out studies for tuning the system. The measurement methodology is described in detail in References 10 and 11. A FORTRAN listing for solving the equations of the local balance model in Reference 6 is included in Reference 10.

It is very difficult with current analytical techniques and measurement devices to model and measure the I/O channel behavior of modern computer systems. From the analytical aspect, it is extremely difficult to model the channel operation when the devices are in various modes of operation, such as seek, latency, data transfer, waiting for a control unit and waiting for a channel. Analytical techniques for modeling overlap are not currently available, and this problem will probably require much research and effort in the future.

Modern I/O channel architecture can also cause problems from the measurement point of view. The service time of a disk or drum operation is equal to the seek time (if the device is a disk) plus the time of rotational latency plus the time of data transfer. The channel mean service time (MST) for the model should include all of these components. Although the channel will always be required for the transmittal of data, with modern computer architecture the channel may not be required for latency or seek. It would be extremely desirable to have a measurement probe point which would indicate channel busy time when any device on that channel is busy. Unfortunately, it appears that such a probe point has not been researched and documented for many computers.

It is thus important to realize that current analytical models of computer systems do not explicitly include all of the complexities of the system. Furthermore, existing measurement devices do not permit measurement of the exact parameters of a queueing network analytical model, particularly with regard to I/O channel behavior. The objective of the project reported in this paper was to determine, with the above limitations in mind, whether a procedure could be developed which would enable one to predict with reasonable accuracy the change in computer performance when modifications are made to a particular system. If such a technique were developed and validated, then a manager could use this approach to estimate performance improvements and compare them with the costs for upgrading the system.

To alleviate the current analytical and measurement problems, a "calibration-prediction" technique is proposed as an interim technique to provide useful engineering results until more powerful analytical techniques and measurement tools are available. A review of the equations of the model in Reference 6 shows that proper calculation of CPU throughput is crucial, since all utilizations depend on this quantity. Therefore, as part of the validation procedure a calibration parameter is computed which forces model CPU utilization to agree with the measured CPU utilization. I/O channel utilizations are recomputed using this parameter. Using this calibration procedure, good results were obtained for estimating new computer system utilizations for reallocating files and adding two channels to a 370/155-2, for adding a channel to a 370/155, and for adding a channel to a 370/168.

A possible interpretation of the calibration procedure is that certain simplifying assumptions must be made for mathematical tractability in order to derive the model. Furthermore, it is not feasible with existing monitors to obtain measurements which exactly correspond to model parameters. To reduce the effect of these inaccuracies, it is necessary to first calibrate the model for a given baseline configuration. After the model has been calibrated, it is then feasible to use the model to predict performance of the reconfigured computer system. This approach is similar to the small-signal analysis of electronic circuits. For small values of input signals, the behavior of the circuit is assumed to be linear in the vicinity of an established operating point.

VALIDATION OF THE MODEL ON AN IBM 370/155-2

The IBM 370/155-2 included virtual storage operating system VS2, HASP and the Time Sharing Option (TSO). This particular system configuration consisted of one CPU, 1.5M bytes of main storage and six channels.

With reference to the block diagram of Figure 1, channel 1 was a byte multiplexor channel and provided a data path through a control unit to one IBM 1403 printer and an IBM 2504 card punch. Channel 2 was a block multiplexor channel and provided a path to four IBM 3330 disks via a 3333 Disk Control Unit (DCU). The remaining four channels were selector channels which connected to independent banks of IBM 2314 disks via DCU's. There were eight 2314's connected to channel 3. Channel 4 had eight 2314's, plus an IBM 2848 Display Control and eight 2260 timesharing terminals. Channel 5 was connected to sixteen 2304's. Channel 6 provided a path to five 2420 Magnetic Tape Units (MTU) via a 2803 Tape Control Unit, and to eight 2314's. There was no multiple path switching option in this system.

As shown in Figure 1, the model parameters which must be measured are channel branching probabilities, channel mean service times (MST's), CPU MST and the degree of multiprogramming (N). An IBM hardware monitor was used to record utilizations and as a check on certain quantities provided by an IBM software monitor. The IBM accounting system (SMF) was not suitable for use as the primary software monitor, but did provide useful supplementary information. The software monitor provided by IBM was proprietary, but this situation should not be considered as a serious limitation since the desired measurement capability is included in the supplied software for VS2 Release 2.0. The procedure for deriving the model parameters using the monitors is discussed in detail in References 10 and 11, so only the results will be presented in this paper.

It might be noted at this point that the objective of these experiments was to study system behavior rather than a detailed examination of the operation of a particular subsystem. In particular, these experiments did not require either a separate or an embedded paging model. Hence the paging statistics were included as an integral part of system statistics, e.g., the number of page-in/page-outs were included in the computation of the branching probability for the channel with the paging pack. Of course, this simplification may not be valid in all cases, but good results using this assumption were achieved for these experiments.

In order to establish a known baseline and to insure repeatability of experiments, a benchmark program similar to that described in Buchholz[5] was obtained from the Federal Computer Performance Evaluation and Simulation Center (FEDSIM). This particular benchmark is capable of exercising computers with batch, TSO and virtual storage capabilities. It executes sixty job steps for batch processing during a running time of approximately twenty-five minutes, and provides a mix of CPU bound, I/O bound and balanced workloads. For TSO operation it executes a circular list of commands for such typical timesharing tasks as editing, allocating and linking. All of the experiments described in this paper were conducted during dedicated time with no other users on the system.

For the first experiment the system was configured without the IBM 3330 disks, i.e., channel 2 was not used. To simplify generation of the operating system no timesharing terminals were used, and only the batch portion of the benchmark was needed. (For this workload channels 5 and 6 were not required.)

The objective of this first experiment was to validate the model and the measurement methodology. Since the time-averaged degree of multiprogramming was measured at 3.7 using IBM's Systems Measurement Facility (SMF), model utilizations were solved for N=3 and N=4. The results are shown in Table I.

IBM 2314 disks are used with selector channels and remain connected to the channel for the data transfer and the full rotational latency of the search. The channel is not

TABLE I—Model vs. Measured Utilizations for the Case of IBM 2314 Disks

SERVER	MODEL UTILIZATIONS N=3	MEASURED UTILIZATIONS N=3.7	MODEL UTILIZATIONS N=4
CPU	0.648	0.696	0.733
CH 1	0.544	0.576	0.615
CH 2	0.000	0.000	0.000
CH 3	0.372	0.400	0.421
CH 4	0.394	0.422	0.445

connected to the 2314 during a seek. An interesting point of this experiment is that good results were obtained without including any mean channel seek time in the value of model channel MST's. The channel MST's were computed from channel busy utilizations which took into account data transfer operation and rotational delay only. Calibration of the model was not required.

Since good results were obtained in the first experiment using the 2314 disks, it was decided that a policy would be established for the experiments on the 370/155-2 by adding the 3330 mean rotational delay of 8.35 ms. to the values of channel 2 MST obtained from channel busy times to account for latency. (IBM 3330 disks are not connected to the channel during a search.) For some experiments slightly more delay would have worked better; for other experiments less delay would have improved accuracy. Overall, it did appear to work reasonably well for a uniform policy and calibration was not required for any of the experiments on the IBM 370/155-2.

The model in References 2 and 6 is the first queueing network model which will permit classes of customers. Partitioning a computer system environment into classes of timesharing and batch jobs appears useful, and the results of the validation on the 370/155-2 for this case are shown in Table II. Both IBM 3330 and 2314 disks were used in this experiment and the TSO and batch benchmarks were executed concurrently. Based on the SMF data the measured degree of multiprogramming for TSO was 0.8 and for batch was 3.8.

The model will permit CPU service disciplines of first-come-first-served (FCFS) and processor-shared (PS). Based on the characteristics of IBM machines, one would expect that FCFS would more accurately model system behavior. CDC 6600 machines are more accurately modeled by the PS discipline, as shown in (1). It is interesting to note in Table II that FCFS does yield better results.

For this configuration the system packs were on channel 2 and the temporary data sets were on channels 3 and 4. The allocation routine in VS2 will seek to balance the application programs' (benchmark) I/O accesses between channels 3 and 4. The allocation routine does not take into account the I/O accesses to the timesharing terminals, and the TSO terminals on channel 4 result in it being more heavily utilized.

REALLOCATION OF FILES ON AN IBM 370/155-2

The next experiment used the model to estimate CPU and channel utilizations after a reallocation of files. The purpose of the reallocation was to achieve better balance of channel utilizations, and was similar to that conducted by Hughes and Moe.[8] Assuming a baseline system configuration with utilizations as shown in Table II, new model parameters must now be estimated in order to predict utilizations after a reallocation of files. The procedure for estimating model parameters is reasonably straightforward, and is described in detail in Reference 10. The basic concept is to use one's knowledge of the baseline system's operation, in conjunction with what one would expect for the reconfigured system behavior, and then estimate new branching probabilities and MST's.

The time that an IBM 3330 disk must be connected to a channel during rotational latency (search) can be as short as 250 microseconds. The channel is not required during a seek. Therefore, even though channel 2 had more I/O traffic, its utilization was much less than that of channels 3 and 4 which used 2314's. One method of overriding the VS2 allocation routine was to change the JCL of the benchmark and direct a certain number of the jobs to the 3330's on channel 2. It was decided to direct 9 of the 20 jobs (45%) in the batch benchmark to channel 2, and to use the model to estimate the new channel utilizations. If one was not satisfied with the resulting balance, then the selected value of 45% could be modified.

Table III presents the results of the experiment showing predicted utilizations in column two. Utilizations which were actually measured using the new allocation strategy for the benchmark are shown in columns three and four. The two runs were made using identical system initialization, system configuration, and benchmarks. Thus columns three and four indicate the degree of variation between runs, and show reasonable consistency from one run to the next.

ADDING TWO CHANNELS TO AN IBM 370/155-2

The previous experiment balanced the channel utilizations (within the limits of TSO operation) for channels 2, 3 and 4, and this balanced system was used as the baseline

TABLE II—Model vs. Measured Utilizations for Concurrent TSO and Batch Operation Using Two Classes of Customers

	MODEL	MEASURED	MODEL
	UTILIZATIONS	UTILIZATIONS	UTILIZATIONS
	N(TSO)=1	N(TSO)=0.8	N(TSO)=1
SERVER	N(BATCH)=4	N(BATCH)=3.8	N(BATCH)=4
	CPU:FCFS		CPU:PS
CPU	0.890	0.893	0.865
CH 1	0.433	0.449	0.517
CH 2	0.371	0.370	0.380
CH 3	0.318	0.320	0.355
CH 4	0.476	0.478	0.518

TABLE III—Predicted vs. Measured Utilizations with Two Classes of Customers Concurrent Timesharing and Batch Operations for Reallocation of Files

	MODEL	MEASURED	
	UTILIZATIONS	UTILIZATIONS	
	N(TSO)=1	N(TSO)=0.8	
SERVER	N(BATCH)=4	N(BATCH)=3.8	
CPU	0.903	0.919	0.916
CH 1	0.443	0.476	0.482
CH 2	0.498	0.497	0.502
CH 3	0.212	0.218	0.221
CH 4	0.371	0.391	0.359

for the next experiment. The plan was to first estimate model parameters for the six channel configuration assuming the benchmark workload. Using information provided by hardware and software monitors, channel branching probabilities and MST's can be estimated as described in Reference 10 and predicted utilizations computed using the model.

The IBM 370/155-2 was then reconfigured to include six channels (two additional selector channels with 2314 disks). The benchmark was executed on this system and the actual utilizations measured, with the results shown in Table IV. Such an experiment has not been previously reported, and indicates that the model can yield useful results for estimating performance resulting from an increase in the number of I/O channels.

EXPERIMENTS ON AN IBM 370/155

The IBM 370/155 with OS/MVT and HASP consisted of one CPU, 2.0M bytes of main storage, one byte multiplexor channel and two selector/block multiplexor channels. TSO capability was not included.

Referring to the block diagram of Figure 1, channel 1 was a byte multiplexor and provided a data path through a control unit to two IBM 1403 printers and one 2504 card reader/punch. Channel 2 provided a path to five IBM 3420 Magnetic Tape Units (MTU) via a 3803 Tape Control Unit (TCU), and to two IBM 3330 disks through a 3333 Disk Control Unit (DCU). When connected to the TCU the channel operated as a selector channel; when connected to the DCU it functioned as a block multiplexor channel. Channel 3 was connected to a second DCU with two 3330 disks, and operated as a block multiplexor channel.

The model assumes that a job experiences alternating CPU and I/O processing. During the I/O processing cycle the model assumes that a job is either in a queue or receiving service. This channel service corresponds to data transfer by the channel, rotational latency, and seek time if the device is a disk. However, the analytical model does not explicitly include the overlap behavior when several devices are on a channel. In addition, for IBM channels with IBM 3330 disks the measurement probe point for "channel busy," which is needed to compute channel MST, reflects only the time that data is transferred. There is no probe point to determine the channel busy time when any device is busy, which is more repesentative of model behavior. It was thus extremely difficult to validate the model on the system with IBM 3330 disks. Therefore a calibration procedure was used which forced model and measured CPU utilizations to agree, then recomputed channel utilizations. Table V illustrates the results of the procedure and shows that after CPU utilizations were brought into agreement, reasonable accuracy existed between model and measured channel utilizations.

TABLE IV—Predicted vs. Measured Utilizations with Two Classes of Customers During Concurrent Timesharing and Batch Operation for Adding Two Channels

SERVER	MODEL UTILIZATIONS N(TSO)=1 N(BATCH)=4	MEASURED UTILIZATIONS N(TSO)=0.8 N(BATCH)=3.8
CPU	0.929	0.914
CH 1	0.482	0.484
CH 2	0.522	0.510
CH 3	0.140	0.143
CH 4	0.311	0.315
CH 5	0.074	0.083
CH 6	0.074	0.071

Since the above validation could be regarded as somewhat irregular, it was even more important than before to investigate the utility of the model for estimating a change from the baseline. Using the above two channel case as a baseline, the model was used to estimate utilizations if another block multiplexor channel were added. Predicted utilizations for the three channel case using the calibration-projection method are shown in column two of Table VI. Columns three and four show utilizations which were actually measured. Note that using the calibration-projection technique, extremely accurate estimates were obtained for CPU and channel utilizations for the case of adding another channel.

Since there was no TSO capability in this system, only a single class of customer was used in the queueing model. The reallocation of files experiment is considered less challenging than adding a channel, and was not repeated on the 370/155.

EXPERIMENTS ON AN IBM 370/168-1

The third and final computer system which was available for validation and experimentation was an IBM 370 model 168-1, with virtual storage operating system VS2, HASP and TSO. The system configuration consisted of one CPU, 4.0M bytes of main storage, and eight channels.

Referring to the block diagram in Figure 1, channel 1 was a byte multiplexor channel and provided a path through a control unit to two IBM 1403/3211 printers and a 2504 card punch. Channels 2, 3, 4 and 5 each connected to independent banks of eight IBM 3330 disks via DCU's. During operations with these disks the channels function as block multiplexors. In addition, there were sixteen TSO terminals on channel 3. Channels 6, 7 and 8 were selector channels which connected to independent banks of eight IBM 3420

TABLE V—Model vs. Measured Utilizations Using Calibration Method

SERVER	MODEL UTILIZATIONS N=4 UNCALIBRATED	CALIBRATED	MEASURED UTILIZATIONS N=3.7
CPU	0.677	0.458	0.458
CH 1	0.770	0.520	0.522
CH 2	0.517	0.349	0.349
CH 3	0.000	0.000	0.000

TABLE VI—Predicted vs. Measured Utilizations for Reconfigured System for the Case of Adding a Block Multiplexor Channel

SERVER	MODEL UTILIZATIONS N=4	MEASURED UTILIZATIONS N=3.7	
CPU	0.472	0.475	0.470
CH 1	0.537	0.537	0.532
CH 2	0.214	0.216	0.212
CH 3	0.151	0.151	0.152

MTU's via 3803 Tape Control Units. There was no multiple path switching option in this system.

The objective and procedures for the tests were identical to those previously described. Since TSO was available, two classes of customers were used to validate the model in Reference 6. Due to the presence of 3330 disks the calibration procedure was used again. For the case of increasing the number of block multiplexor channels from three to four, predicted CPU utilization was 0.363; measured CPU utilization was 0.373. Channel utilizations were of comparable accuracy.

SUMMARY

It is very difficult with current analytical techniques and measurement devices to model and measure the I/O channel behavior of modern computer systems. To provide a method for obtaining useful engineering results, a "calibration-prediction" technique is proposed. This technique calibrates a performance parameter to properly reflect the aggregate behavior of the total I/O subsystem (with respect to a given workload). This calibrated performance parameter is then used in the model to predict the effect of modifications to the system.

Using this "calibration-prediction" technique accurate results were obtained for estimating new computer system utilizations for adding a channel to a 370/155, reallocating files and adding two channels to a 370/155-2, and adding a channel to a 370/168. It is believed that this project has provided additional insight into the ranges of applicability of queueing network models of computer systems. In particular, the experiment for adding a channel has not been previously reported, and may be of interest to managers of systems which are currently I/O bound.

ACKNOWLEDGMENT

I am deeply grateful to the IBM Corporation for the use of their measurement devices and for their very capable systems programming support. I also appreciate the many helpful discussions with Prof. J. Buzen of Harvard University, who reviewed an earlier draft of this paper and provided many useful suggestions.

REFERENCES

1. Baskett, F. and F. P. Gomez, "Processor Sharing in a Central Server Queueing Model of Multiprogramming with Applications," *Proc. of the Sixth Annual Princeton Conference on Information Sciences and Systems,* Princeton Univ., March 1972, pp. 598-603.
2. Baskett, F., K. M. Chandy, R. R. Muntz, and F. Palacios-Gomez, "Open, Closed and Mixed Networks of Queues with Different Classes of Customers," *JACM,* 22, 2, April 1975, pp. 248-260.
3. Bhandiwad, R. A. and A. C. Williams, "Queueing Network Models of Computer Systems," *Proc. of the 3rd Texas Conference on Computing Systems,* Austin, Texas, Nov. 1974.
4. Buzen, J. P., "Computational Algorithms for Closed Queueing Networks with Exponential Servers," *CACM,* 16, 9, Sept. 1973, pp. 527-531.
5. Buchholz, W., "A Synthetic Job for Measuring System Performance," *IBM Syst. J.,* 8, 4, April 1969, pp. 309-318.
6. Chandy, K. M., U. Herzog, and L. Woo, "Parametric Analysis of Queueing Network Models," *IBM J. Res. Develop.,* 19, 1, Jan. 1975, pp. 36-42.
7. Gordon, W. J. and G. F. Newell, "Closed Queueing Systems with Exponential Servers," *Oper. Res.,* 15, 2, April 1967, pp. 254-265.
8. Hughes, P. H. and G. Moe, "A Structural Approach to Computer Performance Analysis," *Proc. AFIPS 1973 NCC,* Vol. 42, Thompson Books, Wash., D.C., pp. 109-119.
9. Moore, C. G., *Network Models for Large-Scale Time-Sharing Systems,* Ph.D. Thesis, Univ. of Mich., Ann Arbor, Mich., April 1971.
10. Rose, C. A., *Measurement and Analysis for Computer Performance Evaluation,* Sc.D. Dissertation, George Washington Univ., Wash., D.C., July 1975.
11. Rose, C. A., "Validation of a Queueing Model with Classes of Customers," *Proc. of the International Symposium on Computer Performance Modeling, Measurement and Evaluation,* Harvard Univ., Cambridge, Mass., Mar. 1976.
12. Sekino, A., *Performance Evaluation of Multiprogrammed Time-Shared Computer Systems,* Ph.D. Thesis, MIT Project MAC Report MAC-TR-103, Cambridge, Mass., Sept. 1972.

CPU-utilization and secondary-storage performance—The demand for a new secondary-storage technology

by PETER SCHNEIDER
Siemens AG
Munich, Germany

ABSTRACT

Previous studies investigated the usability of charge-coupled devices (CCDs) in the context of economies to be achieved in main memory capacity. In systems with virtual memories, such economies result from the use of fast paging devices. In spite of the concomitant savings in main memory costs, which will probably be eaten up by the costs of the new-technology paging device, the price/performance ratio must be expected to be less favorable than that of conventional systems, since the share of the operating system software required for page fault handling will increase.

More recent research has shown, however, that not only the degree of multiprogramming, i.e., the number of processes required to cover an I/O time interval, is a factor of crucial importance for optimum CPU utilization, but also the number of disk devices available in the secondary-storage system: unless the number of storage devices is large enough to handle, within an I/O time interval, at least as many parallel I/O operations as are needed to ensure that a sufficient number of processes are again ready for busying the CPU, the aim of full utilization of the CPU cannot be achieved—not even through a higher degree of multiprogramming. With disk devices of ever higher recording densities but otherwise nearly constant performance data becoming available, fewer devices than today will be required in the future to store the on-line data file volume. Since fast, favorably priced central processing units are likewise becoming available, it must be expected that the future systems, unlike the systems of today, will for the first time be beset with the problem of input/output bottlenecks arising from an insufficient number of storage devices. Rather than attempting to achieve the required I/O data rate through a sufficient number of devices operating in parallel, use should therefore be made of such devices as CCD storages in secondary-storage hierarchies, which offer themselves as the less costly solution to the problem.

INTRODUCTION

At the present state of the technology, memory system costs remain the dominant factor in the overall costs of a computer system, with the costs of the secondary storage, in addition to those of the main memory, representing an appreciable share of the total storage system costs. Thus, if a new technology, such as the charge-coupled device or the bubble, is to be implemented in the storage system, it must ensure a more favorable price/performance ratio of the overall system, i.e., either reduce the costs for an unchanged system performance level or substantially enhance the system performance level for a constant or rising cost burden.

In the future the performance aspect will become ever more dominant, which may eventually give rise to a situation where it is not only desirable but indispensable to adopt new technologies in the secondary-storage environment.

APPLICATION OF NEW MEMORY TECHNOLOGIES

Some of the existing computer systems built on the virtual-memory concept have their secondary storages split up into two functionally separated sections (Figure 1): one section, the paging device, is used for storing the programs which were started by the users connected at a given time. At run time, these programs are loaded from the moving-head disks of the other secondary-storage section, the file memory, into the paging device. The tertiary storage, which is also shown, serves as a long-term archival storage medium and will not be further considered in this paper.

The main memory contains the current pages of the currently active processes. This set of pages belonging to a process is called the "working set of pages."[1] The main memory can thus be said to function more or less as a buffer for the paging device: the individual programs will run without interruption only for such a time as their active environment does not change. If a page is missing in main memory (page fault), it has to be fetched from the paging device. As this transfer takes several milliseconds, the requesting process is put in suspense and the CPU turns to another program that can keep it busy. As prior analyses of buffer systems[2,3] have already shown, the page fault rate diminishes with increasing buffer (in this case main memory) capacity, the optimum being reached when the main memory capacity is equal to the paging device capacity: all

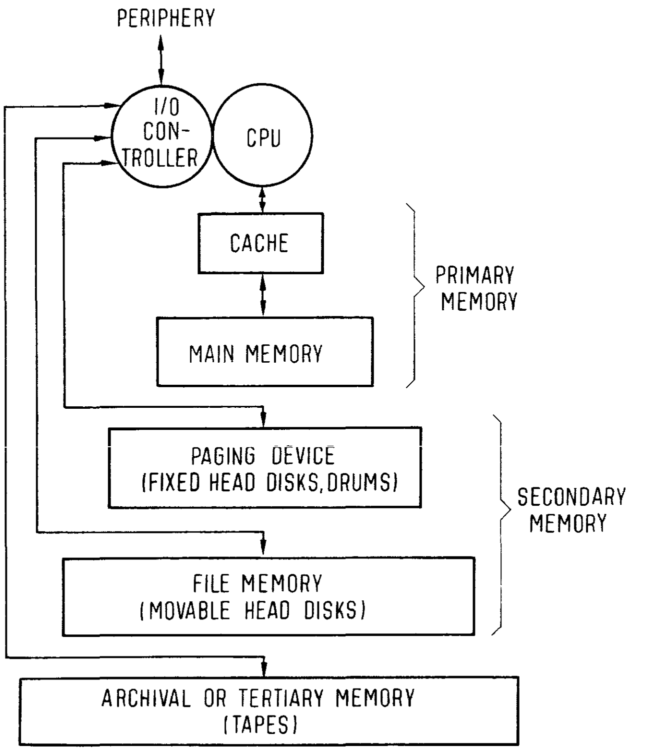

PERIPHERY
I/O CON-TROLLER
CPU
CACHE
MAIN MEMORY
PRIMARY MEMORY
PAGING DEVICE (FIXED HEAD DISKS, DRUMS)
SECONDARY MEMORY
FILE MEMORY (MOVABLE HEAD DISKS)
ARCHIVAL OR TERTIARY MEMORY (TAPES)

Figure 1—Memory hierarchy of a data processing machine

programs will then be main-memory-resident so that the paging rate after the initial phase becomes zero.

In such systems* there will occur, in addition to the mentioned paging I/Os, file I/Os whenever a running program requests file references. In these cases too, the requesting program will be put in suspense and another one ready to busy the CPU processed instead. Moreover, the main memory is not to the same degree used as a buffer for file I/Os as it is for paging I/Os, since the question of a file locality corresponding to the working set of pages for programs has not been studied thoroughly enough so far.

Many of the earlier studies[5,6] on the use of CCD and bubble devices were focused on the possibilities of increasing the speed of page transfer in paging, it being assumed that there would be no bottleneck for file I/Os or that all I/O activities could be handled by paging I/Os. Taking this as a starting point it was demonstrated that, granting an equally high utilization rate of the CPU, the use of a fast paging device, e.g., one in CCD technology, would clearly stand a comparison with a paging device in conventional technology (magnetic drum or fixed-head disk). A shorter access time (sum of latency time and actual transfer time) would enable the hit rate required in the main memory for achieving the same CPU load to be smaller, which means that the main memory capacity could be kept smaller than for systems using conventional paging devices. Thus, the costs of a paging device implemented in the new technology could be allowed to exceed those of a conventional paging device by exactly the amount that corresponds to the costs of the main memory portion saved.

A weakness of this avenue of thinking lies in the fact that it neglects two aspects of existing systems:

1. Such a deliberate increase in the page fault rate will be accompanied by increased operating system overhead for storage management. This implies that with the CPU loaded to capacity (and this is indeed ensured by the fast page transfer) there will be less time available for processing user programs. From the user's point of view the price/performance ratio might therefore appear degraded in comparison with that offered by the known and tried technologies.
2. The limited channel data rates will not at all accommodate page transfer times as short as those realized by, say, a CCD paging device.

Bearing these two limitations in mind and considering further that the existing systems exhibit a more or less well-balanced performance level of all system constituents, which makes for a good CPU utilization (over 90%) it is unlikely for new-technology memories to be integrated in existing systems until the cost picture becomes more favorable.

It is therefore a stringent conclusion that an improved price/performance ratio for existing systems can only be attained if the costs of the new paging device technology are absolutely lower than those of the conventional technology.

The aim of this paper should be seen in an attempt to demonstrate that development trends already recognizable in the CPU and the moving-head disk fields indicate that in future the secondary storage—but now for reasons of performance—will present a sure area of application for the CCD memory concept. For this reason, the first question discussed will be that of the variables exerting an influence on CPU utilization.

CPU UTILIZATION AND INFLUENCING VARIABLES

This investigation was performed with the aid of a simulation model (cf. Figure 2) of the system whose schematic is shown in Figure 1.

Waiting in front of the CPU is a queue of processes which are successively served by the CPU. The maximum length of this queue is a function of the degree of multiprogramming realized. In the model, too, the secondary storage is split up into a paging device and a file memory. When a paging I/O or file I/O request initiates a process change at the CPU, the relevant request is queued at the device addressed. Upon completion of the I/O operation, the associated process is again lined up in the CPU queue. The number of processes in the systems is constant; it corresponds to the degree of multiprogramming realized.

The model is subject to the following constraints, which

* There are also virtual systems where all I/O are handled by paging.[4]

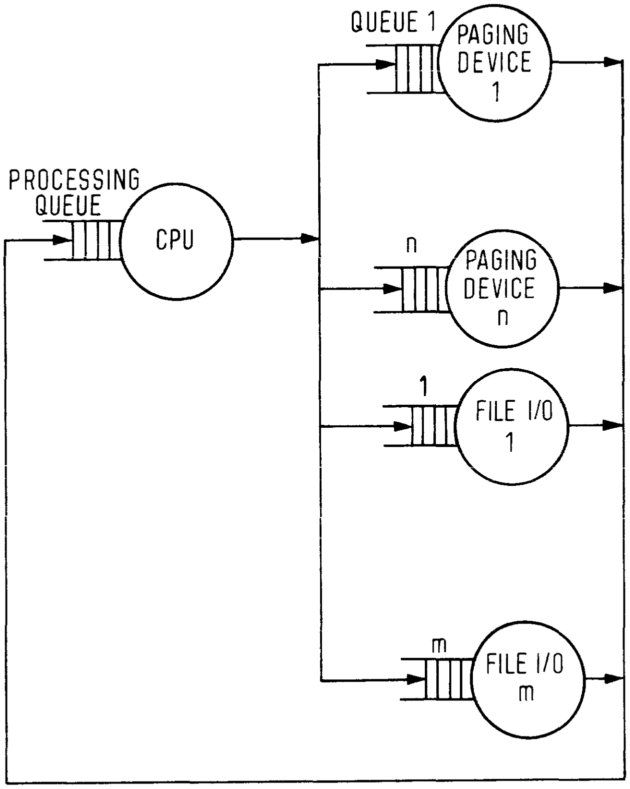


Figure 2—Simulation model used for the performance evaluation of data processing systems

represent better conditions than are encountered in actual systems:

(a) Each of the connected devices has its own independent interface to the main memory. Collisions will therefore occur only in cases where two or more requests compete for the same device.

(b) File I/O requests and paging I/O requests are evenly distributed among all file memory devices and paging devices respectively. In reality, it may happen that, depending on the file allocation, certain devices are accessed much more frequently than others.

(c) File I/O and paging I/O requests have an equal share of 50% in all causes of process change. Other causes, such as time slice runout or terminal I/O, are neglected. Real systems are more likely to exhibit a predominance of file I/O operations.

In order to establish realistic paging I/O and file I/O rates within the simulation system, the distribution of the process run times between process changes was determined at a time-sharing service computer center during open-sessions on different days and at different times. The measurement revealed a mean distribution of process run time lying at about 3000 instructions; Figure 3. This distribution was adopted for the simulation model.

The model itself was built with the aid of the SIAS (SIEMENS ABLAUF VERFOLGER) simulation language, which is equivalent to IBM's GPSS language.

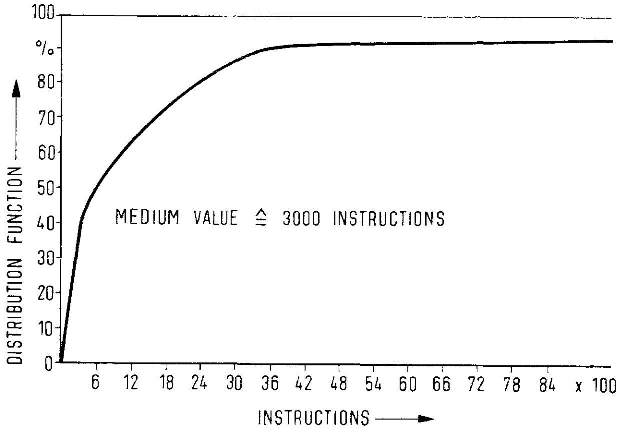


Figure 3—Distribution function of program processing time between I/O-operations (paging I/O and file I/O)

The investigation was carried out on systems with differing numbers of disks, with and without an explicit paging device, and with differing CPU performance levels. The following sections summarize some results obtained for systems with unlimited numbers of devices, with 4 and 10 devices, and with varied degrees of multiprogramming.

Except for the results shown in Figure 7, where the duration of I/O operations $D_{I/O}$ (made up of the latency time and the transfer time) was varied, all results given here are based on an assumed 34 ms for the duration of I/O operations.

SYSTEMS WITHOUT EXPLICIT PAGING DEVICE

For the sake of a clear and straightforward discussion, we shall first deal with the results of the simulation of models without an explicit paging device.

Systems with unlimited numbers of devices

If the number of devices is not limited, an input/output bottleneck cannot arise in a computer system. The utilization level of the CPU is, in this case, determined solely by the degree of multiprogramming; Figure 4. The necessary degree of multiprogramming M should be equal to the quotient

$$\alpha = \frac{D_{I/O}+L}{L},$$

with $D_{I/O}$ representing the duration of an I/O operation and L the mean run time of processes between I/O operations. At the degree of multiprogramming ($M=\alpha$) denoted by this quotient, the CPU reaches a utilization level of 100%: any further increase of M is harmful and will lead to the well-known "thrashing" effect. Looking at the CPU utilization level from this angle, a secondary-storage configuration can be regarded as being sufficiently large the instant the number of devices becomes greater than the theoretical degree of multiprogramming; this is so because at this

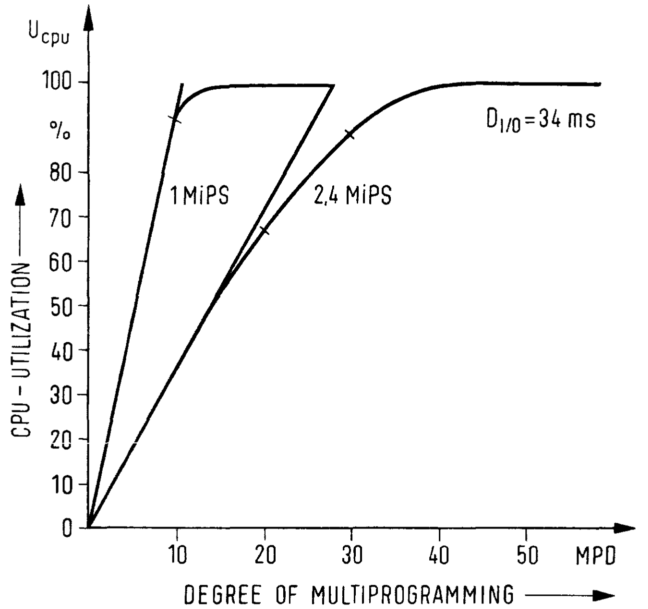


Figure 4—CPU utilization as a function of the degree of multiprogramming without limitation of the number of devices

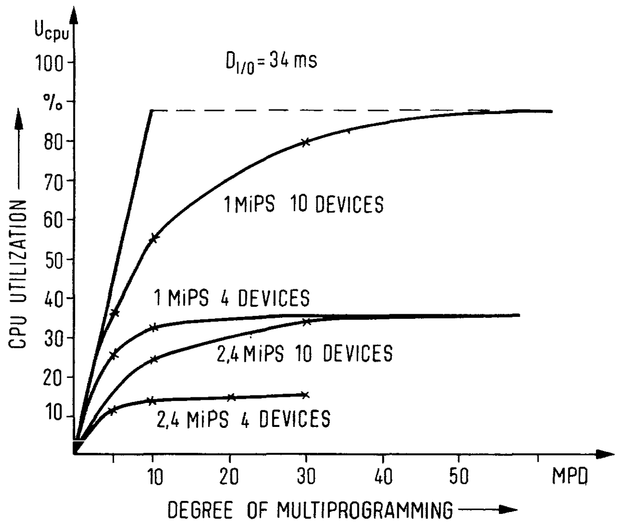


Figure 5—CPU utilization as a function of the degree of multiprogramming with a limited number of devices

instant there exists a balanced ratio between I/O request and I/O terminations, which prevents the CPU from idling.

It can further be seen from Figure 4 that a constant distribution of process run time but differing CPU performance levels will produce pronounced differences in the necessary degree of multiprogramming—a fact that is immediately evident from the above analysis.

The study of this simulation model also revealed that systems with a high CPU performance level and a CPU-bound load are equivalent to systems with a correspondingly lower CPU performance level but I/O-bound load. This, too, is immediately evident: as far as the CPU utilization level is concerned, it does not make any difference whether the program run time between I/O operation is taken up by a great number of fast-executing instructions (fast CPU, CPU-bound programs) or by only a few slow-executing instructions (slow CPU, I/O-bound programs). This holds true also for all further cases considered.

Systems with a limited number of devices

To show the strong influence exerted by the number of devices on the CPU utilization level, the graph in Figure 5 plots secondary-storage configurations with 4 and 10 disks and a mean block transfer time of 34 ms. Again, a distinguishing criterion is the different CPU performance level, which, as mentioned above, can also be interpreted as more or less pronouncedly CPU-bound load.

What becomes apparent here is an effect similar to that observed when the degree of multiprogramming is too low: if the number of devices is too small, a full-capacity utilization of the CPU is impossible. Even very high degrees of multiprogramming are of no avail then, since, after the initial phase, I/O request from all processes will be competitively bidding for access to the secondary storage and be queuing up to be served. During an I/O time interval, it is possible to handle, at best, as many I/O request as there are devices. To ensure a high CPU load, the minimum number of devices to be provided** must be equal to

$$G=\frac{D_{I/O}}{L}=\propto-1.$$

It can here be stated as a general rule that the crucial determinant for the CPU utilization level is the minimum of the degree of multiprogramming and the number of devices. The following greatly simplified model was established:

- The process run time L was considered constant.
- Every I/O request bids for access to the next device capable of servicing the request.
- These prerequisites granted, the idle time of a processor is given by:

  $T_{IDL}=D_{IO}-\min(M-1,\ G)\cdot L$ for $M\leq\propto$ and $G\leq\propto-1$,

  and the CPU utilization deduced therefrom, by:

  $U_{CPU}=\min(1,\ M/\propto,\ (G/a-1).$

A utilization curve calculated from these formulas is plotted in Figure 5 for the case of 10 devices being connected to a 1 MIPS processor.

The curve obtained through simulation approaches this theoretical curve, reaching the theoretical limit value only at a degree of multiprogramming of 60. This discrepancy

** If use is made of an additional paging device, paging I/O time ($D_{I/O}^{PF}$) and file I/O time ($D_{I/O}^{FIO}$) will differ. The number of devices is then given by

$$G=\frac{1}{L}\cdot(h_{PF}\cdot D_{I/O}^{PF}+h_{I/O}\cdot D_{I/O}^{FIO})$$

where h_{PF} and $h_{I/O}$ represent, respectively, the paging and the file I/O rates.

can be accounted for as follows:

(a) The assumption that an I/O request will invariably bid for access to the device next capable of servicing a request is a far cry from reality. On the contrary: even when assuming evenly distributed bids for access to all secondary-storage devices, there will be short-term rushes for individual devices. This exactly is the reason why it takes a higher than the theoretically calculated degree of multiprogramming to achieve full CPU utilization: a high degree of multiprogramming makes it easier to busy all existing devices.
(b) The distribution of run time may lead to a premature depletion of the processor queue, namely before a CPU-busying process is made available again by the termination of an I/O operation.

Up to this point, the discussion has been restricted to systems in which no separate paging device whatsoever was used. All I/O requests, both paging and file, were handled by the same type of device (e.g., disks). In the following, the influence of a separate paging device on system performance will therefore be briefly analyzed.

SYSTEMS WITH SEPARATE PAGING DEVICES

The table given hereunder indicates the improvement in CPU utilization which, for a given limitation of the number of devices, is attainable if a separate paging device is used. It has been assumed that with an explicit paging device employed the transfer of one page will take 1 ms, including the latency time. Assuming a mean running length of 3000 instructions and 2.4 MIPS, this is approximately equal to one processing interval of a process between two I/O operations. Viewed from the angle of the processor, page faults are not time-critical, since the mean program run time and the duration of the page fault are of the same order of magnitude.

As the simulation model assumes that 50 percent of all I/O requests are paging I/O requests, which for the processor are no longer time-critical, the use of an explicit paging device can be expected to improve the CPU utilization level by a factor of 2. This is borne out by the results.

An interesting result is that as long as there is no explicit paging device available both paging and file I/O requests should be honored from all disks. Reserving disk areas on some few devices will lead to an overload on these and an underload on all others, with immediate consequences for the CPU utilization level.

TABLE I—CPU utilization for different numbers of disk devices with and without explicit paging device.

CPU performance MIPS	CPU utilization in %	Number of devices			Degrees of multiprogramming
		Disks	thereof for paging	Paging device	
2.4	15.1	4	4	—	30
2.4	30.6	4	—	1	30
2.4	22.8	10	4	—	30
2.4	34.2	10	10	—	30
2.4	69.6	10	—	1	30

With 10 disk devices connected and all of them used for paging and file I/O, a 2.4 MIPS processor will be utilized to a level of 34.2 percent. If, in contrast to this disk areas for paging are reserved on only 4 disks, the processor utilization level will be only 22.8 percent.

SUMMARY OF SIMULATION RESULTS

The investigations have revealed that the degree of multiprogramming and the number of devices are factors of similar weight as regards the CPU utilization level, but that the number of devices represents the more crucial bottleneck—in other words: if the secondary-storage system is underrated in terms of the number of devices employed in the secondary-storage system rather than in terms of the storage capacity, not even a high degree of multiprogramming will be able to raise the CPU utilization to a satisfactory level. It has further been demonstrated that, on the basis of the measured program run-time distribution values, powerful systems of, say, 2.4 MIPS without any device bottlenecks would require a degree of multiprogramming whose attainability appears doubtful.

This brings into view the first one of the set of causes which in the future will make it necessary to use faster secondary-storage technologies than today:

Always assuming a run-time distribution as shown in Figure 3, the ratio between mean program run time and I/O processing time will continuously deteriorate, because ever faster processors are becoming available at ever more attractive prices, while the performance data of the secondary-storage devices will remain comparatively constant.

The only factor in the domain of secondary storages that undergoes pronounced changes is the storage density of moving-head disk storages.

The ever-increasing storage density of the moving-head disk storage enables the data files of computer centers with fast systems to be accommodated on ever- fewer disk drives. Thus, with the costs of the mechanical section of the disk storage remaining virtually constant, the increased recording density will bring substantial economies in secondary-storage costs. Let us assume that a given set of data files comprises a data volume of 10.000 MB: if use is made of 100-MB disk devices, this data volume requires 100 devices for storing. Using a 500-MB disk storage, however, a mere 20 devices will be needed. It goes without saying that from the point of view of the computer center the latter configuration is the more cost-effective one.

This gives rise to a situation where even the use of a paging device with short page transfer times may no longer be sufficient to ensure full CPU utilization. Specifically, this applies to cases where the number of secondary-storage devices is no longer sufficient to accommodate all file I/Os. Full utilization of the CPU can then only be achieved through an apparent increase in the speed of the secondary storage. This means, however, that provision has to be made for a separate buffer (in CCD technology for instance) for the secondary-storage devices.

SECONDARY-STORAGE HIERARCHY

Figure 6 shows the schematic diagram of a data processing system with a secondary-storage hierarchy. In contradistinction to many existing systems, the secondary storage should be connected to the main memory through a special storage processor SCU rather than through an I/O processor and the central processor (cf. Figure 1).[7] In the case of high-performance processors, this will clearly relieve the load on the CPU/main memory interface. As a consequence, CPU references to the cache will no longer be obstructed by I/O activities.

The function of the CCD storage within the secondary-storage hierarchy is analogous to that of the cache within the main memory hierarchy. Granting sufficient hit rates, the mean duration of I/O operations will be reduced to nearly that of page transfers. The task falling to the secondary-storage processor is to manage the secondary-storage hierarchy. As soon as the secondary-storage processor finds an I/O request directed to it, it ascertains whether the requested page is located in the CCD page buffer, in which case it initiates page transfer between main memory and paging buffer.

In the case of a miss, the storage processor assumes responsibility for the transfer of the requested page not only between the disk devices and the CCD buffer but also to the main memory. If the organization of the paging buffer permits, the data units (blocks) transferred between disk and CCD buffer may be larger. Owing to the fact that the CCD buffer capacity is larger than that of the main memory, there is, in the case of an I/O request, some probability for pages already used previously to be found again in the CCD buffer (backward hit). Depending on the block size within the CCD buffer, it is also possible for hits in the forward environment of a page request (forward hits) to occur.

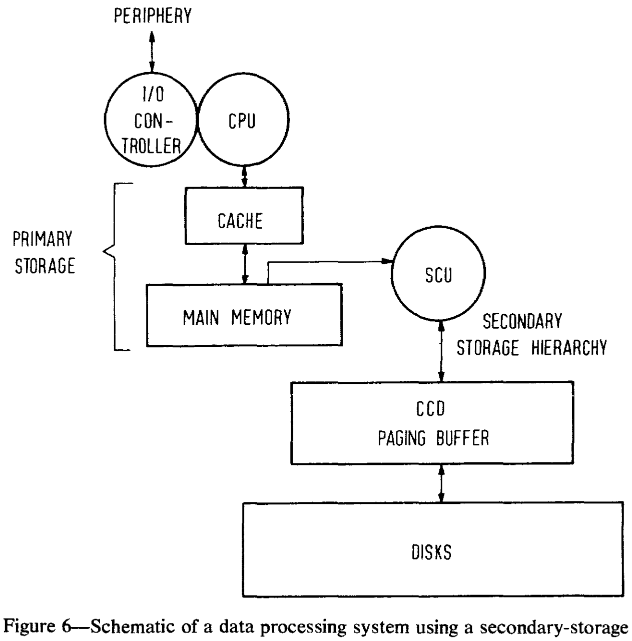


Figure 6—Schematic of a data processing system using a secondary-storage hierarchy

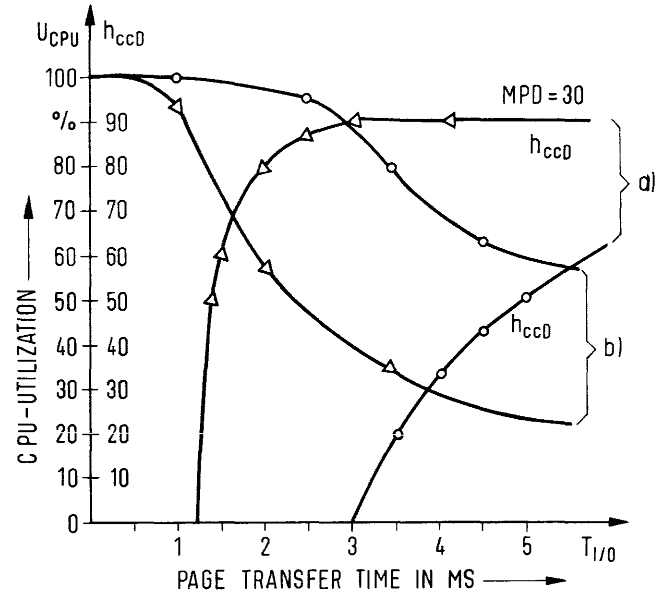


Figure 7—(a) CPU utilization as a function of the mean page transfer time and (b) corresponding hit ratio necessary for full CPU utilization

Figure 7a plots the utilization of a 2.4 MIPS and 1 MIPS processor vs. the page transfer time. It should be mentioned here that, with a sufficiently high degree of multiprogramming provided, the transfer times plotted can, on an average, be exactly attained by parallel operation of slow devices or the use of a fast device of adequate capacity.

We shall now deduce an algorithm for calculating the hit rates in the CCD paging buffer required for full utilization of the central processor.

The effective access time of storage hierarchies is given by the well-known formula[7]

$$T_{eff}=h_1 \cdot t_1+(1-h_1)\cdot t_2$$

where h_1 represents the hit rates in the buffer stage, t_1 the access time of the buffer stage, and t_2 the access time to the second stage.

For purposes of this analysis, these quantities are interpreted as follows:

T_{eff} = $\bar{T}_{I/O}$ corresponds to the mean period of time required for handling an I/O request.†

t_1 = t_{CCD} is the access time of the buffer stage,

t_2 = $T_{I/O} = \frac{D_{I/O}}{G}$ corresponds to the mean serving time attained through the parallel utilization of G devices with access time $D_{I/O}$.

h_1 = h_{CCD} represents the buffer stage hit rate to be found.

† Two means are calculated for this purpose: the first one for the number of devices, the second one for the hierarchical stages of the memory system.

Full utilization of the CPU is ensured if one I/O is served per run time L. It follows that $\bar{T}_{I/O}$ i.e., the average time required for handling an I/O request, must be equal to the mean program run time between I/O interrupts.

Thus, the hit rate required for full utilization is given by:

$$h_{CCD} = \frac{G \cdot L - D_{I/O}}{G \cdot t_{CCD} - D_{I/O}} \text{ (1) or, after transformation, by}$$

$$h_{CCD} = \frac{L - T_{I/O}}{t_{CCD} - T_{I/O}} \text{ (2).}$$

It can be recognized from this formula that if the number of devices G is sufficiently large the required hit rate becomes zero and a buffer storage is not necessary.

Plotted on the abscissa in Figure 7 are the $T_{I/O}$ values. Assuming, for example, that $t_{CCD}=1$ ms, the various $T_{I/O}$ values are associated with corresponding hit rates required for full CPU utilization. These are plotted in Figure 7b for a 1 MIPS and a 2.4 MIPS system. As can be seen here quite clearly, it is necessary, as a function of time $T_{I/O}$ and, hence, as a function of the number of devices G, to have relatively high hit rates in the buffer. If use were made of paging devices with transfer times shorter than those assumed here, there would be no need to make such high demands on the hit rate in the buffer.

CONCLUSION

In view of the ever increasing recording density in disks and the enhanced CPU performance levels, the emergence of a bottleneck in the secondary-storage environment must be anticipated. It would appear that the adoption of a secondary-storage hierarchy, managed by a storage processor, represents a more cost-effective approach than the provision of the devices required for parallel operation. For a constant on-line data file volume, an increase in the number of devices, i.e., a distribution of the files among such a number of devices as are necessary to attain the required I/O handling time, would considerably boost the costs and yet fail to lead to a full utilization of the devices.

Even in systems with a large on-line data file volume, the cost situation will give such a storage hierarchy, complemented by a cassette storage, a competitive edge on a system with a large number of disks.

The performance data of the CCD technology, to be gathered from the report by Bhandaker[5], will in any case accommodate the page transfer times which may possibly be required in the system. For this reason, the design goal for CCD modules should be to achieve maximum storage density with its attendant cost advantages, because even in the case of relatively long access times sufficiently high data rates are attainable through a favorable storage organization.

REFERENCES

1. Denning, P., "The Working Set Model for Program Behaviour," *Commun. of the ACM,* 11, 1968, pp. 323-333.
2. Mattson, R. L., "Evaluation of multilevel Memories," *IEEE Trans. Magn.,* Vol. MAG-7, Dec. 1971, pp. 814-819.
3. Meade, R. M., "On memory system design," in *AFIPS Conf. Proc.* (FJCC), Vol. 37, Nov. 1970, p. 33.
4. Boyse, I. W. and D. R. Warn, "A Straightforward Model for Computer Performance Prediction," *Comput. Surveys,* Vol. 7, No. 2, June 1975.
5. Bhandaker, D. P., "Cost Performance Aspects of CCD Fast Auxiliary Memory," *Proc. CCD '75',* Charge-Coupled Devices Appl. Conf., 1975, San Diego, pp. 435-442.
6. Pohm, A. V., "Cost/Performance Perspectives of Paging with Electronic and Electromechanical Backing Stores," *Proc. of the IEEE,* Vol. 63, No. 8, Aug. 1975.
7. Schneider, P., "Working Set Restoration—A method to increase the performance of multilevel storage hierarchies," in *AFIPS Conf. Proc., NCC* '76', pp. 373-380.

Nonlinear parameter estimation for probabilistic finite-state automata*

by FRED J. MARYANSKI and KUANG CHAN WU**
Kansas State University
Manhattan, Kansas

ABSTRACT

In this report, a nonlinear parameter estimation technique is employed to compute state transition probabilities of probabilistic finite-state automata. The technique is applied to the modelling of the DNA meiosis process. Using the nonlinear estimation method, transition probabilities are estimated which project an external behavior for the model that is very close to experimental data. The estimation method depends strongly upon the initial values supplied. Since it is not possible to iterate to an optimal set of initial values, the selection of initial values must be an intuitive process. However, a very close fit can be obtained in a small number of executions. The nonlinear estimation method is applicable to the estimation of transition probabilities of probabilistic finite-state automaton models for any stochastic system.

INTRODUCTION

In many situations it is possible to observe the external behavior of physical systems and then use these observations to hypothesize the internal structure of the system. The particular system considered here is DNA (Deoxy-Nucleic Acid) meiosis. The internal structure of the meiosis process is modelled by a probabilistic finite-state automation.

Definition 1

A probabilistic finite-state automaton is a 5-tuple.

M=(S, X, P, I, F) where
- S is a finite state set;
- X is a finite input alphabet;
- P is a mapping from $S x X x S \rightarrow [0, 1]$ such that $p(S_i, x, S_j)$ is the probability of M undergoing a transition from state S_i to state S_j upon receiving input x;
- I is the initial state stochastic row vector such that i_k is the probability of state S_k being the initial state; and F is a binary column vector such that f_k is 1 if state S_k is a final state.

The mapping P can be represented by a set of $|S| x |S|$ stochastic matrices, $P(x)=\|p_{i,j}(x)\|$ where $p_{i,j}(x)=p(S_i, x, S_j)$.

The details of the internal structure of the meiosis process are described by the transition probabilities of the finite automaton model. The transition probabilities are obtained from estimates made based upon the external observations. In the particular study described here nonlinear programming methods[1] were applied to compute a set of estimators.

In this paper, the Aviemore model of DNA meiosis[2] is used as a basis for developing a probabilistic finite-state automaton which describes that process. Transition probabilities of the automaton model are estimated from data provided in Reference 2. The effects of normalization in estimation process and simplification of the model are investigated in terms of their effects on goodness of fit and the number of calculations required to determine an acceptable estimate.

DNA MEIOSIS MODEL

The Aviemore model is shown in Figure 1. A probabilistic finite-state automaton, M, can be defined which represents that Aviemore model.

$$M=(S, X, P, I, F)$$

$$\text{where} \quad S=\{I, II, III, \ldots, IX\},$$

$$X=\emptyset,$$

$$P = \begin{bmatrix} Z_1 & P_1 & P_1 & P_1 & P_1 & P_2 & P_2 & 0 & 0 \\ C_3 & Z_2 & 0 & 0 & 0 & 0 & 0 & C_4 & 0 \\ C_2 & 0 & Z_3 & 0 & 0 & 0 & 0 & 0 & C_1 \\ C_1 & 0 & 0 & Z_4 & 0 & 0 & 0 & C_2 & 0 \\ C_4 & 0 & 0 & 0 & Z_5 & 0 & 0 & 0 & C_3 \\ 0 & C_2 & C_3 & 0 & 0 & Z_6 & 0 & 0 & 0 \\ 0 & 0 & 0 & C_4 & C_1 & 0 & Z_7 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & Z_8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & Z_9 \end{bmatrix}$$

* The work reported herein was partially supported by the Bureau of General Research, Kansas State University.

** Author's current address is Bendix Corporation, Columbia, Md.

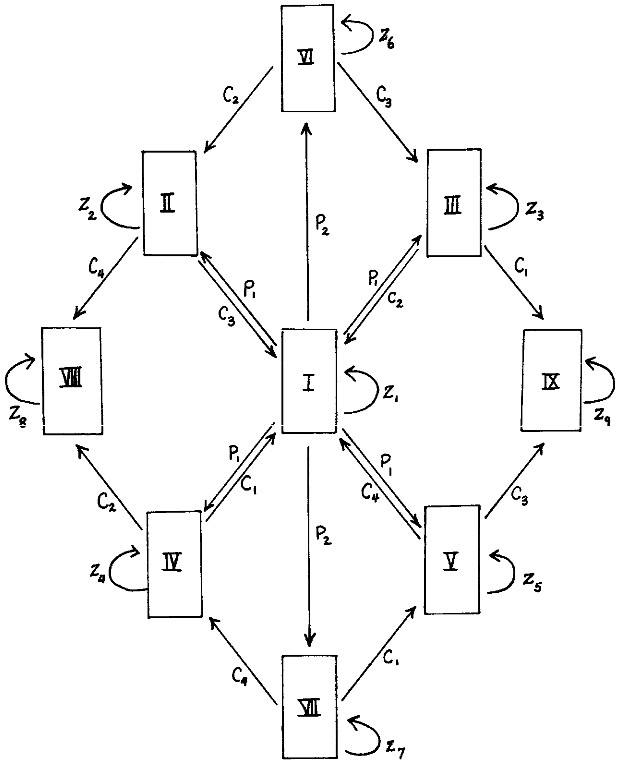


Figure 1—Aviemore model

represents the next state mapping (since $X=\emptyset$ we only have one matrix for P),

$$I=[1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0],$$

and $F=1$.

Since M is a probabilistic finite-state automaton, P must be a stochastic matrix, that is, all the rows must sum to 1. Another property of M is that all entries in P be non-negative. Based upon these facts, the following equations can be derived.

$$Z_1+4P_1+2P_2=1 \text{ or } Z_1=1-4P_1-2P_2 \quad (1)$$

$$Z_2+C_3+C_4=1 \text{ or } Z_2=1-C_3-C_4 \quad (2)$$

$$Z_3+C_1+C_2=1 \text{ or } Z_3=1-C_1-C_2 \quad (3)$$

$$Z_4+C_1+C_2=1 \text{ or } Z_4=1-C_1-C_2 \quad (4)$$

$$Z_5+C_4+C_3=1 \text{ or } Z_5=1-C_3-C_4 \quad (5)$$

$$Z_6+C_2+C_3=1 \text{ or } Z_6=1-C_2-C_3 \quad (6)$$

$$Z_7+C_1+C_4=1 \text{ or } Z_7=1-C_1-C_4 \quad (7)$$

$$Z_8=1 \quad (8)$$

$$Z_9=1 \quad (9)$$

and

$$0\leq P_1, P_2, C_1, C_2, C_3, C_4, Z_1, Z_2, \ldots, Z_9\leq 1. \quad (10)$$

In order to derive a set of equations indicating the probabilities of reaching each final state, we must allow for a potentially infinite number of traversals of the loops in the model. For example, in order to reach state II from state I, there may be some large number of transitions between states I and III, and a large number between I and V (or any other states), before finally terminating at state II. The transition looping phenomenon is described by a variable T which represents the probability of starting at state I and returning to I after having traversed all possible loops from zero to an infinite number of times.

$$T=\left(\sum_{k=0}^{\infty}(P_2C_2C_3)^k\right)^2*\left(\sum_{i=0}^{\infty}(P_2C_1C_4)^i\right)^2*$$
$$\sum_{m=0}^{\infty}(P_1C_1)^m*\sum_{j=0}^{\infty}(P_1C_2)^j*\sum_{n=0}^{\infty}(P_1C_3)^n*\sum_{d=0}^{\infty}(P_1C_4)^d.$$

Since for $0\leq X<1$,

$$X^0+X^1+X^2+\cdots+X^{\infty}\cong\frac{1}{1-X},$$

$$T=\frac{1}{(1-P_2C_2C_3)^2}*\frac{1}{(1-P_2C_1C_4)^2}*\frac{1}{(1-P_1C_1)}*\frac{1}{(1-P_1C_2)}*\frac{1}{(1-P_1C_3)}*\frac{1}{1-(P_1C_4)}. \quad (11)$$

The termination of a derivation at state K is indicated by the transition from K to K with probability Z_K. In our model of the meiosis process, a Z_K transition is assumed to occur exactly once for each derivation.

A derivation in M starts at state I, makes some number of traversals about the loops in the model (as indicated by T), proceeds from state I to its final state by a direct path, and then terminates with a Z_K transition. This behavior is described by the following equations for terminating in each of the 9 states.

$$P(I)=Z_1*T \quad (12)$$

$$P(II)=Z_2*T*(P_1+P_2*C_2) \quad (13)$$

$$P(III)=Z_3*T*(P_1+P_2*C_3) \quad (14)$$

$$P(IV)=Z_4*T*(P_1+P_2*C_4) \quad (15)$$

$$P(V)=Z_5*T*(P_1+P_2*C_1) \quad (16)$$

$$P(VI)=Z_6*T*P_2 \quad (17)$$

$$P(VII)=Z_7*T*P_2 \quad (18)$$

$$P(VIII)=Z_8*T*(P_1C_2+P_1C_4+2P_2C_2C_4) \quad (19)$$

$$P(IX)=Z_9*T*(P_1C_1+P_1C_3+2P_2C_1C_3) \quad (20)$$

EXPERIMENTAL DATA

Although there are 9 final states in the model, only four classes can be distinguished experimentally. The classes and the states contained within them are

$$A=\{I\}$$

$$B=\{II, III, IV, V\}$$

$$C=\{VI, VII\}$$

$$D=\{VIII, IX\}.$$

Reference 2 provides one set of observed probabilities for each class. The observed probabilities are:

$$P(A)=P(I)=0.9466 \tag{21}$$

$$P(B)=P(II)+P(III)+P(IV)+P(V)=0.0079 \tag{22}$$

$$P(C)=P(VI)+P(VII)=0.0001 \tag{23}$$

$$P(D)=P(VIII)+P(IX)=0.0454 \tag{24}$$

In order to express the class probabilities, the final state equations (eq. (12-20)) are substituted into eq. (21-24) yielding

$$P(A)=T*Z_1=0.9466 \tag{25}$$

$$P(B)=T*(Z_2*(P_1+P_2C_2)+Z_3(P_1+P_2C_3) \tag{26}$$
$$+Z_4(P_1+P_2C_4)+Z_5(P_1+P_2C_1))=0.0079$$

$$P(C)=T*P_2(Z_6+Z_7)=0.0001 \tag{27}$$

$$P(D)=T*(P_1C_2+P_1C_4+2P_2C_2C_4+P_1C_1 \tag{28}$$
$$+P_1C_3+2P_2C_1C_3)=0.0454$$

ESTIMATION OF TRANSITION PROBABILITIES

There are six unknown variables, P_1, P_2, C_1, C_2, C_3, C_4 but only four nonlinear equations, eq. (25-28). Therefore an infinity of solutions may exist. Since the equations cannot be solved directly, the values of the transition probabilities must be estimated as closely as possible from the given data.

Due to the small amount of experimental data neither the maximum likelihood nor least squares methods could be applied in this situation. However, it was determined that nonlinear programming methods could be applied. The particular method employed is the nonlinear parameter estimation program of Bard.[1] In order to use the nonlinear programming method, eq. (25-28) must be stated in the form of objective functions as:

$$Y_1=P(A)-0.9466 \tag{29}$$

$$Y_2=P(B)-0.0079 \tag{30}$$

$$Y_3=P(C)-0.0001 \tag{31}$$

$$Y_4=P(D)-0.0454. \tag{32}$$

The program will attempt to minimize the difference between the observed and estimated transition probabilities by maximizing the following function:

$$G=(Y_1{}^2+Y_2{}^2+Y_3{}^2+Y_4{}^2)*(-1). \tag{33}$$

Eq. (1-10) serve as the constraint functions for the estimation procedure which requires a set of initial guesses of the six transition probabilities. The initial guesses are iteratively modified until G (eq. (33)) is as close as possible to 0.

A large number of runs were made with varying sets of initial guesses. The results of these runs are tabulated in Reference 3. The best results, that is, maximum G value, were obtained with an initial guess of

$$P_1=0.1,\ P_2=0.025,\ C_1=0.05,\ C_2=0.05,\ C_3=0.05,\ C_4=0.05.$$

The parameter estimates obtained from the estimation program were

$$P_1'=0.02342$$

$$P_2'=0.1506E(-6)$$

$$C_1'=0.4389$$

$$C_2'=0.4810$$

$$C_3'=0.4810$$

$$C_4'=0.4389$$

The value of G is $-.1939E(-6)$. The execution time for this run was 2.17 seconds on an IBM 370/158. When the estimated parameters are inserted into eq. (25-28), the probabilities for the four observed classes shown below, result:

$$P'(A)=0.9464$$

$$P'(B)=0.007837$$

$$P'(C)=0.02519E(-7)$$

$$P'(D)=0.04501$$

The closeness of our estimated class probabilities to the experimentally provided values can be measured by the sum of weighted squared error.

$$ERR'=\frac{(P'(A)-P(A))^2}{P(A)}+\frac{(P'(B)-P(B))^2}{P(B)} \tag{34}$$
$$+\frac{(P'(C)-P(C))^2}{P(C)}+\frac{(P'(D)-P(D))}{P(D)}.$$

For these estimates ERR=0.1411E(−4).

NORMALIZATION

The results presented in the previous section show that the class probabilities can be estimated with a close error tolerance. However, the estimated value of P′(C) differs very significantly from the observed value of P(C) in eq. (23). Since the magnitude of P(C) is so small it has little effect upon the computation of the objective function. The estimation program maximized the objective function based upon the larger terms in eq. (33).

This problem can be alleviated by normalizing the objective functions in eq. (29-32). Normalization entails dividing Y_1, Y_2, Y_3, and Y_4 by the proportionality coefficients, P(A), P(B), P(C) and P(D), respectively. This yields a set of normalized functions:

$$Y_1'=Y_1/0.9466=1 \tag{35}$$

$$Y_2'=Y_2/0.0079=1 \tag{36}$$

$$Y_3'=Y_3/0.0001=1 \tag{37}$$

$$Y_4'=Y_4/0.0454=1. \tag{38}$$

The resulting objective function is

$$G'=(Y_1'-1)^2+(Y_2'-1)^2+(Y_3'-1)^2+(Y_4'-1)^2*(-1) \tag{39}$$

When the estimation program was executed using eq. (39) as the objective function, the best estimators were obtained with an initial guess of

$$P_1=0.01, P_2=0.001, C_1=0.1, C_2=0.1, C_3=0.1, C_4=0.1.$$

The estimated transition probabilities were:

$$P_1'=0.02334$$

$$P_2'=0.5980E(-3)$$

$$C_1'=0.4736$$

$$C_2'=0.4463$$

$$C_3'=0.4462$$

$$C_4'=0.4736.$$

The value of the objective function was −0.6106E(−6). Execution time for this run was 3.20 seconds.

Substituting the estimated transition probabilities into eq. (25-28) produces the following class probability estimates:

$$P'(A)=0.9459$$

$$P'(B)=0.007900$$

$$P'(C)=0.0001$$

$$P'(D)=0.04540.$$

The sum of weighted square error for the estimated class probabilities obtained from eq. (34) is

$$ERR=0.5754E(-6).$$

The best estimates obtained using the normalization technique were two orders of magnitude closer to the experimental results than the best estimates obtained without normalization.

SIMPLIFICATION

The equations for the class probabilities eq. (25-28) were developed by assuming that there might be some large number of transition paths originating and returning to the start state before a transition path to a final state is traversed. This assumption is theoretically correct but it greatly complicates the model. From the transition probabilities estimated thus far, it can be seen that the loops are not expected to be traversed a large number of times. Therefore, an attempt was made to simplify the model by hypothesizing that only direct paths to the final state would be followed.

Under this simplification hypothesis, the following class equations can be derived for the model:

$$P(A)=Z_1*(2P_2*(C_2C_3+C_1C_4) \tag{40}$$
$$+P_1*(C_1+C_2+C_3+C_4)+1)$$

$$P(B)=Z_2*(P_1+P_2C_2)+Z_3*(P_1+P_2C_3) \tag{41}$$
$$+Z_4*(P_1+P_2C_4)+Z_5*(P_1+P_2C_1)$$

$$P(C)=(Z_6+Z_7)*P_2 \tag{42}$$

$$P(D)=P_1C_2+P_1C_4+2P_2C_2C_4 \tag{43}$$
$$+P_1C_1+P_1C_3+2P_2C_1C_3$$

The best estimates for the simplified model were obtained using normalization with initial guesses of

$$P_1=0.001, P_2=0.001, C_1=0.005,$$
$$C_2=0.005, C_3=0.005, C_4=0.005.$$

The estimated transition probabilities were:

$$P_1=0.02438$$

$$P_2=0.6247E(-3)$$

$$C_1=0.4954$$

$$C_2=0.4173$$

$$C_3=0.4310$$

$$C_4=0.4962.$$

The value of the objective function was −0.2242E(−4). Execution time for the run was 8.73 seconds.

The following class probability estimates are obtained for the simplified model:

$$P'(A)=0.9421$$

$$P'(B)=0.007899$$

$$P'(C)=0.0001$$

$$P'(D)=0.04539$$

The sum of the weighted square error for the simplified model was

$$ERR=0.2116E(-4).$$

The error coefficient for the simplified model is two orders of magnitude greater than the coefficient for the full model with normalization. Since the best fit of the simplified model was not as close as the full model, the simplification hypothesis cannot be accepted.

SENSITIVITY TO INITIAL GUESSES

The nonlinear parameter estimation program produced very close estimates when compared with the observed meiosis data. However, the process is heavily dependent upon the initial guesses. Considerable variations in execution time and closeness of fit were obtained for different initial guesses using all three modelling approaches. The extreme cases for each set of equations used are presented in Table I. Reference 3 contains complete details of all estimation runs made and their objective function values and execution times.

Since there are an infinite number of solutions to the model equations there does not exist a procedure to iterate from an arbitrary starting point to the best possible solution. The process of selecting the initial guesses is intuitive. Therefore, the user must perform several runs of the estimation program. The exact number of executions is dependent upon the nature of the experiment and the cost of the runs.

TABLE I

	Maximum Objective Function Value	CPU Time (secs)
Best Estimate—Initial Model	−0.2464E(−6)	2.17
Worst Estimate—Initial Model	−0.2443E(−2)	17.63
Best Estimate—Normalized Model	−0.6106E(−6)	3.20
Worst Estimate—Normalized Model	−0.9604	2.58
Best Estimate—Simplified Model	−0.2242E(−4)	8.73
Worst Estimate—Simplified Model	−0.9588	1.90

CONCLUSION

Nonlinear parameter estimation is used to produce a very close fit to experimental data for the Aviemore model of DNA meiosis. The Aviemore model is represented by a probabilistic finite-state automaton whose estimated parameters are the state transition probabilities. Excellent results in terms of weighted squared error were obtained for the model. However, the estimation procedure is highly dependent upon the initial guesses provided to the program. In this case satisfactory results were obtained in a relatively small number of program executions.

The methods used to estimate the state transition probabilities of the DNA meiosis model are applicable to any probabilistic finite-state automaton model. The transition probability estimation technique could be incorporated into an inference procedure for probabilistic finite-state automata (or grammars).[4] This would provide the capability of automatically synthesizing and parameterizing probabilistic models of systems.

REFERENCES

1. Bard, Y., *Nonlinear Parameter Estimation and Programming,* IBM Scientific Center, 1967.
2. Mortimer, R., private communication, 1975.
3. Wu, K. C., *Parameter Estimation of a Probabilistic Automaton Model of DNA Meiosis,* M.S. Report, Dept. of Computer Science, Kansas State University, Manhattan, Kansas, 66506, 1976.
4. Fu, K. S. and T. L. Booth, "Grammatical Inference: Introduction and Survey—Part II," *IEEE Trans. on SMC,* Vol. 5, No. 4, July, 1975, pp. 409-423.

A comparison between two paradigms of intelligent systems—An example

by ABRAHAM WAKSMAN
Temple University
Philadelphia, Pennsylvania

ABSTRACT

Almost all intelligent computer systems of the past decade could be characterized by the General Problem Solver (GPS) paradigm. This paradigm states that the intelligent system activity consists of two distinct elements considered as separate modules. The first module is the generalist, the general problem solver while the second module could be considered as its data base, consisting of facts about the universe of discourse.

Current research indicates that to bridge the gap between simple display of inference making ability and an actual complex world situation requires a shift in philosophical approach. A new approach which promises to overcome the major drawbacks of the old paradigm could be characterized as the Plan-Debug paradigm. Similar to the old paradigm it could also be characterized as consisting of two modules, the plan making module and the debugging module. Conceptually this paradigm states that in order to execute any task or solve a problem we need to start with a plan of action regardless how imperfect. Once we get stuck, we consult a specialist with a lot of knowledge about the particular situation.

INTRODUCTION

Almost all intelligent computer systems of the past decade could be characterized by the General Problem solver paradigm. This paradigm states that the intelligent system activity consists of two distinct elements considered as separate modules. The first module is the generalist, the general problem solver while the second module could be considered as its data base, consisting of facts about the universe of discourse.

It has become more and more apparent that a fundamental difficulty exists in the building of systems along the General Problem Solver paradigm which are capable of handling more than toy problems.

Current research indicates that to bridge the gap between simple display of inference making ability and an actual complex world situation requires a shift in philosophical approach.

A new approach which promises to overcome the major drawbacks of the old paradigm could be characterized as the Plan-Debug paradigm. Similar to the old paradigm it could also be characterized as consisting of two modules, the plan making module and the debugging module. Conceptually this paradigm states that in order to execute any task or solve a problem we need to start with a plan of action regardless how imperfect. Once we get stuck, we consult a specialist with a lot of knowledge about the particular situation. In this Plan-Debug paradigm the emphasis has shifted from a large and powerful generalist module to a simple plan making module, from a small and simple data base to a large and dynamically structured data base. This shift facilitates the optimization of search processes in a semantically relevant way. It also facilitates the updating without requiring system modifications.

EXAMPLE

Design problems are very often problems in constraint satisfaction. Given a final design as a goal, the problem is to accomplish the goal without violating a set of prespecified conditions concerning the inter-relationships between pa~~rameters. and concerning the resources available. The final~~ design is as a rule, a compromise between these factors in a way which optimizes some predefined criteria.

What follows is an example of the constraint satisfaction problem. We relate it to the two approaches to the design of intelligent systems as discussed above.

Consider a design problem, one which allows no compromises, but rather total constraint satisfaction with no duplications. We feel however, that it is not a toy problem in the sense that the solution will carry over to problems solved on more complex systems, while still employing the underlying concepts.

Our example displays the suitability of the new approach to solving problems encountered in computer aided design. We bring forth the difference in the processing load exhibited by the two systems. We show that the ability to perform more direct searches result in a more effective system.

Computer-aided design problems of the constraint-satisfaction form can be characterized by a query and a set of partial information elements relating to the query. The problem solution corresponds to interrelating the partial information in a nonconflicting way. The response to the query is then directly derivable.

Constraint-satisfaction problems could also be compared to problems of tiling the finite plane with a set of non-regular tiles. The completely tiled plane constitutes the solution. The size of the plane, shape and number of each type of tile used constitute the set of constraints imposed on the solution.

Consider five modules that have to be arranged in a row from left to right, one next to the other. Each module is supposed to perform a specific task different from the tasks of the other modules. To each such task (function), we assign four specific attributes. The problem is to assign one, yet unassigned, attribute to one of the functions, in such a way that a prespecified set of constraints, relating to the modules, functions, and attributes and to the relations between them, will not be violated.

Let us state this problem in terms of the so-called Zebra-problem:

There are five gentlemen who live in a row of five houses.

1. The gentleman that smokes Old-Gold has a snail for a pet.
2. The gentleman that smokes Kool lives in a green house and has a neighbor with a horse for a pet.
3. The gentleman that smokes Chesterfield lives next to the gentleman that has a fox for a pet.
4. The gentleman that smokes Lucky-Strike drinks Orange.
5. The gentleman that smokes Parliament is Japanese.
6. The Spanish gentleman has a dog for a pet.
7. The English gentleman lives in the red house.
8. The gentleman who lives in the green house drinks coffee and is to the right of the ivory house.
9. The Norwegian gentleman who lives in the first house lives next to the blue house.
10. The gentleman in the third house drinks milk.
11. The Ukranian gentleman drinks tea.

The problem is to find out which of the above five gentleman owns the zebra, given that a zebra is one of the five pets belonging to the five gentlemen.

Theorem proving systems based on the general problem solving paradigm, when given a problem such as the zebra problem, will proceed in converting the given constraints into a set of axioms. Using the axioms, they will then proceed to state all possible solutions as theorems. These theorems are to be proven by the system true or to be refuted by it.

The eleven statements of the zebra problem will become then the set of axioms. The theorems to be proven true will become:

Th1. The zebra belongs to the Japanese.
Th2. The zebra belongs to the Englishman.
Th3. The Ukranian owns the zebra.
Th4. The Norwegian owns the zebra.

In order to gain some insight into the processing load imposed by the theorem proving system, we proceed to develop a search path for the proof of Th3.

Theorem:	Uk.—zebra	
Then:	Uk.—not (O.G., Par., L.S.)	Fm. (1,4,5)
Then:	Uk.—or(Kool, Ch.)	
Lemma1:	Uk.—Kool.	
Then:	Uk.—not(1st, 2nd, 3rd, 4th)	Fm. (9,2,7,8)
Then:	Uk.—5th.	
Then:	Ivory—4th.	
	Green—3rd.	Fm. (2,8,12)
Then:	Green—Milk	Fm. (10)
But:	Green—Coffee	Fm. (8)
	Contradiction. (Lemma1)	
Then:	Uk.—not (Kool, O.G., Par., L.S.)	
Then:	Uk.—Chesterfield.	
Then:	Uk.—not (1st., 4th)	
Then:	Uk.—or(2nd., 3rd., 5th.)	
Lemma2:	Uk.—2nd.	
Then:	Uk.—Blue	Fm. (12)
	Red—or(3rd., 5th.)	
	Green—or(3rd., 4th.)	
	Ivory—or(4th., 5th.)	
	Yellow—1st.	
Then:	Uk.—Horse	Fm. (2)
But:	Uk.—Zebra	Fm. (Th3)
	Contradiction. (Lemma2)	
Then:	Uk.—not(1st., 2nd., 4th.)	
Then:	Uk.—or(3rd., 5th.)	
Lemma3:	Uk.—3rd.	
Then:	Uk.—Milk	Fm. (10)
But:	Uk.—Tea	Fm. (11)
	Contradiction. (Lemma3)	
Then:	Uk.—not(1st., 2nd., 3rd., 4th.)	
Then:	Uk.—5th.	
	4th.—Ivory	Fm. (8)
	3rd.—Green	Fm. (8)
	2nd.—Blue	Fm. (12)
	1st.—Yellow	Fm. (2)
Then:	Uk.—Red.	
But:	Eng.—Red.	
	Contradiction. (Th3)	

We have thus disproved that the Ukranian gentleman is the owner of the zebra. We still have to tackle the other three theorems. Even when we discover a proof for the validity of one of the theorems, we cannot stop since there might be more then one solution. The constraints might be satisfied in more than one way.

Inherent in a solution as the above is the inability to make use of prior deductions to aid in the solution process. It is, however, possible to derive and save axioms which are needed more then once. In our case, the fact that the second house is blue (Axiom 12) is such as axiom, deduced from the fact that the 1st house has a blue house for a neighbor (Ax. 9).

A somewhat longer chain of deduction is needed to conclude that the Norwegian lives in the yellow house (9, 8, 12, 7). An automatic system such as a theorem prover cannot have, however, prior knowledge as to the utility of a given deduction that results in a new axioms derived from old axioms.

The danger in giving a theorem prover freedom to derive new axioms is that proliferation of axioms can very easily get out of hand causing the solution process to bog down for any moderate size problem.

In contrast to the theorem prover, the plan-debug system approach to the solution of constraint-satisfaction problems consists of relegating the pre-processing activity to a set of specialized procedures which have the responsibility to do the domain-specific processing before the main algorithm is getting activated and when it is confronted with a conflict. We can divide the plan-debug system structure into three main modules as follows:

1. A data base that consists, initially, of the problem-constraints as its data entities.
2. A set of "trigger-functions" or demons. These demons could be considered axiom-activated update functions. Each demon is associated with an axiom or a set of axioms which activate (envoke) it whenever they enter into new relation with other axioms. The demons' task is to insure that no side effect of any newly formed association is left uninspected, unrecorded or reported if need be. The demons might add new axioms to the data base; they might introduce data base elements into new relations; and will declare conflicts as a consequence of improper or illegal update.

A *color-demon*, for example, might have the responsibility to see to it that:

A. No color is used in more than one house.
B. No house can have two colors.
C. The 'next-to' data type has the following properties;
  a. IF Loc.—5th. Then Next-to—4th.
  b. IF Loc.—1st. Then Next-to—2nd.
  c. IF Loc.—x. Then Next-to—or((x−1), (x+)).
Also, IF Loc.—x. Then To-The—Right-of—(x−1).

3. A general purpose search algorithm, the planner, which follows simple guidelines for the initial search. As the search progresses, the data base upon which the planner acts gets modified by the demons. Demons that remain active for any length of time become temporarily part of the planner and remain active under its control. This is, in effect, a form of parallel processing in the sense that when update occurs, the planner is considering the total data base in terms of a solution rather then one part of it at a time. Each update will, in effect, generate a modified version of the data base together with a list of modifications that occurred since the start of the solution process.

We can characterize the activity of the planner as follows:

1. Consider every constraint as an incomplete tuple ranging over all the domains in the data base. Thus the constraint includes, besides the original set of domain-value pairs, domain-value pairs with unspecified value.
2. The logical deduction process has the effect of introducing the appropriate values to fields with unspecified values.
3. Whenever a value is entered in an incomplete tuple's field, the planner checks to see if some other incomplete tuple has such a value under a similar domain. In that case a join is performed, that is, the two incomplete tuples are combined into one resulting in less unspecified fields.
4. A solution exists whenever there are no more incomplete tuples in the data base. That is, when all the tuples ranging over all the available domains consist of domain value pairs and all the values are uniquely specified. This will also result in the smallest number of tuples in this final version of the data base.

Let us consider the solution process in a plan-debug system as described above:

The color-demon establishes at the start of the computations the following fact:

(Blue—2nd.) and (Green—to-the-right-of Ivory)
Ivory—or(4th., 5th.)
Lemmal: Ivory—4th.
Then: Yellow—1st.
Blue—2nd.
Green—3rd.
Red—5th.
Then: Coffee—3rd.
But: Milk—3rd.
Contradiction. (Lemmal)
Then: Ivory—5th.
Then: Uk.—or(2nd., 5th.)
Lemma2:: Uk.—2nd.
Then: (since there are no contradictions to lemma2,) we proceed to represent the five complete tuples in a table form. In actuality, such a table will result from the collapsing of the data base with the incomplete tuples as tuples continue to be joined.

Cig.	Pet	Color	Loc.	Nat.	Drink	Next (Pet)	(Color) To-R-of
Kool	Fox	Yellow	1st.	Nor.	—	Horse	
Che.	Horse	Blue	2nd.	Uk.	Tea		
O.G.	Snail	Red	3rd.	Eng.	Milk		
Par.	Zebra	Green	4th.	Jp.	Coffee		Ivory
L.S.	Dog	Ivory	5th.	Sp.	Orange		

In the process of generating a duplicate data base for the case of (Uk.- - - - 5th.) we discover a contradiction as follows:

Lemma3: Uk.- - - - 5th.

Then: 2nd—Japanese— Orange—Parliament.

But: Orange—Lucky Strikes

Contradiction. (Lemma3)

Thus, the above table represents the only possible solution to the zebra problem. The zebra belongs to the Japanese and the data base consists of five tuples ranging over nine domains each. This is the only combination of parameters which will allow for a solution without a conflict.

CONCLUSION

The dramatic improvement in the way that the plan-debug system handles the zebra problem as against the theorem prover is due fundamentally to the ability of the set of demons to capture in a procedural way the semantics of the data base (the set of constraints). This knowledge, which is specific to the problem at hand, need not be encapsulated in a more general way in the main algorithm.

The main algorithm, the planner, uses general deduction to add values to fields in incomplete tuples. It also performs the joins.

The debug facility introduces special deduction initially as well as when the general deduction is not adequate.

REFERENCES

1. Newell, Allen, "Artificial Intelligence and the Concept of Mind," in R. C. Shank (ed), *Computer Models of Thought and Language,* W. H. Freeman and Co., San Francisco, 1973.
2. Minsky, Marvin, "New Directions in Artificial Intelligence," a talk presented at IBM, San Jose, California, Summer 1976.

Concatenated group theoretic codes for binary asymmetric channels*

by SERBAN D. CONSTANTIN and T. R. N. RAO
Southern Methodist University
Dallas, Texas

ABSTRACT

A brief description of group theoretic codes is given and their suitability for binary asymmetric channels is exemplified.

Previous research has shown the superiority in the information rate of the group theoretic codes over the existing codes for binary asymmetric channels and has left open the problems posed by the encoding/decoding procedures.

The present paper introduces more sophisticated codes constructed from the already existing single 1-error correcting group theoretic codes.

The new class of codes, which we will refer to as concatenated group theoretic codes, will have improved encoding/decoding features while maintaining a high information rate comparable with that of equivalent length group theoretic codes.

As their name indicates, a code of length 2n will be obtained by concatenating two sets of group theoretic codes of length n.

INTRODUCTION

Given an abelian group G of order n+1, one can put in $1-1$ correspondence the binary vectors of length n with the linear combinations (with coefficients 0 or 1) of non-zero elements of G.

The correspondence is rather intuitive and is given by:

$$\sum_{i=1}^{n} \alpha_i \cdot a_i \leftrightarrow (\alpha_1, \alpha_2, \ldots, \alpha_n) \qquad (1.1)$$

$$\text{where } \begin{cases} 0\cdot a_i = a_0 \\ \qquad i=1, 2, \ldots, n \\ 1\cdot a_i = a_i \end{cases}$$

$a_i \neq a_0$ are the non-zero elements of the group and

$\alpha_i = 0, 1$

Without loss of generality, assume the group operation to be addition. Moreover, the above correspondence will partition the set of 2^n binary vectors V of length n into n+1 disjoint classes of vectors $V_0, V_1, \ldots, V_n$.

The linear combinations corresponding to vectors of a class V_i, will sum up to a_i, and vectors of each class will form a group theoretic code. Since one tries to optimize the number of codewords in the code, we shall look for the set V_i, having the most number of vectors (codewords) in it.

If the best possible code obtainable by this method is desired, one must consider all abelian groups or order n+1 and look at the classes generated by each group and then select the largest such class.

For an n as small as 10, one must use a computer in order to generate all classes of vectors generated by a group of order 11 or higher.

The error correcting properties and a more detailed description of group theoretic codes can be found in Reference 1.

For a comparison of Hamming code[4] and Kim and Freiman code[3] with the group theoretic codes of the same length the reader is referred to Table I.

In this paper Z_n will have the standard meaning of the addition modulo n group.

MATHEMATICAL CONSIDERATIONS

Consider two vectors:

$$\begin{aligned} A&=(a_1, a_2, \ldots, a_n) \\ B&=(b_1, b_2, \ldots, b_n) \end{aligned} \qquad (2.1)$$

where

$a_i, b_i \quad i=1, \ldots, n$ are real numbers.

Without loss of generality let's assume:

$$\begin{cases} a_1 \leq a_2 \leq a_3 \leq \cdots \leq a_n \\ b_1 \leq b_2 \leq b_3 \leq \cdots \leq b_n \end{cases} \qquad (2.2)$$

If π is a permutation, $\pi=(i_1, i_2, \ldots, i_n)$, then by $\pi(B)$ we will denote

$$\pi(B)=B_\pi=(b_{i_1}, b_{i_2}, \ldots, b_{i_n})$$

* The authors' research was supported in part by a grant from National Science Foundation Eng76-11237.

TABLE I—Number of Codewords for Single-Error Correcting Codes

Code Length n	Hamming Code	Kim-Freiman Code	Group Theoretic Codes $G=Z_{n+1}$†	Group Theoretic Codes **
2	2	2	2	2
3	2	2	2	2
4	2	$2+2=4$	4*	4
5	$2^2=4$	$2^2+2=6$	6	
6	$2^3=8$	$2^3+2^2=12$	10*	10
7	$2^4=16$	$2^3+2^2=12$	16	16
8	$2^4=16$	$2^4+2^3=24$	30	32
9	$2^5=32$	$2^5+2^3=40$	52	
10	$2^6=64$	$2^6+2^4=80$	94*	94
11	$2^7=128$	$2^7+2^4=144$	172	
12	$2^8=256$	$2^8+2^5=288$	316	316
13	$2^9=512$	$2^9+2^5=544$	586	
14	$2^{10}=1024$	$2^{10}+2^6=1088$	1096	
15	$2^{11}=2048$	$2^{10}+2^6=1088$	2048	2048
16	$2^{11}=2048$	$2^{11}+2^7=2176$	3856	3856
17	$2^{12}=4096$	$2^{12}+2^7=4224$	7286	
18	$2^{13}=8192$	$2^{13}+2^8=8448$	13798	13798
19	$2^{14}=16384$	$2^{13}+2^8=8448$	26216	
20	$2^{15}=32768$	$2^{14}+2^9=16896$	49940	

* Figures in these entries are best possible using this method, due to the fact that there is only one group of order n+1. See Reference 5 for the number of groups of various order.

** The groups considered were the additive groups of the Galois fields GF(p^q).

Note: The number of codewords in the group theoretic codes in the above table was generated by computer.

† These codes were also obtained by Varshamor.[6]

i.e., the vector whose elements are the elements of B permuted according to π.

We are interested in finding π^* such that $A \cdot B_\pi \leq A \cdot B_{\pi^*}$ holds for any other permutation π, where by $A \cdot B$ is meant the scalar product of the two vectors i.e.,

$$A \cdot B = \sum_{i=1}^{n} a_i \cdot b_i \tag{2.3}$$

This maximum scalar product i.e., $A \cdot B_{\pi^*}$ will be referred to M(A, B), where M acts like an operator on the two vectors A and B into the real numbers.

Lemma 1

For any two vectors A and B satisfying (2.2):

$$M(A \cdot B) = A \cdot B = B \cdot A \tag{2.4}$$

Proof

Let's consider first the trivial case, i.e.,

$$A=(a_1, a_2),\ B=(b_1, b_2)$$

such that

$$a_1 \leq a_2,\ b_1 \leq b_2$$

What we have to show then to prove the lemma in this case is:

$$a_1b_2 + a_2b_1 \leq a_1b_1 + a_2b_2 \tag{2.5}$$

This follows immediately from:

$$(a_2 - a_1)(b_2 - b_1) \geq 0 \tag{2.6}$$

by expanding the product.

Having proved this, to get the general result we are only one step away. Consider any two permutations π_1 and π_2 and assume:

$$M(A, B) = \pi_1(A) \cdot \pi_2(B)$$

and let p be the largest integer such that the scalar product $\pi_1(A) \cdot \pi_2(B)$ does not contain $a_p \cdot b_p$ as a term. Then $\pi_1(A) \cdot \pi_2(B)$, must certainly contain $a_p \cdot b_i + a_j \cdot b_p$

where

$$a_j \leq a_p \quad \text{and} \quad b_i \leq b_p$$

But then using the trivial case considered above, i.e.,

$$a_p \cdot b_i + a_j \cdot b_p \leq a_j \cdot b_i + a_p \cdot b_p$$

substituting $a_p \cdot b_i + a_j \cdot b_p$ in the scalar product $\pi_1(A) \cdot \pi_2(B)$ with $a_j \cdot b_i + a_p \cdot b_p$ we contradict that $M(A, B) = \pi_1(A) \cdot \pi_2(B)$. Repeating the procedure described until p=1 we obtain the desired result. Q.E.D.

Corollary 1:

For any arbitrary vector $A=(a_1, a_2, \ldots, a_n)$

$$M(A, A) = A \cdot A = \sum_{i=1}^{n} a_i^2 \tag{2.7}$$

Similarly, if we define the operator m(A, B) to be:

$$m(A, B) = A \cdot B_{\pi^*} \leq A \cdot B_\pi \tag{2.8}$$

for any permutation π, then the following similar result is obtained:

Lemma 2

For any two vectors A and B satisfying (2.2):

$$m(A, B) = \sum_{i=1}^{n} a_i \cdot b_{n-i+1} \tag{2.9}$$

Proof:

Similar to the one for Lemma 1.

If the elements of a vector A are non-positive and the elements of a vector B are non-negative the following relations hold:

$$\begin{cases} M(A, B) = -m(|A|, B) \\ m(A, B) = -M(|A|, B) \end{cases} \tag{2.10}$$

where

$$|A| = (|a_1|, |a_2|, \ldots, |a_n|).$$

Finally, one more result is needed. We want to construct a single error correcting code of length two, over the alphabet {0, 1, 2, . . . , n}, where the only possible single errors occurring in a codeword (i, j) are:

$$\begin{cases}(i,j)\rightarrow(i-1,j) & i\neq 0\\ (i,j)\rightarrow(i,j-1) & j\neq 0\end{cases} \tag{2.11}$$

Maximizing the number of codewords in such a code is also one of the objectives. Consider the following picture:

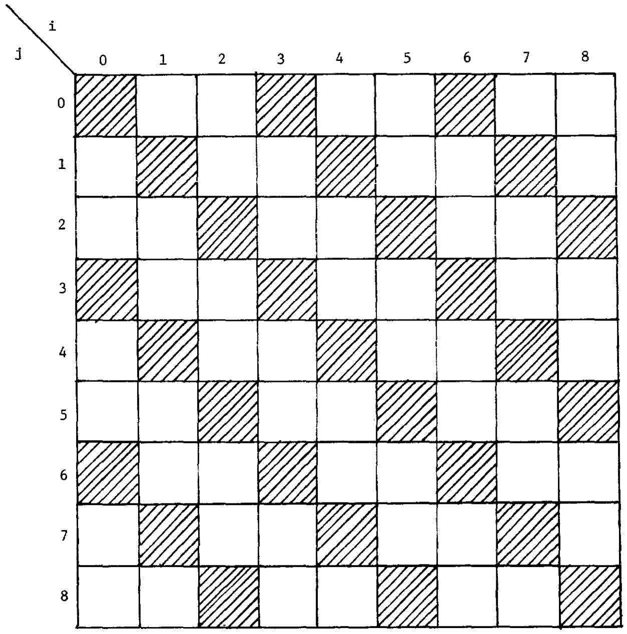


Figure 1

Let (i, j) be the name of the square in the ith row and jth column. The collection of the (i, j) 2-tuples corresponding to the shadowed squares in Figure 1, can easily be checked to form a single error correcting code under the conditions of error occurrence as described by (2.11). Note that no additional square could be shadowed such that the augmented code be still single error correcting. However, this does not prove that no other code could have more codewords than this code.

Lemma 4

Consider the following length 2 code over the alphabet {0, 1, 2, . . . , n}:

(a) (i, i) is a codeword for i=0, 1, 2, . . . , n
(b) if (i, j) is a codeword, so is (i+3, j) and (i, j+3) for i+3≤n and j+3≤n

Then, such a code is cyclic, is single error correcting and contains the maximum number of codewords.

Proof.

The first two assertions can be easily disposed of using the definition of the code and (2.11). We shall prove only the third property, i.e., no other code can have more codewords than the code described above.

Obviously, the number of possible 2-tuples over the alphabet {0, 1, 2, . . . , n} is $(n+1)^2$. Let $n+1=3m+k$, $k=(n+1) \bmod 3$. Then, the number of codewords in our code is:

$$(n+1)+\sum_{i=1}^{m} 2\cdot[(n+1)-3\cdot i]=3\cdot m^2+2\cdot m\cdot k+k \tag{2.12}$$

In general, for every codeword (i, j) that we include in our code, we eliminate from the list of potential codewords two other tuples, (i−1, j) and (i, j−1); i.e., the contaminated tuples corresponding to the codeword (i, j). With this observation, one can actually convert the original problem into a tile covering problem, namely we will be concerned with covering an (n+1)×(n+1) rectangle of squares with tiles of the shape and orientation of the one in Figure 2, such that no two tiles overlap and as much as possible of the surface of the (n+1)×(n+1) rectangle is covered with tiles.

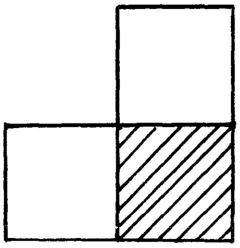


Figure 2

The simplest upper bound for the number of tiles covering the rectangle is

$$\text{max number of tiles} \leq \left[\frac{\text{area of the surface}}{\text{area of a single tile}}\right]=\left[\frac{(n+1)^2}{3}\right]$$

$$=\left[\frac{(3m+k)^2}{3}\right]=3m^2+2mk+\left[\frac{k^2}{3}\right]=3m^2+2mk+k$$

because

$$\left[\frac{0^2}{3}\right]=0,\ \left[\frac{1^2}{3}\right]=1,\ \left[\frac{2^2}{3}\right]=2$$

But, the upper bound coincides with the number of codewords in our code. Q.E.D.

The method of finding such codes can be generalized in different directions. For example, one can build a single error correcting length 2 code where the first component can take on values from {0, 1, . . . , n} and the second component can take on values from {0, 1, . . . , m}. The code could be constructed using a similar picture as the one on Figure 1 except that the surface will be an (n+1)×(m+1) rectangle.

Lemma 5

Given N=m+n, N fixed, the length 2 code with the most number of codewords, is obtained for n=N/2, m=N/2.

Proof. Area of the $(n+1)\times(m+1)$ rectangle is maximum when $n=m$. Q.E.D.

Using extensions of the rules given in Lemma 4, one can build a single error correcting length k code over the alphabet $\{0, 1, \ldots, n\}$. Proving the maximality of the number of codewords of the length k code constructed by the rules similar to the ones in Lemma 4 requires more involved calculations.

At last, the length 2 code over the alphabet $\{0, 1, \ldots, n\}$ will be called a weight-code and will be used for constructing the concatenated group theoretic codes.

CONSTRUCTION OF CONCATENATED GROUP THEORETIC CODES

Let G_1 and G_2 be two abelian groups of the same order $n+1$ (as it will be seen G_1 and G_2 need not necessarily be distinct) and let $\{V_0, V_1, \ldots, V_n\}$ and $\{U_0, U_1, \ldots, U_n\}$ be the partitions of the set of 2^n binary vectors of length n into $n+1$ disjoint classes of codes as induced by the two groups G_1 and G_2 respectively.

Each class (code) V_i, U_i $i=0, 1, 2, \ldots, n$ is a group theoretic code in itself. Distinguishing the codewords in each class by their weight (i.e., # of 1's in the codewords) we obtain the following classification

TABLE II

class \ weight	0	1	2 .... n
V_0	a_{00}	a_{01},	.... a_{0n}
V_1	.	.	.
⋮	⋮	⋮	⋮
V_n	a_{n0}	a_{n1},	.... a_{nn}

TABLE III

class \ weight	0	1	2 .... n
U_0	b_{00}	b_{01},	.... b_{0n}
U_1	.	.	.
⋮	⋮	⋮	⋮
U_n	b_{n0}	b_{n1},	.... b_{nn}

where

$a_{ij}=$ # of vectors (codewords) of weight j in the class V_i
$b_{ij}=$ # of vectors (codewords) of weight j in the class U_i

The length 2n code we will construct, as its name suggests, will be the result of concatenating codewords from some class V_i with codewords from some class U_j in a manner that will result in a maximum number of codewords of length 2n.

For ease of reference, let's adopt the following notations:

- C the concatenated group theoretic code to be constructed
- W the weight-code. This is a length 2 code, over the alphabet $\{0, 1, \ldots, n\}$ as described earlier.
- C_1 the set of codes generated by G_1 i.e.: $V_0, V_1, \ldots, V_n$
- C_2 the set of codes generated by G_2 i.e.: $U_0, U_1, \ldots, U_n$
- lower case letters will be used to denote codewords
- $|w|$ weight of codeword w.

Then, a condensed description of C could be given in the following form:

$$C=\{(w_1w_2) \| |w_1|=i,\ |w_2|=j,\ (i, j)\in W,\ w_1\in V_k,\ w_2\in U_l) \tag{3.1}$$

a_{ki} and b_{lj} are both the rth largest elements, for some r, in the ith column of Table II and jth column of Table III, respectively}

For the case where G_1 and G_2 have been selected to be one and the same group, in the above definition (3.1) pick $k=l$. (See Corollary 1). Given the two groups G_1 and G_2 of order $n+1$, one selects a length 2 weight-code, as described in Lemma 4, determining what weights the codewords to be paired together should have.

Let C_i' denote the ith column of Table II and C_j'' denote the jth column of Table III, $i, j=0, 1, 2, \ldots, n$.

Then, for every codeword (i, j) of the weight-code W, we will generate codewords (w_1w_2) of C, where w_1 is a codeword in C_1 of weight i corresponding to some code V_k and w_2 is a codeword in C_2 of weight j corresponding to some code U_l where k and l are selected as dictated by Lemma 1 applied to the vectors C_i' and C_j''.

Error detection and correction in C:

Let (r_1r_2) be a received message and let's assume that at most one 1-error has occurred to the transmitted message (w_1w_2) of C. From the construction of C we know $(|w_1|, |w_2|)\in W$.

Let $i=|r_1|$ and $j=|r_2|$. If (i, j) is a valid codeword of W then $(r_1r_2)=(w_1w_2)$ and hence no error has occurred. However, if $(i, j)\notin W$ then a single 1-error has occurred to the transmitted message (w_1w_2) and more precisely a single 1-error has occurred to either w_1 or w_2 decreasing the weight of one of the two codewords by 1. Therefore, either $i=|w_1|$ and $j=|w_2|-1$; i.e. the error has occurred in w_2 or $i=|w_1|-1$ and $j=|w_2|$; i.e. the error has occurred in w_1. Since W is a single error correcting code we can determine$(|w_1|, |w_2|)$ and hence know where the single 1-error has occurred. Let's suppose the error has occurred in w_2 and as a result we have received $(r_1r_2)=(w_1r_2)$. By the correspondence (1.1) and the structure of G_1 we establish the membership of w_1 in some class (code) V_k. Let r be the rank of a_{ki} in C_i'. Find then the rth largest element (the element of rank r) in C_{j+1}'' and let this be $b_{l(j+1)}$. Now we know that r_2 must have come from a codeword of U_l and since U_l is a single 1-error correcting code we can correct r_2 and obtain w_2. Thus, we will correct (r_1r_2) and obtain the transmitted message (w_1w_2) of C.

The following example goes through each step of the detecting and correcting procedure described in the previous paragraph for a length 16 concatenated group theoretic code generated by the additive group of the Galois field GF(3^2).

Example

Consider the additive group G_9 of the Galois field GF(3^2) whose addition table is given below:

	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8
+	0	1	2	X	2X	1+X	2+X	1+2X	2+2X
0	0	1	2	X	2X	1+X	2+X	1+2X	2+2X
1	1	2	0	1+X	1+2X	2+X	X	2+2X	2X
2	2	0	1	2+X	2+2X	X	1+X	2X	1+2X
X	X	1+X	2+X	2X	0	1+2X	2+2X	1	2
2X	2X	1+2X	2+2X	0	X	1	2	1+X	2+X
1+X	1+X	2+X	X	1+2X	1	2+2X	2X	2	0
2+X	2+X	X	1+X	2+2X	2	2X	1+2X	0	1
1+2X	1+2X	2+2X	2X	1	1+X	2	0	2+X	X
2+2X	2+2X	2X	1+2X	2	2+X	0	1	X	1+X

For each i, i=0, 1, . . . , 8 the codewords in V_i form a group theoretic code, and for purposes of clarity, the codewords of such a code, namely of V_0, will be listed out. They are:

$c_0=(0,0,0,0,0,0,0,0)$; $c_1=(1,1,0,0,0,0,0,0)$; $c_2=(0,0,1,1,0,0,0,0)$
$c_3=(0,0,0,0,1,0,0,1)$; $c_4=(0,0,0,0,0,1,1,0)$; $c_5=(1,1,1,1,0,0,0,0)$
$c_6=(1,1,0,0,1,0,0,1)$; $c_7=(1,1,0,0,0,1,1,0)$; $c_8=(0,0,1,1,1,0,0,1)$
$c_9=(0,0,1,1,0,1,1,0)$; $c_{10}=(0,0,0,0,1,1,1,1)$; $c_{11}=(1,1,1,1,1,0,0,1)$
$c_{12}=(1,1,1,1,0,1,1,0)$; $c_{13}=(1,1,0,0,1,1,1,1)$; $c_{14}=(0,0,1,1,1,1,1,1)$
$c_{15}=(1,1,1,1,1,1,1,1)$; $c_{16}=(1,0,1,0,0,0,0,1)$; $c_{17}=(0,1,0,1,1,0,0,0)$
$c_{18}=(0,1,1,0,0,0,1,0)$; $c_{19}=(1,0,0,1,0,1,0,0)$; $c_{20}=(1,0,0,0,1,0,1,0)$
$c_{21}=(0,1,0,0,0,1,0,1)$; $c_{22}=(0,0,1,0,1,1,0,0)$; $c_{23}=(0,0,0,1,0,0,1,1)$
$c_{24}=(1,1,1,0,1,1,0,0)$; $c_{25}=(1,1,0,1,0,0,1,1)$; $c_{26}=(1,0,1,1,1,0,1,0)$
$c_{27}=(0,1,1,1,0,1,0,1)$; $c_{28}=(0,1,1,0,1,0,1,1)$; $c_{29}=(1,0,0,1,1,1,0,1)$
$c_{30}=(1,0,1,0,0,1,1,1)$; $c_{31}=(0,1,0,1,1,1,1,0)$

The distribution by weight of the codewords in the nine classes V_0, V_1, . . . , V_8 induced by G_9 into the set of 2^8 binary vectors of length 8 is given by the following table:

TABLE IV

class \ Weight	0	1	2	3	4	5	6	7	8
V_0	1	0	4	8	6	8	4	0	1
V_1	0	1	3	6	8	6	3	1	0
V_2	0	1	3	6	8	6	3	1	0
V_3	0	1	3	6	8	6	3	1	0
V_4	0	1	3	6	8	6	3	1	0
V_5	0	1	3	6	8	6	3	1	0
V_6	0	1	3	6	8	6	3	1	0
V_7	0	1	3	6	8	6	3	1	0
V_8	0	1	3	6	8	6	3	1	0

The length 16 concatenated group theoretic code C will be constructed from C_1, the set of codes generated by $G_1=G_9$; and C_2, the set of codes generated by $G_2=G_9$. In this case G_1 and G_2 will be identical and equal to the addition group of GF(3^2). This implies that C_1 and C_2 will be identical and equal to the set of codes V_0, V_1, . . . , V_n. The weight-code W will be the one determined by the names of the shadowed squares of the picture in Figure 1.

To each codeword (w_1w_2) of C it will correspond a codeword ($|w_1|$, $|w_2|$) of W and conversely to each codeword (i, j) of W there will correspond a set of codewords in C. The number of codewords in C corresponding to a codeword (i, j) in W is given in the following table:

TABLE V

$\vert w_1\vert$ \ $\vert w_2\vert$	0	1	2	3	4	5	6	7	8
0	1			8			4		
1		8			64			8	
2			88			176			4
3	8			288			176		
4		64			512			64	
5			176			288			8
6	4			176			88		
7		8			64			8	
8			4			8			1

Let

$$V_i^j=\{v|v\in V_i,\ |v|=j\}$$

and (3.3)

$$(V_i^j\cdot V_k^l)=\{(v_1v_2)|v_1\in V_i^j,\ v_2\in V_k^l\}$$

Then, using the notation (3.3), the set of codewords (w_1w_2) of C corresponding to the codeword (2, 5) of W is:

$$\{(V_0^2\cdot V_0^5),(V_1^2\cdot V_1^5),(V_2^2\cdot V_2^5),\ldots,(V_8^2\cdot V_8^5)\}$$

which comprises a total of 176 codewords.

To see how detection and correction is done when a single 1-error has occurred, let:

$$r=(r_1r_2)=(1010101001001001)$$

be the received message and assume (w_1w_2) of C was the actually transmitted message.

For the above received message we have:

$$i=|r_1|=|(10101010)|=4, j=|r_2|=|(01001001)|=3$$

and since (4, 3) is not a codeword of W and assuming a single 1-error has occurred, the codeword of W corresponding to the transmitted message should have been (4, 4). Thus, r_2 is in error.

Since

$$a_1+a_0+a_3+a_0+a_5+a_0+a_7+a_0=a_3$$

it must be that

$$r_1=(10101010)\in V_3$$

$|V_3^4|=8$ and 8 is the largest element in the 4th column of Table IV. Since there are seven other entries equal to 8, suppose the following pairing of codewords of C_1 and C_2 corresponding to the codeword (4, 4) of W, has been done:

$$(V_0^4\cdot V_0^4),(V_1^4\cdot V_8^4),(V_2^4\cdot V_7^4),(V_3^2\cdot V_5^4),(V_2^4\cdot V_6^4)$$

(except for $(V_0^4\cdot V_0^4)$ all the other pairings are done arbitrarily; yet one must know before trying to correct transmitted messages what the pairings are).

Therefore, r_1 must have been paired with a codeword of weight 4 from V_6. However

$$a_0+a_2+a_0+a_0+a_5+a_0+a_0+a_8=a_2$$

and the error has occurred in the third position as given by

$$a_6+(-a_2)=a_6+a_1=a_3\quad(-a_2\text{ is the additive inverse of }a_2)$$

Thus the transmitted message was $(w_1w_2)=(1010101001101001)$.

The number of codewords in C as given in Table V is 2470 and comparing it with the Kim & Freiman code of length 16 or the Hamming code of the same length (see Table I) we see a definite improvement in the information rate. Although the number of codewords in C is slightly less than the number of codewords in the group theoretic code of length 16, efficiency could be gained in the encoding/decoding process.

If ROM was to be used for encoding and decoding purposes, smaller size ROM could be utilized if a message $m=(m_1m_2)$ was encoded as:

$$(m_1m_2)\Rightarrow\left\{\begin{matrix}m_1\mapsto w_1\\ m_2\mapsto w_2\end{matrix}\right\}\Rightarrow(w_1w_2)\in C$$

and decoded as:

$$(w_1w_2)\Rightarrow\left\{\begin{matrix}w_1\mapsto m_1\\ w_2\mapsto m_2\end{matrix}\right\}\Rightarrow(m_1m_2).$$

Longer weight-codes could be used in generating concatenated group theoretic codes, but it appears as though this would have a negative impact on the encoding/decoding procedures and maybe in the correction process while accomplishing a high information rate. An optimal concatenation (pairing) of three or more smaller codewords into a longer code is also not apparent, and could be regarded as a natural generalization of the method presented in this paper.

CONCLUSIONS

The future research to be pursued by the authors of this paper will be focused in the direction of finding efficient encoding/decoding procedures for both group theoretic codes as well as concatenated group theoretic codes.

Finding an efficient encoding/decoding procedure for the two types of codes is believed to be possible due to the structure of the two codes inherited from the groups that have generated them.

REFERENCES

1. Constantin, Serban D. and T. R. N. Rao, "Group Theoretic Codes for Binary Asymmetric Channels," Technical Report CS 76014, Department of Computer Science, Southern Methodist University, Dallas, TX.
2. Rao, T. R. N. and A. S. Chawla, "Asymmetric Error Codes for Some LSI Semiconductor Memories," *7th Annual Southeastern Symposium on System Theory,* March 1975. pp. 170-171.
3. Kim, Wan H. and Charles V. Freiman, "Single Error Correcting Codes for Asymmetric Channels," *I.R.E. Transactions on Information Theory,* June 1959.
4. Peterson, W. W. and E. J. Weldon, Jr., *Error Correcting Codes,* M.I.T. Press, Cambridge, Massachusetts, 1970.
5. Hall, Marshall Jr., *The Theory of Groups,* Macmillan, 1959.
6. Varshamor, R. R., "A Class of Codes for Asymmetric Channels and a Problem from the Additive Theory of Numbers," *Trans. on Information Theory,* January 1973, pp. 92-95.

The TICOM model—A network data base approach to review and evaluation of internal control systems

by JAMES I. CASH, JR
Harvard University
Boston, Massachusetts

and

ANDREW D. BAILEY, JR. and ANDREW B. WHINSTON
Purdue University
West Lafayette, Indiana

ABSTRACT

EDP based accounting information systems have grown in complexity and size. This growth has partially been the result of new and advanced software techniques introduced by computer scientists. This paper is one in a series of papers which provide a new perspective for auditors of EDP based AIS's facilitated by new software development methodologies which address development of "reliable" software systems. In another paper we asserted reliable software would eliminate the need to verify computer programs except for an authenticity check; thus, facilitating a more thorough examination of the total internal control system. We describe a model which facilitates review and evaluation of internal control systems from a "total" systems perspective.

INTRODUCTION

In earlier papers, we surveyed the extant literature on verification techniques for EDP based accounting information systems (AIS), and presented a methodology that facilitates the development of reliable AIS software systems.[1,16] In the second paper, we asserted that reliable software would eliminate the need to verify computer programs except for an authenticity check; thus, facilitating a more thorough examination of the total internal control system. This paper describes a model which facilitates review and evaluation of internal control systems from a "total" systems perspective. The model can be applied to any existing AIS and will contribute significantly to the auditor's comprehension and testing of the AIS. However, its greatest potential impact is in its application to reliable AIS. The fourth paper in this series presents the detailed development of the TICOM model presented in this paper.

INTERNAL CONTROL OVERVIEW

Auditing objectives have changed over time. R. Gene Brown[2] traced this evolution as shown in Figure 1.

Period	
1500-1850	Detection of fraud
1850-1905	Detection of fraud and clerical errors
1905-1940	Detection of fraud and clerical errors; determination of fairness in reporting
1940	Present determination of fairness in reporting

Figure 1—Evolution of auditing objectives

Implicit in each of these objectives was the method and extent of system evaluation and verification. Fitzpatrick[3] states that during the sixteenth century, when auditing existed specifically to verify the honesty of persons charged with fiscal responsibilities, the extent of verification was very detailed, while evaluation of internal control was not considered a relevant function in performing audit objectives. Currently, substantial emphasis is placed on evaluation of internal control as a means of determining the scope and extent of verification. This emphasis is reflected in the second standard of field work included in the ten generally accepted auditing standards:

> There is to be a proper study and evaluation of the existing internal control as a basis for reliance thereon and for the determination of the resultant extent of the tests to which auditing procedures are to be restricted.[4]

As explicit as this statement is on a general level, very few aids exist to help the auditor make *objective* statements and judgments about a specific internal control system at the overall level. Numerous articles exist using statistical techniques to make statistically objective statements concerning the internal control system. However, the statements made are related to very specific and limited issues within the system and nothing is said about the system at a more general level.

Numerous professional statements can be found which suggest that this is an area requiring the highest degree of judgment and professional experience. Though we believe no audit tool will or should completely remove professional judgment from this function, most researchers continue to fail in the development of a general model with which the auditor can ask for and receive objective evidence about his client's internal control system. Bodnar,[5] Ishikawa,[6] Yu and Neter,[7] and Cushing[8] are researchers who provided models which addressed this objective, formal evaluation problem, but did not provide a facile means of interface with the models. Other writers have suggested effective subjective means of exploring a system for strengths and weaknesses, but do not provide straightforward criteria for decisions about the system. The work contained herein addresses several important issues:

(a) provides an encoding mechanism for internal control systems that views automated and manual procedures equally which provides a total systems perspective for review and evaluation.
(b) uses recent EDP technological advances to facilitate review and evaluation of the system which provides for more thorough investigation.
(c) facilitates positional analysis.
(d) eliminates "slanted" questions on a questionnaire.
(e) allows simulation of accounting subsystems to check for "lagged" (also termed "compensatory") control procedures, and subsystem overlap that might condone fraudulent activity.
(f) facilitates viewing internal control systems at different levels of detail which allows the auditor to specify the level of detail needed to perform the review and evaluation.

CHARACTERISTICS OF INTERNAL CONTROL

Internal controls are organizational arrangements and the actions instituted under such arrangements taken within an organization to direct and regulate activities of that organization. Both management and auditors have recognized the potential benefits of effective internal control. More specifically, auditors have realized that improved internal control ". . . permits reductions in audit work made possible by the concomitant increase in the credibility of accounting records. (In fact, it may be argued that without a minimum level of internal control an audit on the fairness of financial statements would not be possible.) The effect on auditing has thus been to reduce the need for routine, mechanical verification of bookkeeping accuracy, permitting substitution of a less time consuming approach that involves reasoning and judgment and stresses such activities as review, analysis, evaluation, and statistical sampling."[9]

Obviously, if the auditor limits the scope of his examination based on the reliability of his client's internal control system, he must have sufficient basis for formulating an opinion on the effectiveness of the system. This implies he should be aware of some basic characteristics of good internal control.

Numerous methods and techniques exist for achieving good internal control. SAS1 indicates that the methods should minimally include the following characteristics:

(a) a plan of organization which provides appropriate segregation of functional responsibilities;
(b) a system of authorization and record procedures adequate to provide reasonable accounting control over assets, liabilities, revenues, and expenses;
(c) sound practices to be followed in performance of duties and functions of each of the organizational departments;
(d) personnel of a quality commensurate with responsibilities.

The first characteristic addresses the concept of division of duties. That is, no one department (or person) should be responsible for handling all phases of a transaction. Another way of looking at this characteristic is that no department (or person) should control the accounting records relating to its own operation.

The second characteristic concerns checks and proofs of accuracy and authorization. Although not explicitly stated, this characteristic also involves procedures for error checking and correction. That is, "reasonable accounting control" must involve procedures for investigating and correcting errors when they occur.

Characteristics three and four are requirements of any efficient organization and are redundant when applied to internal control within an organization. However, the "reliability" of a specific internal control system cannot be assessed without contextual evaluation of these characteristics.

More specifically, one must adopt a perspective for examining an internal control system which highlights the basic characteristics mentioned. Such a perspective is presented in the next section.

INTERNAL CONTROL PERSPECTIVE

The most prudent classificatory technique for identifying internal control is to distinguish between *characteristics that constitute controls* and *activities subject to control*.[10] This classification scheme facilitates use of the traditional control matrix for logical analysis of these systems. Figure 2 illustrates this perspective. The internal control primitives to be presented later can be thought of as activities subject to control. Control characteristics are subdivided into three types: preventive, detective, and corrective. Within each of these types, detail features may be encoded. Every public accounting firm has a list of features it considers prudent for their objectives. Therefore, we do not provide an exhaustive list. The fact that TICOM addresses this issue is

		Characteristics That Constitute Controls		
		Preventive	Detective	Corrective
Activities Subject to Controls	INITIALIZE			
	CREATE OBJECT			
	RECORD			
	PROOF			
	CONTROL			
	COMPARE			
	TRANSFER			
	AUTHORIZE			
	STORE			
	MERGE			
	EJECT			
	HALT			
	SORT			
	UPDATE			
	INTERVAL			

Figure 2—Internal control perspective

the point we want to emphasize. Example detail features are shown in Figure 3.

This perspective can be further expanded by considering different contextual areas within an internal control system. For example, computerized phases of the system have been characterized in terms of three areas:[11]

(a) *Application controls*—unique to individual subprocesses of this system
(b) *Information Processing Facility (IPF)*—which affect the computer installation and environment, and how most applications are processed in a facility
(c) *Systems Management controls*—which assure design, implementation and maintenance are performed in a prudent, secure, and systematic manner.

The crux of this matter is the flexibility afforded by the model to allow different public accounting firms to view the system within their own perspective.

PROBLEM STATEMENT

According to Stettler,[9] there are three basic, closely related questions, that provide a basis for the auditor's conclusion on internal control:

(a) What are the purported internal control procedures?
(b) Are those procedures being followed?
(c) How satisfactory are those procedures?

Currently, these questions are addressed by the auditor in obtaining and storing information about the organization, in such a way that provides a comprehensive picture of the organization. Questionnaires and system flow-charts are currently used, and are stored in a "permanent audit file" for the given client. Although such methods are flexible, and widely accepted, we argue that many key issues of internal control are not properly addressed by these techniques because of the sequential and segmented nature of elicitation, storage, and retrieval of the relevant data.

For example, because each of the accounting subsystems (e.g., accounts payables, cash receipts, accounts receivables) is explored individually and often in some sequence, the common points of two subsystems that might condone fraudulent activity could easily be overlooked. Likewise, employees who perform logically different functions, but whose physical work areas permit access to logically unauthorized functions may not be discovered. The classic example of the need for *positional analysis* is illustrated by Mautz and Mini[12] and summarized below:

Suppose we have a questionnaire with the following questions:

(1) Is the handling of customer remittances separated from the recording of such remittances in the cash receipts journal and accounts receivable subsidiary ledger?
(2) Is a pre-listing of mail receipts prepared?
(3) Is this list compared with cash book entries?

Assuming these questions were answered "no," "yes," and "yes" respectively, it is impossible to determine if the described conditions constitute a vulnerable point in the system. If, for example, the person handling and recording customer remittances was the same person working with the pre-list of mail receipts or had access to the pre-list, vulnerability is high.

It is obvious that the questionnaire could be recon-

I. PREVENTIVE

A. Authorization
B. Sequenced Forms
C. Dual Access and Control
D. Rotation of Duties
E. Segregation of Duties
F. Physical Security (Protection Rings, etc.)
G. Turn-around Documents

II. DETECTIVE

A. Edits and Checks (Sequence, Over-flow, Format, etc.)
B. Control Totals (Hush, Batch, etc.)
C. Dating
D. Read-back (Echo)
E. Redundant Processing
F. Suspense and Tickler Files

III. CORRECTIVE

A. Discrepancy Reports
B. Resubmission (Upstream and Reinitialization)
C. Backup and Recovery
D. Automated Error Correction
E. Error Source Statistics

Figure 3—Characteristic control-type examples

structed to expose this weakness; however, as will be illustrated later, the network approach presented herein for describing a system forces the inclusion of *positional analysis* inquiries on a questionnaire.

Another issue worthy of comment centers on "lagged" or compensatory control procedures. These terms refer to control procedures that offset or counteract an action elsewhere in the system; which if evaluated singly would be construed as a weakness in the internal control system. An example would be a cash receipts system in which a cashier received cash payments from customers and was charged with making the accounting entry that reflected receipt of the cash, and the customer was not given a receipt slip of any kind. In such a system the cashier could "pocket" the cash and omit making the accounting entry with no mechanism to provide a signal indicating improper action. Now, if we added to the above description that the customer leaves the cashier and goes to another window to record the cash payment (the customer cannot leave the system without doing this), we have added a control feature which appears after an action that possesses undesirable control characteristics. This type of analysis is facilitated by simulation of the control system. Thus, simulation would seem to be an essential part of a mechanism to evaluate an internal control system.

It is easily deduced that this discussion of issues could continue over numerous pages. The key point is that a model is needed which ameliorates the auditor's review function with respect to internal control systems. This can only be accomplished by adoption of the aforementioned total system approach made possible by recent advances in data management systems and interfaces. Decreasing costs for computer hardware, increased capability of software to handle complex data structures, and increased complexity of constituent parts of control systems (for example program logic) have all contributed to motivate expanded use of computerized information systems. This effort is another which views that continued expansion as imminent.

Stated formally, the objective of this work is to define a facile means for formally describing and objectively evaluating an internal control system. The proposed mechanism should have the capacity to organize and retrieve data about the system in a manner that permits the auditor to objectively address the issues listed earlier. Such a mechanism will facilitate more thorough examination of these systems.

TICOM OVERVIEW

Figure 4 depicts schematically an overview of the proposed system. After initialization, the auditor elicits information about the organization and records this information in the "Internal Control Description Language." The ICDL is submitted to the "Internal Control Description Language Analyzer" (ICDLA) which checks for inconsistencies and other errors in the ICDL. If no problems or errors are discovered by the ICDLA then the data is loaded onto the "Internal Control Description Data Base" ICDDB. Figure 5 depicts a simplified version of ICDDB. At this point the auditor is able to make queries of the data base. Each of these steps in the TICOM cycle is described in the following sections.

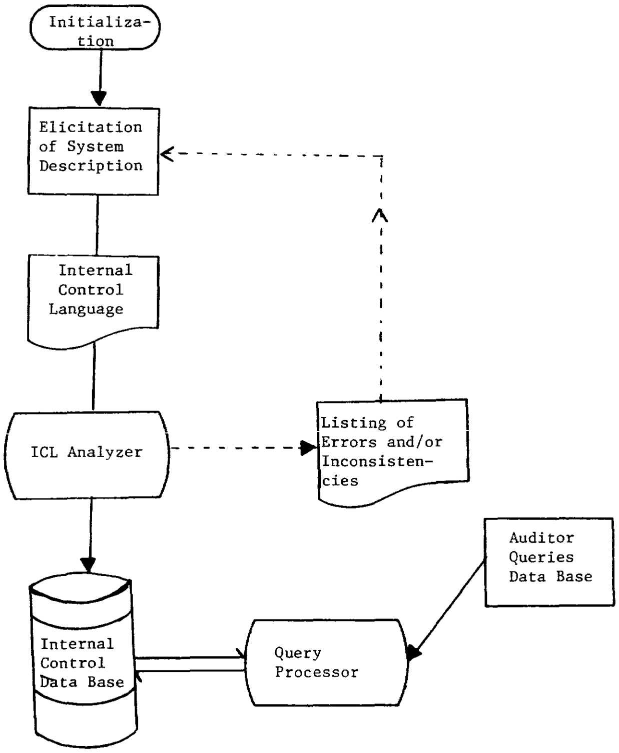

Initialization
Elicitation of System Description
Internal Control Language
ICL Analyzer
Listing of Errors and/or Inconsistencies
Auditor Queries Data Base
Internal Control Data Base
Query Processor

Figure 4—TICOM schematic

While examining the material that follows, remember that our main purpose in developing this model was to provide the auditor with aids for more objective evaluation of internal control than currently available to him.

Initialization

The first step in the TICOM cycle is setting up the static and literal information in the internal control system. We

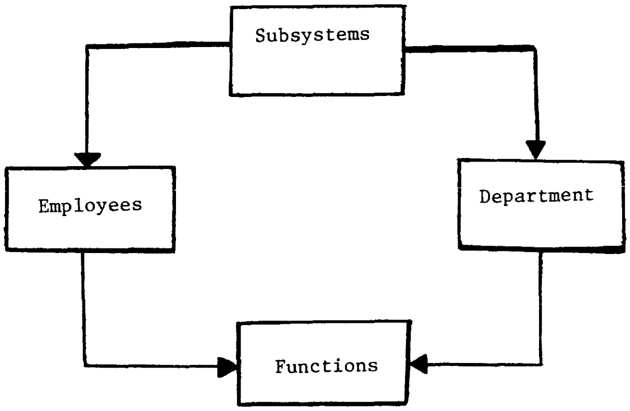
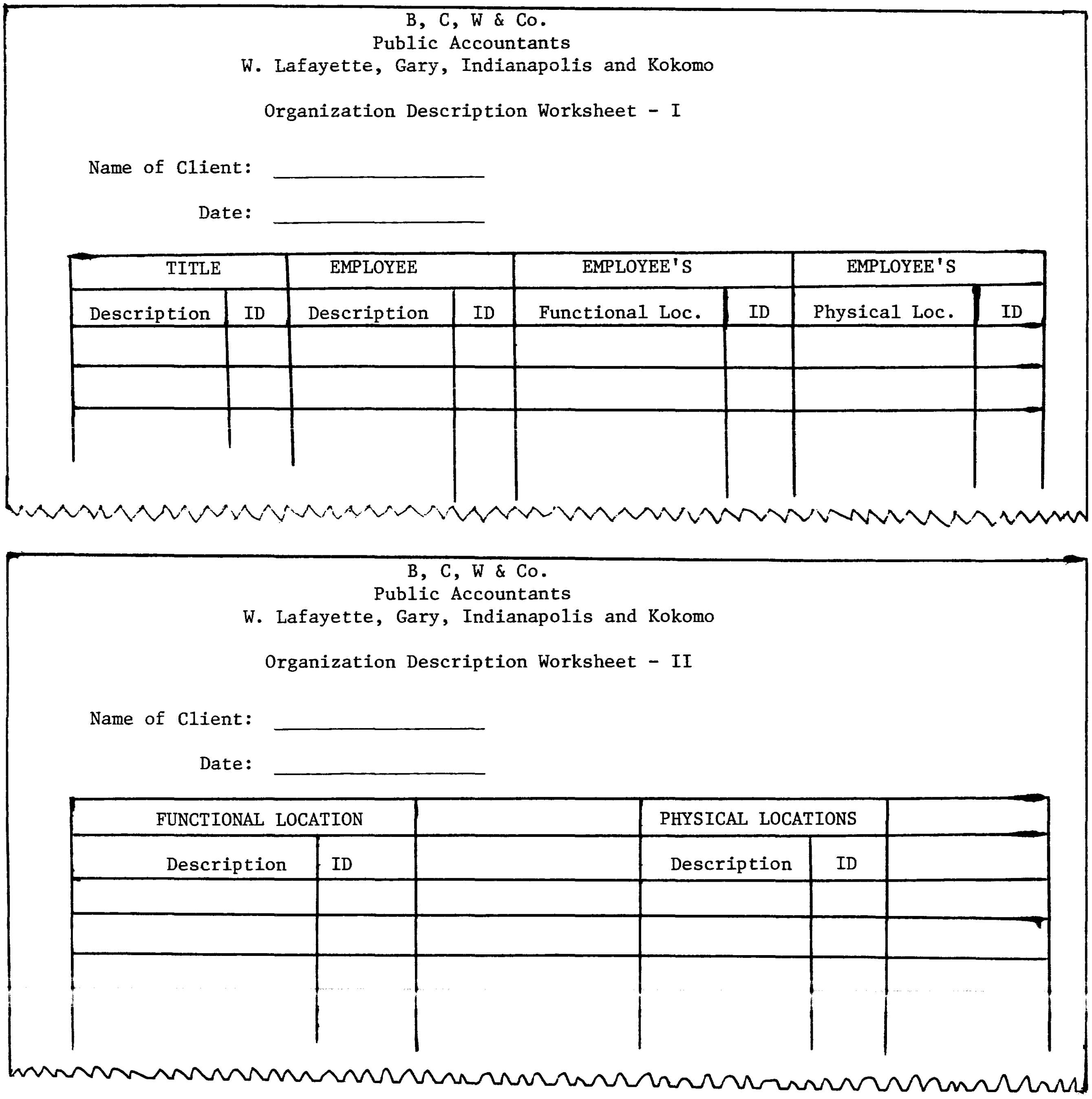


Figure 5—Simplified schematic of data base organization



B, C, W & Co.
Public Accountants
W. Lafayette, Gary, Indianapolis and Kokomo

Organization Description Worksheet - I

Name of Client: ________________

Date: ________________

TITLE		EMPLOYEE		EMPLOYEE'S		EMPLOYEE'S	
Description	ID	Description	ID	Functional Loc.	ID	Physical Loc.	ID

B, C, W & Co.
Public Accountants
W. Lafayette, Gary, Indianapolis and Kokomo

Organization Description Worksheet - II

Name of Client: ________________

Date: ________________

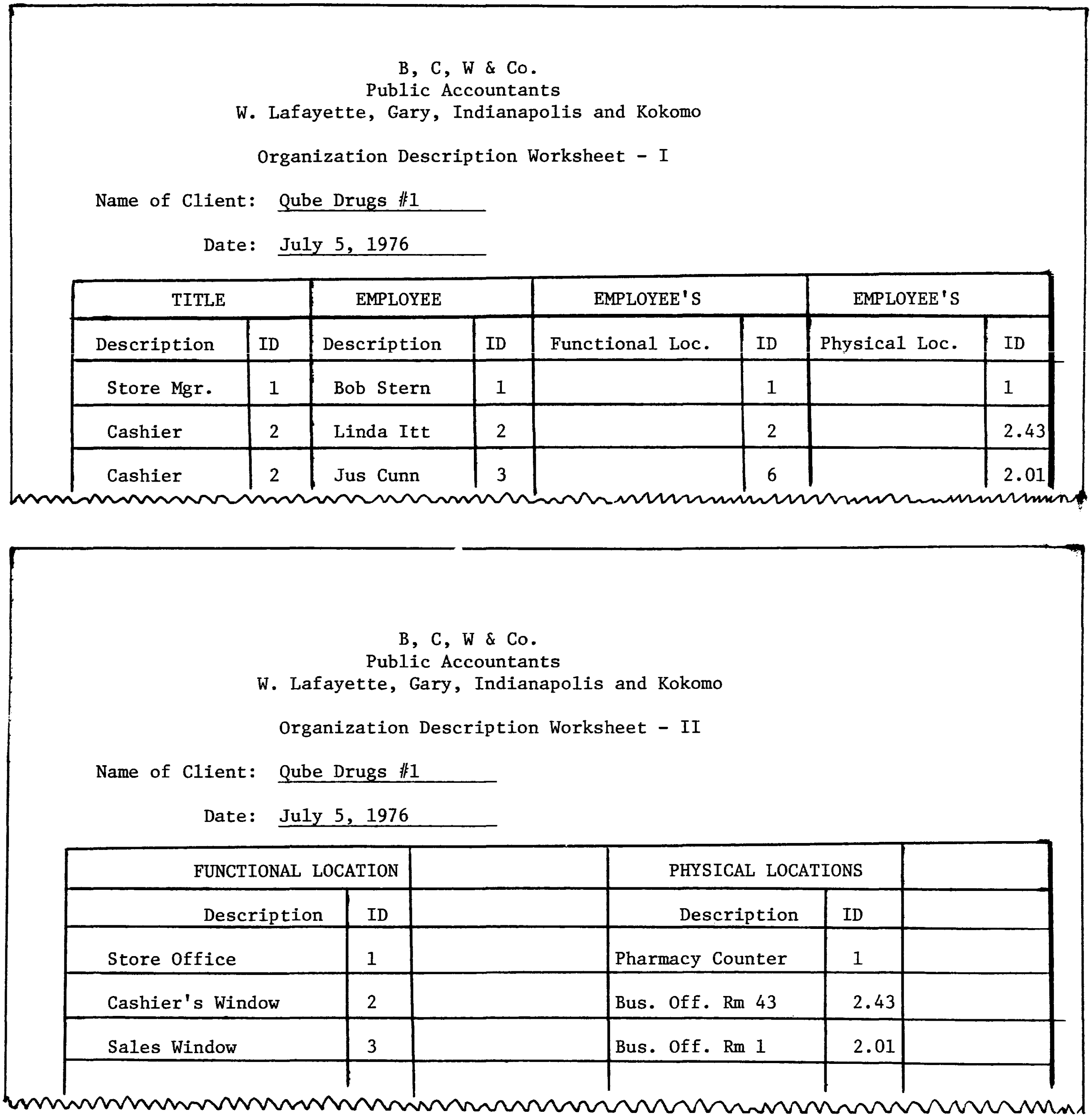
FUNCTIONAL LOCATION			PHYSICAL LOCATIONS		
Description	ID		Description	ID	

Figure 6—Forms for recording static and literal data

will explain the use of this information later. For now we provide an outline of the order of events in this step, list the information required at this point, give a short description of its use, and illustrate in Figures 6 and 7 forms for recording the data. (It should be noted that the information requested in this step does not require the physical presence of an auditor. This data can be gathered and submitted via mail carrier before the elicitation step):

I. Request Allocation of EDP Equipment (Client's or Firm's)

   A. Auxiliary storage allocation
   B. CPU time allocation



B, C, W & Co.
Public Accountants
W. Lafayette, Gary, Indianapolis and Kokomo

Organization Description Worksheet - I

Name of Client: Qube Drugs #1

Date: July 5, 1976

TITLE		EMPLOYEE		EMPLOYEE'S		EMPLOYEE'S	
Description	ID	Description	ID	Functional Loc.	ID	Physical Loc.	ID
Store Mgr.	1	Bob Stern	1		1		1
Cashier	2	Linda Itt	2		2		2.43
Cashier	2	Jus Cunn	3		6		2.01

B, C, W & Co.
Public Accountants
W. Lafayette, Gary, Indianapolis and Kokomo

Organization Description Worksheet - II

Name of Client: Qube Drugs #1

Date: July 5, 1976

FUNCTIONAL LOCATION			PHYSICAL LOCATIONS		
Description	ID		Description	ID	
Store Office	1		Pharmacy Counter	1	
Cashier's Window	2		Bus. Off. Rm 43	2.43	
Sales Window	3		Bus. Off. Rm 1	2.01	

Figure 7—Sample completed forms for recording static and literal data

C. Programs for TICOM
   1. ICDLA
   2. Query Processor
   3. etc.

II. Obtain Static and Literal Data

A. Use forms similar to those in Figures 6 and 7 to collect this data.

B. Data can be loaded as a part of hardware initialization for this client.

C. Required Information (Example)
   1. *TITLE*—names of positions similar to those found on company organization charts.
   2. *EMPLOYEE*—names of persons who the relevant position. Note that if two employees have the same job title but different authorization responsibilities we should have no problem

differentiating between the two via the employee identification code.

3. *FUNCTIONAL LOCATION*—to facilitate the previously mentioned positional analysis.
4. *PHYSICAL LOCATION*—for those firms that segregate employees by logical accounting functions.
5. *IDENTIFICATION*—from the standpoint of implementation, it may be desirable to index or tag this information rather than use literal descriptions.

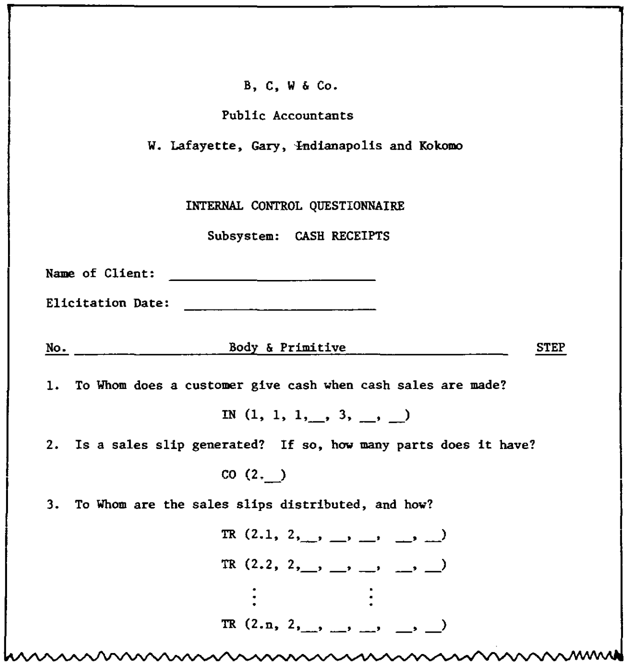
Elicitation procedure

There are two approaches one could take for evoking a description of the internal control system: *questionnaire* and *free-form*.

Free-form is equivalent to the process a programmer follows in writing code. The first step is to outline the logic for the algorithm to be coded; and next to specify instructions in the given language which corresponds to that logic. The analogue of that procedure is an auditor developing a mental image of the internal control system and transposing that image into the descriptive language presented later. Proponents of this approach argue that a descriptive language such as the one we propose which uses terms familiar to the auditor will require no more technical proficiency than the use of flow-charts when they were first introduced.

Proponents of the questionnaire approach argue if the proposed model is to possess attributes which imply current applicability and practicality, the elicitation procedure should be kept as close as possible to current practice while capturing the information required for the data base. We have chosen to use the questionnaire approach for illustration of primitives. This decision was not meant to infer that the questionnaire approach is superior, but rather more illustrative of basic concepts for this model. Figure 8 illustrates a sample elicitation form and serves as an introduction to primitives of ICDL.

The first thing to note about the questionnaire is the ~~elimination of "slanted" questions. That is, the traditional~~ wording of a questionnaire such that "NO" answers suggest undesirable practice relative to achieving good internal control. Behavioral implications of such questionnaires are not in the scope of this work but are definitely noteworthy. Our questionnaire elicits description without involved evaluation thus separating the two activities.



```
                     B, C, W & Co.

                  Public Accountants

        W. Lafayette, Gary, Indianapolis and Kokomo


              INTERNAL CONTROL QUESTIONNAIRE

                Subsystem:  CASH RECEIPTS

Name of Client:  ____________________

Elicitation Date:  ___________________

No. ____________Body & Primitive_______________   STEP

1.  To Whom does a customer give cash when cash sales are made?

                IN (1, 1, 1,__, 3, __, __)

2.  Is a sales slip generated?  If so, how many parts does it have?

                CO (2.__)

3.  To Whom are the sales slips distributed, and how?

                TR (2.1, 2,__, __, __,  __, __)

                TR (2.2, 2,__, __, __,  __, __)

                 :          :

                TR (2.n, 2,__, __, __,  __, __)
```

Figure 8—Example questionnaire

The function listed below each question is the ICDL primitive that corresponds to the question being asked. Note that the auditor should not have to reproduce these primitives since they are preprinted on the form and *imply the initial level of detail* required by the auditing firm. The auditor is required to fill in the arguments of the primitives based on the response made and a list of coded alternatives. Figures 9 and 10 list ICDL primitives and sample argument alternatives. We do not propose that the given primitives are an exclusive or totally comprehensive set, but that they serve to illustrate the desirable level of detail and flexibility of this model. Clearly, each public accounting firm may view the organization as based on different sets of primitives, and that should not cause any problem with the application of this model. The key issue is storage of a machine readable version of the system, at arbitrary levels of detail, for more thorough investigation of its possible strengths and weaknesses.

The column labeled "STEP" is used to form an index for the sequence of execution and level of detail of the relevant primitives.

Once the questionnaire has been completed, the auditor would submit it to the firm's (or the client's) EDP personnel for conversion into machine readable form and input to the ICDLA. Example execution of this next step in the TICOM cycle is given in the next section.

Generation of internal control description language

The primitives that appear on the questionnaire (or free-form, if that approach is chosen) next serve as input to the ICDLA. This step is performed by keying responses into machine readable form. A special header record would probably be used to delimit subsystems. An example of the information that might be submitted, which corresponds to each ICDL statement, is:

(a) Primitive name
(b) Primitive arguments
(c) Step
(d) Related employee or title id's

$$\text{Example:}\quad \underbrace{\text{IN}}_{(a)}\quad \underbrace{(1,\ 1.1,\ 1,\ 2.1,\ 3,\ 0,\ 0)}_{(b)}\quad \underbrace{1}_{(c)}\quad \underbrace{14,\ 351,\ 41}_{(d)}$$

The input format of this data would be straightforward depending only on the conventions of the coder. Note that functional information is not included, which is a result of the questionnaire approach that uses preconceived data.

INITIALIZE (STIMULI-TYPE, CONTENT, SOURCE, DESTINATION, CARRIER CONTROL-ATTRIBUTE, ATTRIBUTE-DETAIL)

IN(a_1, a_2, a_3, a_4, a_{14}, a_{15}) An action or event that causes execution of a subsystem (e.g., receipt of a payment on account via the mail invokes the cash receipts subsystem)

CREATE OBJECT (LITERAL-DESCRIPTION, CONTROL-ATTRIBUTE, ATTRIBUTE-DETAIL)

CO(a_5, a_{14}, a_{15}) The generation of paper as a result of, or to record asset flow (e.g., the generation of a receipt to record acceptance of cash)

RECORD (LITERAL-DESCRIPTION, SOURCE, DESTINATION, CONTROL-ATTRIBUTE, ATTRIBUTE-DETAIL)

RD(a_5, a_3, a_4, a_{14}, a_{15}) The documentation of an asset (e.g., recording cash and checks in a cash receipts book or accounts receivable (control) account)

PROOF (LITERAL-DESCRIPTION, MECHANICAL-DEVICE, CONTENT, CONTROL-DOCUMENT, NEXT-STEP, CONTROL-ATTRIBUTE, ATTRIBUTE-DETAIL)

PR(a_5, a_8, a_2, a_7, a_{12}, a_{14}, a_{15}) A control or check point at which a mechanical device generates a control item (e.g., a cash register's receipts total)

CONTROL (LITERAL-DESCRIPTION, CONTROL-DOCUMENT, CONTROL-DOCUMENT, NEXT-STEP, CONTROL-ATTRIBUTE, ATTRIBUTE-DETAIL)

CT(a_5, a_7, a_7, a_{12}, a_{14}, a_{15}) A check for offsetting paperwork entries

COMPARE (LITERAL-DESCRIPTION, MECHANICAL-DEVICE, MECHANICAL-DEVICE, NEXT-STEP, CONTROL-ATTRIBUTE, ATTRIBUTE-DETAIL)

CM(a_5, a_8, a_8, a_{12}, a_{14}, a_{15}) A completely mechanical control point or check (e.g., the hardwired checks between two cash registers)

TRANSFER (LITERAL-DESCRIPTION, CONTENT, SOURCE, DESTINATION, CARRIER, CONTROL-ATTRIBUTE, ATTRIBUTE-DETAIL)

TR(a_5, a_2, a_3, a_4, a_6, a_{14}, a_{15}) Physical movement of an asset or paperwork (e.g., movement of checks from the mailroom to the cashier's office)

Figure 9—ICDL primitives

AUTHORIZE (LITERAL-DESCRIPTION, CONTENT, AUTHORIZATION-METHOD, AUTHORIZER, CONTROL-ATTRIBUTE, ATTRIBUTE-DETAIL)

AR(a_5, a_2, a_9, a_{10}, a_{14}, a_{15}) This primitive encodes the points of authorization in the internal control system

STORE (LITERAL-DESCRIPTION, CONTENT, SOURCE, DESTINATION, CARRIER, CONTROL-ATTRIBUTE, ATTRIBUTE-DETAIL)

ST(a_5, a_2, a_3, a_4, a_6, a_{14}, a_{15}) The transfer of an item to a quasi-final or normal place of residence (e.g., cash deposited in a bank)

MERGE (LITERAL-DESCRIPTION, LITERAL-DESCRIPTION, LITERAL-DESCRIPTION, CONTENT, CONTROL-ATTRIBUTE, ATTRIBUTE-DETAIL)

MG(a_5, a_5, a_5, a_2, a_{14}, a_{15}) A description of the merger of two items into one. This combination will continue through the system as one item (e.g., the combination of day's receipts of cash and checks into cash receipts)

EJECT (LITERAL-DESCRIPTION, CONTENT, SOURCE, DESTINATION, CARRIER, CONTROL-ATTRIBUTE, ATTRIBUTE-DETAIL)

EJ(a_5, a_2, a_3, a_4, a_6, a_{14}, a_{15}) The emission of items out of the internal control system (e.g., finished goods that were sold and carried away by a customer)

SORT (LITERAL-DESCRIPTION, CONTENT, CONTROL-ATTRIBUTE, ATTRIBUTE-DETAIL)

ST(a_5, a_2, a_{14}, a_{15}) A description of item rearrangement

UPDATE (LITERAL-DESCRIPTION, LITERAL-DESCRIPTION, OPERATOR, RESULT, CONTROL-ATTRIBUTE, ATTRIBUTE-DETAIL)

UP(a_5, a_5, a_{16}, a_{14}, a_{15}) Item modification, such as accumulation of data values or information

INTERVAL (PERIOD, CONTROL-ATTRIBUTE, ATTRIBUTE-DETAIL)

IV(a_{17}, a_{14}, a_{15}) Primitive to facilitate time intervals between adjoining primitives

HALT (SUBSYSTEM)

HT(a_{13}) The end of primitives related to this subsystem

Figure 9 (continued)

Free-form would require encoding of this data. In the example above, the INITIALIZE primitive is encoded. Referring to Figures 9 and 10 we are able to decode the primitive (b) as follows:

1. The first parameter, 1, corresponds to a_1, of the IN specification given in 9. As shown, it specifies the STIMULI-TYPE. Examination of 10 reveals that 1

a_0 ≡ EMPLOYEE: = (1 − TITLE)|(2 = NAMES)
a_1 ≡ STIMULI-TYPE: = (1 = EXTERNAL)|(2 = INTERNAL)
a_2 ≡ CONTENT: = (1 = ASSET {1 = CASH, 2 = INVENTORY, . . . ,
n = EQUIPMENT})|(2 = PAPERWORK REPRESENTATIVE OF
AN ASSET)|
⋮
(n = DOCUMENTATION)
a_3 ≡ SOURCE: = (1 = CUSTOMER)|(2 = FUNCTIONAL AREA {1 = FINISHED
GOODS AND SERVICE, 2 = MAIL ROOM, 3 = BILLING
4 = GENERAL LEDGER, . . . , n = })|
(3 = TITLE {1 = PRESIDENT, 2 = CASHIER, . . . , n = })|
(4 = CASH REGISTER RECEIPT)|(5 = EXTERNAL TO SYSTEM)|
(6 = SALES SLIP)|(7 = SALES JOURNAL)|
(8 = CASH SALES CLEARING ACCOUNT)|. . .|(n = BANK)
a_4 ≡ DESTINATION: = (SOURCE)
a_5 ≡ LITERAL-DESCRIPTION: = (1 = CASH)|(2 = SALES SLIP {1 = COPY 1
2 = COPY 2, . . . n = COPY n})|(3 = CASH RECEIPT|
(4 = RELEASE OF INVENTORY)|(5 = FINISHED GOODS)|
(6 = PRICES)|(7 = # SEQUENCE)|(8 = RECEIPTS)
a_6 ≡ CARRIER: = (1 = MAIL)|(2 = MECHANICAL DEVICE)|(3 = CUSTOMER)
(4 = EMPLOYEE)|. . .|(n = UPS)
a_7 ≡ CONTROL-DOCUMENT: = (1 = CASH REGISTER RECEIPT)|(2 = DEPOSIT
SLIP)|(3 = SALES SLIP)|(4 = PRICE SHEET)|
(5 = NUMERICAL SEQUENCE)|(6 = SALES JOURNAL)|. . .
(n =)
a_8 ≡ MECHANICAL-DEVICE: = (1 = CASH REGISTER)|. . .|(n =)
a_9 ≡ AUTHORIZATION-METHOD: = (1 = SIGNATURE)|(2 = RUBBER STAMP)|. . .|
(n = REPROCESSING)
a_{10} ≡ AUTHORIZER: = (1 = EMPLOYEE)|(2 = MECHANICAL DEVICE)
a_{11} ≡ CHECK: = (1 = PROOF)|(2 = CONTROL)|(3 = COMPARES)|. . .|
(n = AUTHORIZATION)
a_{12} ≡ NEXT STEP: = (NUMBER)
a_{13} ≡ SUBSYSTEM: = (1 = A/P)|(2 = A/R)|(3 = C/R)|. . .|(n = C/D)
a_{14} ≡ CONTROL-ATTRIBUTE: = (0 = NOT APPLICABLE)|(1 = PREVENTIVE)|
(2 = DETECTIVE)|. . .|(n = CORRECTIVE)
a_{15} ≡ ATTRIBUTE DETAIL: = (0 = NOT APPLICABLE)|(1 = SEGREGATION
OF DUTIES)|(2 = ROTATION OF DUTIES)|. . .|
(n = DESCREPANCY REPORTS)
a_{16} ≡ OPERATOR: = (1 = ARITHMETIC SYMBOL {1 = *, 2 = +, . . . , n = **})|
. . .|(n = SYSTEM PROCESS)
a_{17} ≡ PERIOD: = (1 = MONTH)|(2 = DAY)|(3 = YEAR)|. . .|(n = MINUTES)

Figure 10—Arguments for ICDL primitives

denotes EXTERNAL stimuli were responsible for initialization of this subsystem. An example of this type of stimulus is a customer making a payment on account and invoking the accounts receivable system. This is contrasted with an employee submitting a time card and initiating the payroll system; which we term INTERNAL stimulus.

2. The second parameter indicates the CONTENT parameter denotes ASSET, with CASH being the particular type of asset.
3. The SOURCE parameter denotes CUSTOMER.
4. DESTINATION is the FUNCTIONAL AREA described as FINISHED GOODS AND SERVICE.
5. The CARRIER in this example is CUSTOMER.
6. The CONTROL-ATTRIBUTE is not applicable in this instance.
7. As a result of 6, ATTRIBUTE-DETAIL is not applicable.

Components (c) and (d) of the example specify sequence and related employees. The "step" parameter defines this ICDL statement to be the first for the given subsystem. The employee numbers (id) identify employees related to the statement that are not explicitly included via the EMPLOYEE-FUNCTION-FUNCTION LOCATION relationship.

This example facilitated the introduction of several issues concerning the language. First, we are able to differentiate between asset and paperflow movement through the system. Second, the high degree of parameterization in this system facilitates the extreme flexibility and adaptability we eluded to in earlier sections. Next, the encoding of control objectives via parameters a_{14} and a_{15}, allows the auditor to explicitly identify and label these features (this point will be discussed further below). The indexing illustrated, such as 1.1 and 2.1 for a_2 and a_4 respectively, facilitates the aforementioned "levels of detail" when applied to STEP.

Internal control description language analyzer

This process is the main link between the auditor and a description of the given internal control system stored in a network data base structure. The inherent tasks of any analyzer are also in evidence here. That is, the logical analysis of a model of any information system includes determination of *consistency* and *completeness* of the model. Consistency implies the system follows a design determined by propositional statements of definitions and relations. When these statements are applied to the model, inconsistencies are detected. Completeness is a consistency concern which involves the unique resolution of given relations. We will omit detailed explanation of this process since there exists a substantial body of literature on this topic. More pertinent are its tasks related to generating the target data base (Figure 11). Omitting the obvious tasks of any analyzer, we assume it sufficient to say appropriate diagnostics would be printed.

The major task of the analyzer is the generalized load function it must perform. Figure 12 schematically outlines this procedure. Each subsystem will be denoted by the aforementioned header record. When a new subsystem is encountered the analyzer first checks that the subsystem just finished ended with the proper primitives (HALT). It then creates (or allocates) a record occurrence corresponding to the record type that denotes "SUBSYSTEM." The next ICDL statement is read and its type is determined. If it is a primitive, the associated record type is created, linked to the other primitives, relevant data items stored, and finally, the next statement is read. If it was a HALT statement, subsystem "housekeeping" is performed, or the next statement is read.

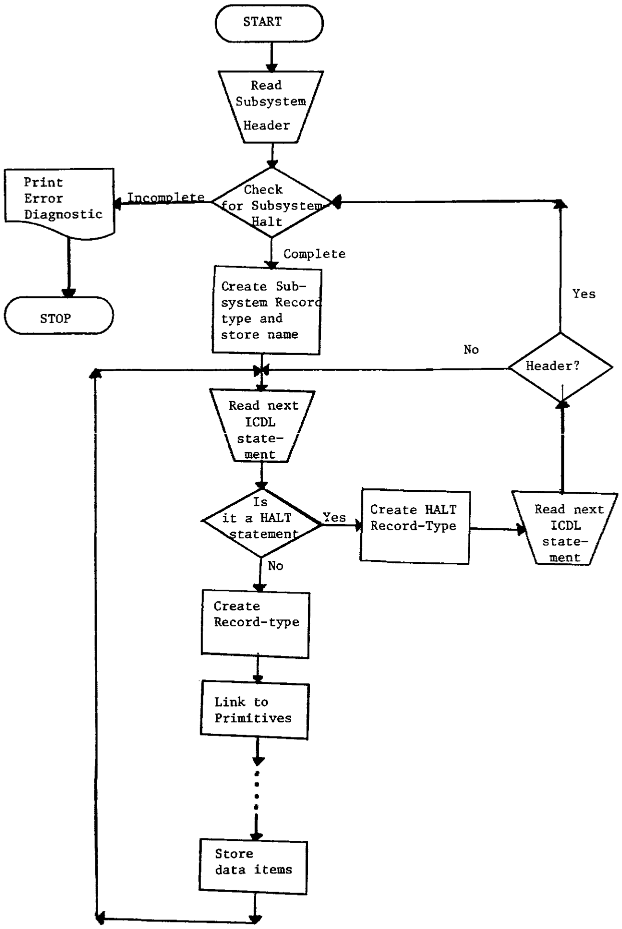


Figure 12—ICLDA schematic

Internal control description data base

Figure 13 schematically depicts a schema which would support query access to such a system. In this organization the primitive-type would be encoded in the record occurrence with the other static information given as shown. Details of this structure are the topic of another paper.

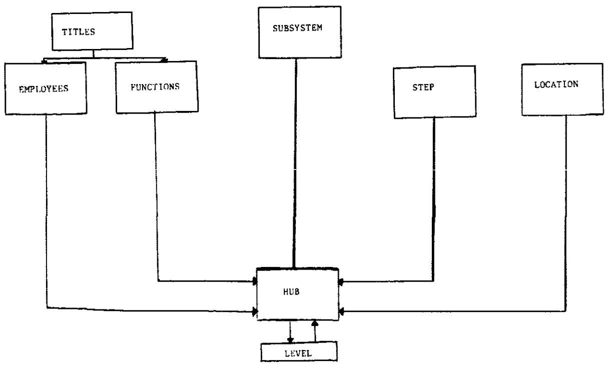


Figure 11—General target data base for ICDDB

However, we will provide a few comments to make the structure comprehensible.

The LINK records that are shown in the figure are included to facilitate many-to-many relationships between the adjoined entities. This is a restriction imposed by the network data base structure we are using. For example, since a given function such as dating incoming mail may be encoded as a step in numerous subsystems by several employees at different locations, it is impossible to establish a one-to-many relationship between FUNCTION and LOCATION as dictated by the network data base implementation requirements. Therefore an intermediate record is introduced to facilitate this relationship.

The HUB record denotes the occurrence of primitives in this structure which highlight information and data flow through the internal control system. The relationship between this record-type and LEVEL facilitates the different levels of detail that can be stored via this structure.

LINK-3 is included specifically to provide quick retrieval of employee names and their functional and physical locations.

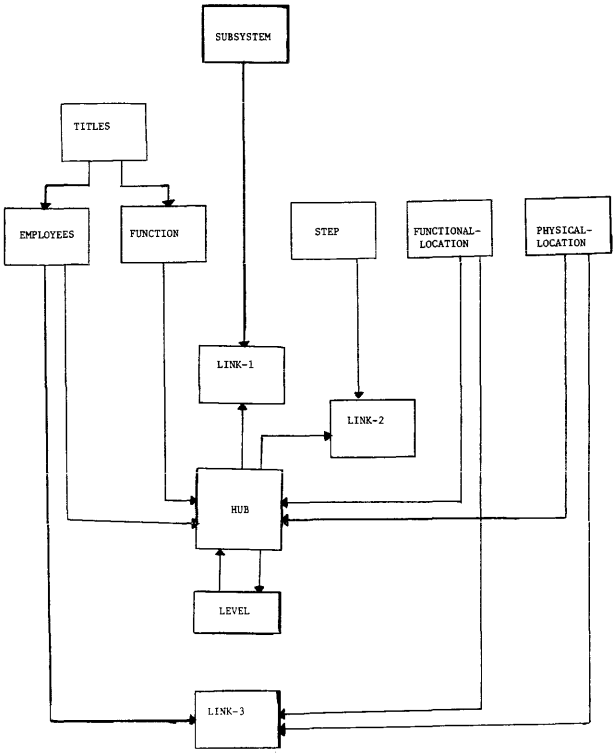


Figure 13—General schema for ICDDS

Query processor

The proposed system could be implemented by the GPLAN [Bonczek, et al., 1975] data management facility. In this case the associated query processor would be used by the auditor to access the internal control description stored in the data base. The GPLAN query language [13,14] and automatic path determinator[15] would extricate the auditor from developing interfacing programs with the data base. That is, the auditor is not required to be a programmer to access this data base. Sample queries that the auditor might submit are:

(a) SIMULATE SUBSYSTEM = 'CASH SALES,' LISTING ALL EMPLOYEES FOR PRIMITIVE = 'AUTHORIZE'
(b) LIST THE NEXT PRIMITIVE FOLLOWING PRIMITIVE = 'TRANSFER' FOR CONTENT = 'ASSET'
(c) LIST COMMON EMPLOYEES FOR SUBSYSTEM = 'CASH SALES' AND SUBSYSTEM = 'CASH DISBURSEMENTS'
(d) LIST ALL SUBSYSTEMS FOR EMPLOYEE = 'JOHN GREEN'
(e) FOR PRIMITIVE = 'AUTHORIZE' LIST ALL OCCURRENCES

Upon receipt of a command the query system analyzes the request, sets up the necessary DML commands, executes those commands, and provides the user with the requested information.

EXAMPLE FREE-FORM ENCODING

Before providing detail development of the ICDDB structure, an example of primitive encoding is presented. In this instance we will take a well-known description of an accounting subsystem and present that system as it may have been encoded using the free-form approach.

Stettler[9,pp.274–275] and the AICPA depict accounting subsystems in procedural flowchart form. One of the subsystems they depict is CASH-RECEIPTS. We first provide a literal description of the initial phase of that system and list the corresponding primitive description in Figure 14. Note that only primitives and STEP have been listed; other static information would not aid in this presentation and have been omitted.

> Customer initializes the CASH RECEIPTS system by making a cash purchase. A *sales slip (2-part)* is prepared by a *customer service* employee in the *Finished Goods and Service* Department. One part of the sales slip is transferred to the *Billing* Department of the *Controller's* office. The other part is carried by the *customer* to the cashier's window. The cashier accepts the customer's cash and keys relevant information into a *cash register*. The *cash register* generates a *receipt* that is given to the *customer*. The customer returns to the *customer service* area and presents the *receipt*. The relevant goods are given to the *customer* and they both *exit* the system.

SUMMARY

This paper was designed to provide a general overview and justification for the use of a network data base to facilitate the auditor's review and evaluation of internal control systems. It is the third in a series of four papers which present a new methodology for auditing of advanced EDP based accounting information systems. A facile means for

STEP	PRIMITIVES & ARGUMENTS	
1	IN	(1, 1.1, 1, 3.2, 3, 0, 0)
2	CO	(2.2, 2, 10)
3	TR	(2.1, 2, 2.1, 2.3, 4, 2, 1)
3	TR	(2.2, 2, 2.1, 3.2, 3, 1, 1)
3	TR	(1, 1.1, 1, 3.2, 3, 0, 0)
4	RD	(1, 1, 4, 2, 9)
5	CO	(3, 1, 9)
6	TR	(3, 2, 3.2, 1, 4, 1, 1)
7	TR	(3, 2, 1, 2.1, 3)
8	AR	(4, 1.2, 3, 1, 1, 1)
9	EJ	(5, 1.2, 2.1, 3, 0, 0)
10	HT	(3)

Figure 14—Free form encoding of literal description

formally describing and objectively evaluating an internal control system was presented. The mechanism was assented to have the capacity to organize and retrieve data about the system in a manner that permits auditors to objectively address issues listed in the INTERNAL CONTROL OVERVIEW section of this paper.

The next and last paper of this series will provide technical details of the TICOM model.

REFERENCES

1. Cash, J. I., A. D. Bailey, and A. B. Whinston, "A Survey of Techniques for Auditing EDP Based Accounting Information Systems," *The Accounting Review*, October 1977.
2. Brown, R. G., "Changing Audit Objectives," *The Accounting Review*, October, 1962.
3. Fitzpatrick, L., "The Story of Bookkeeping Accounting and Auditing," *Accountants Digest IV*, March, 1959.
4. AICPA, "Codification of Auditing Standards and Procedures," *Committee on Auditing Procedure*, November, 1973.
5. Bodnar, G., "Reliability Modeling of Internal Control Systems," *The Accounting Review*, October, 1975.
6. Ishikawa, A., "Feedforward Control in the Planning and Control System," *Cost and Management*, November/December, 1972.
7. Yu, S. and J. Neter, "A Stochastic Model of Internal Control System," *Journal of Accounting Research*, Autumn, 1973.
8. Cushing, B. E., "A Mathematical Approach to Analysis and Design of Internal Control Systems," *Accounting Review*, January, 1974.
9. Settler, H. F., *Systems Based Independent Audits*, Prentice Hall, 1974.
10. Touche Ross & Co., *Computer Controls and Audit*, National Accounting and Auditing Staff, September, 1974.
11. Touche Ross & Co., *Computer Fundamentals*, Professional Continuing Education Series, June 1975.
12. Mautz, R. K. and D. L. Mini, "Internal Control Evaluation and Audit Program Modification," *The Accounting Review*, April, 1966.
13. Bonczek, R. H., J. I. Cash, W. D. Haseman, C. Holsapple, and A. B. Winston, "Generalized Planning System/Data Management System (GPLAN/DMS), Users Manual," Krannert Graduate School of Management, August, 1975.
14. ——, "Structure of a Query Language for a Network Data Base," Technical Report, Krannert Graduate School of Management, August, 1975.
15. Bonczek, R. H., W. D. Haseman, and A. B. Whinston, "Automatic Path Determination in a Network Data Base," Technical Report, Krannert Graduate School of Management, April, 1976.
16. Cash, J. I., A. D. Bailey and A. B. Whinston, "System Design Methodology For Formal System Assertions," Krannert Graduate School of Management, July, 1976.

Design and implementation of an information base for decision makers*

by R. H. BONCZEK, C. W. HOLSAPPLE and A. B. WHINSTON
Purdue University
West Lafayette, Indiana

ABSTRACT

When considering the design and implementation of systems for decision support, a crucial point is the power and flexibility of available tools for representing data contexts. The value of such systems is constrained by the "richness" of patterning allowed by their data structure mechanisms. We introduce the notion of an information base as a natural step forward in the continuing evolution of data structures. The outstanding features of the information base are (1) its accommodation of the horizontal and vertical integration of information parcels into a single semantic mechanism, and (2) the integration of operators into this semantic structure.

INTRODUCTION

A topical and decidedly significant area of research involves the identification of those criteria which a computerized information system must satisfy if it is to be of value to non-programming decision makers. The ensuing discussion focuses upon such criteria and their implications for system design and implementation. In particular, we introduce the notion of an *information base* and demonstrate how it may be developed and implemented as an extension to the CODASYL DBTG[1] approach to data management. We commence with the characterization of an information base as a semantic network. It is then shown that this semantic network may be realized as an extension to an approach that has been used in commercial environments. Moreover, we illustrate how the information base serves as the cornerstone for a generalized decision support system.

Within the scope of this paper a distinction is drawn between the terms information and data. Observe, first of all, that information is an abstraction; it is not something which can be pointed to or seen. However it may be conveyed by patterns of "matter-energy,"[2] i.e., by configurations of symbols, by data. Data and information invariably accompany one another. The words on this piece of paper are not information, but rather a pattern of matter-energy which as a consequence of certain activities (e.g., inputing, transmitting, decoding, associating, storing, deciding, etc.) conveys information.[3] The important point is the patterning of data; the "richness" of a notation in terms of the kinds of data relationships which it can represent has obvious implications for its power in conveying information. With this in mind, we can note a pronounced trend in the history of information systems from the relatively impoverished linear data structure to the tree and network data structures, capable of a greater variety of data configuration; correspondingly the ease with which comparatively complex information can be conveyed has also grown. Summarizing, ". . . we can say that data is an objective notation which has no significance in itself, versus information as a subjective concept which relates a datum to a context."[4]

In order to understand the varieties of contexts or configurations in which data must appear if there is to be a comprehensive conveyance of information, we examine the field of semantics. Of special interest is the notion of a semantic net. The results of this examination constitute a basis for the specification of *information base* features which permit the unambiguous representation of all types of information pertinent to decision support applications. This representation must configure data such that all significant ~~relationships among parcels of information (e.g., among~~ facts, procedures, empirical information, etc.) are accommodated. Furthermore, these objectives for information base features must be met in a manner that is amenable to processing for the purposes of inference and deduction.

Since semantics deals with the *relationships* between symbols and what they denote or mean,[5] what we call the information base may be viewed as a semantic mechanism capable of representing meanings in terms of data configurations. Its storage technique must be general enough to handle the basic kinds of information involved in decision making regardless of the specific decision application. These types of information are: directive information, conceptual information, empirical information, stimulatory information, information about expectations, information concerning valuations, and procedural information. In addition the information base must be flexible enough to represent the often intricate interrelationships among information parcels, relating them so as to capture their full meaning

* Research supported by Office of Water Research and Technology Grant No. 6538-62-1310.

and impact with respect to other parcels of information. This latter point is particularly significant in that it furnishes a basis for the synthesis of separate parcels of information that are all related to the same object, concept, observations, etc.

Woods[5] defines a semantic network to be an attempt to combine into a single mechanism both the ability to store factual knowledge and the ability to model associative connections which render certain parcels of information accessible from certain others. Moreover he indicates three criteria which must be satisfied by a notation used for semantic representation:

1. Logical adequacy. The notation must provide an exact, formal and unambiguous representation of any particular interpretation that may be given to a sentence.
2. There must be an algorithm for translating an initial sentence into this notation.
3. There must be algorithms capable of using the semantic representation in order to perform needed inferences and deductions.

The information base detailed in the subsequent discussion will be shown to satisfy the definition of a semantic network. A query language will be described which, in conjunction with the information base, will be shown to satisfy the three requirements of notations for semantic representation.

FEATURES OF THE INFORMATION BASE

A specific design and implementation of the information base is described in ε later section; this design and implementation is based in part upon the idea of a network data base advanced in the CODASYL DBTG Report of 1971[1] and subject to extensions and modifications outlined in Reference 6. The term information base, rather than data base, is used to emphasize its incorporation of two fundamental features which do not appear in the general data base management literature. Both features concern ways of patterning data that can convey information not commonly treated in the guise of data base management, but of value to decision makers; they furnish methods for introducing two novel kinds of context into data structures.

In observing the progression from linear structures to trees, to networks, we note increased facility for relating a datum with other data; there is an increased capacity for specifying the context of a datum in terms of data structures. Though there is little context inherent in linear data structures, the data content of groups of such structures may be used to represent trees. This becomes complex and cumbersome as the tree to be represented grows in size. Similarly, though it is possible to twist tree structures to the task of representing large or complex networks by using collections of tree-like structures, this cannot be accomplished in a facile, straightforward manner. This analogy may be continued with respect to the two features being introduced in this section. That is, they can, in some sense, be represented within network data structures, but such an approach leads to certain asymmetries (with respect to processing) and difficulties akin to those encountered when representing trees in linear structures. Since the two features are not inherent in the common notion of a network data base, we introduce the information base as a mechanism which encompasses both while allowing full network capabilities.

The first feature involves the introduction of the concept of resolution levels within the mechanism for information organization. A simple example of this is described by Winograd.[7] Consider data about cars in which specific weights and colors are related (linked) to each car; on a higher level of resolution, we may want to somehow store information about what the properties of cars are. So on one level of resolution we are interested in specific attributes of specific cars and on another level we are concerned with properties of cars. Thus two distinctive characteristics of the information base are links which integrate individual information parcels on a given level of resolution into a single network structure and secondly, the integration of information of varying levels of resolution into a single structure. We term the former characteristic "horizontal integration" and the latter "vertical integration." So horizontal refers to linkage of entities on the same level; whereas vertical denotes linkage among different levels via information parcels that participate in both levels (though the nature of participation is different on each level). A subsequent section of this paper describes both an implementation and the implications of this feature.

The second outstanding feature of the information base involves its ability to handle the integration of programs into its logical structure. Not only does this permit the linkage of a datum with a program that uses it; it allows the construction of networks (in both the horizontal and vertical sense) of programs. This capacity has two primary effects. First, it provides the basis for model formulation. Second, it furnishes a more comprehensive mechanism for semantic representation.

The aspect of model formulation involves the action of relating certain modules into a desired configuration. This necessitates a knowledge of which configurations are meaningful and which are not. Such knowledge is stored in the information base's semantic network. This approach has much in common with the notion of structured programming. Programs devised according to the tenets of structured programming[8] are readily amenable to storage within the information base; indeed there is also the ability to store alternative modules (e.g., alternative functional forms) for performing a particular role within the context of either other modules or a higher resolution level. The advantages of structured programming in terms of maintainability and extensibility[9] are also apparent in the strategy of integrating program modules into the logical structure of an information base. That is, it is possible to add, replace or delete a module in the same manner that one would add, replace or delete an occurrence of data.

It is useful at this juncture to point out a distinction between program modularity and program resolution. The idea of resolution level also goes under the name of level of abstraction. Dijkstra[8] indicates that each level of a system's software hierarchy constitutes an abstract resource which participates in the next higher level and which has available to it the resources of lower levels. So ". . . at one level the programming amounts to manipulation of the abstract resources supported by the next lower level of the hierarchy. The programs at that level manipulate abstractions—the abstractions of the resource, whatever it may be—and at the same time participate in generating a higher level of abstraction for the next level of the hierarchy to manipulate."[10] Furthermore, Miller and Lindamood suggest that a ". . . highly modular implementation is one in which specific functions are performed by specific modules (and nowhere else); on the other hand, a system which preserves a hierarchy of abstract resources would appear to require modularity as a minimum, and perhaps a great deal more 'structure'."[10] Such a structure is effectively treated by the information base feature of resolution levels which allows the arrangement of program resources into levels of abstraction.

The second effect of allowing the integration of programs into the structure of the information base is the more comprehensive semantic representation that is permitted. Much literature about semantic networks is concerned with the network representation of English sentences (e.g., see References 5 and 11). These sentences consist of patterns of verbs and arguments. The typical decision maker who queries the information base requests the execution of some model (i.e., operators, verbs) using certain data (i.e., operands, arguments) as inputs. The usual data base structures handle information about arguments only; the meaningful operator contexts in which such arguments may appear is not represented in standard types of data base structures. A more detailed discussion and practical application of this feature of representing programs in an information base is presented in Reference 12. The remainder of this paper focuses on details and examples of the resolution level feature and on the utility of the information base as a device for semantic representation.

REQUIREMENTS FOR A DECISION SUPPORT SYSTEM

Recall that, in this paper, we are principally concerned with the information base from the standpoint of its contribution to the realization of a general decision support system. Although there are several facets involved in reaching decisions, we investigate three in particular: information access, model formulation, and analysis. The efficacy of a decision support system may be evaluated in terms of its flexibility, facility, scope, timeliness and cost in supporting these three facets.

With respect to information access there must be a mechanism for the systematic, integrated storage of all pertinent information. The information base outlined above provides just such a mechanism, through both horizontal and vertical integration and through its capacity to relate operators with each other and with arguments. Given such a storage mechanism there must be a technique for interrogating (and modifying) it that can be used by decision makers who are not computer experts or programmers. The query language for accomplishing this is presented later.

The second facet which must be supported is the activity of model formulation. This facet refers both to models that are subsequently used for purposes of analysis and to models in the sense of plans to be implemented. This is a crucial aspect for resolving unstructured problems and for supporting the exploratory aspects of decision making. In short, the decision support system must have a component for the generation and evaluation of alternatives for achieving a stated goal. As already indicated, the information base contributes to such an end.

The decision support system must also provide for the activity of analysis; i.e., the fitting of data with models and models with data, thereby resulting in some expectation, beliefs or knowledge. Implicit in the very nature of the planning activity is the dynamic quality of the interface between model and data; for even though a collection of data may be comparatively stable over some time period, both the problems and the models used for problem solving may be subject to frequent alteration. Notice that a model operates on a particular subset of the entire collection of operands available, and it requires a certain configuration of this data as input. We contend that the tedious, cumbersome task of interfacing data and models for purposes of analysis should be automatically handled by the decision support system in response to the commands of a non-programming user. The method for accomplishing this is discussed in subsequent sections.

FORMALIZATION OF THE INFORMATION BASE

We now present a formal description of what is meant by the term "information base." We define a record occurrence to be a uniquely labeled aggregate of data (i.e., string of symbols). Where I^+ is the set of positive integers, let X_0 be the set of labels associated with a finite set of record occurrences, such that $X_0 \subset I^+$. A record type, uniquely denoted by the label p_i, may be described by a function r_i as follows. Define R_k as the set of all $r_i : X_k \rightarrow \{0,1\}$ such that:

(1) $\forall x \in X_k,\ r_i(x) = \begin{cases} 1 & \text{If } x \text{ is of the type labeled } p_i \\ 0 & \text{otherwise} \end{cases}$

(2) $\forall x \in X_k,\ \sum_i r_i(x) \leq 1$

(3) $\forall r_i \in R_k,\ \sum_j r_i(x_j) > 0 \quad \text{where } X_k = \{x_j\}$

Property (1) states that r_i defines the collection of $x \in X_k$ of the type labeled p_i. Property (2) indicates that each $x \in X_k$ can belong to at most one p_i. Property (3) states that each r_i is non-trivial.

Before defining X_k for $k > 0$, we note that $P_k = \{p_i\}$ is the set of all labels associated with the elements of R_k. Since X_0

is finite, we can define these labels such that $P_k \subset I^+$, $P_k \cap X_0 = \emptyset$; furthermore we can define each of these sets of labels such that it has no elements in common with any other P_k. Define:

$$X_1 = \{p_i \in P_0 \mid \exists x \in X_0 : r_i(x) \neq 0\} \cup X_0$$
$$X_2 = \{p_i \in P_1 \mid \exists x \in X_1 : r_i(x) \neq 0\} \cup X_1$$
$$\vdots$$
$$X_N = \{p_i \in P_{N-1} \mid \exists x \in X_{N-1} : r_i(x) \neq 0\} \cup X_{N-1}$$

It follows from the definition of X_0 and R that there must exist a K such that $X_k = X_{k+1} = \ldots$; then let $X = X_k$. Observe that X is the set of labels of all record occurrences within an information base; these labels are unique identifiers, thereby serving as information base keys. All occurrences of a record type denoted by the label p can be determined by successive applications of the function r to the set X. The magnitude of K indicates the levels of resolution inherent in the information base. The reader will notice that P is always a subset of X; if it were not desired to treat all record types as record occurrences, one could define $X = X_{k-1}$. There are advantages to defining $X = X_k$, especially for purposes of altering the logical structure of an information base after it has been loaded. This will be elaborated in a subsequent section.

Continuing, we now formally define the information-set (in-set). This construct, as implemented in the information base, is drawn in part[6] from the "set" idea of the CODASYL DBTG Report,[1] hence the term "in-set." It is important to differentiate this from the familiar notion of a mathematical set. Let $Q_i = \{x \in X \mid r_i(x) \neq 0\}$. If a function associates each element of its domain with no more than one element of its range it is said to be a functional relation. Then each functional relation $f : Q_j \rightarrow Q_i$ uniquely defines an in-set of which the record type r_i is said to be the owner and the record type r_j is called the member. It is important to make several observations about the in-sets of an information base. It is permissible, and sometimes useful,[6] to allow $i = j$. Second, an in-set may be used to associate record types of different levels of resolution. Third, the set F of in-sets of an information base must be carefully defined so that its elements are consistent; e.g., one should exercise caution in defining both $f_1 : Q_i \rightarrow Q_j$ and $f_2 : Q_j \rightarrow Q_i$ as elements of F. Finally if $f_1 : Q_j \rightarrow Q_i$ and $f_2 : Q_i \rightarrow Q_k$, then we can form the composite in-set $f_1 \circ f_2 : Q_j \rightarrow Q_k$ defined by

$$(f_1 \circ f_2)(x) = f_1(f_2(x)) \; \forall x \in Q_j.$$

This is sometimes desirable from the standpoint of access efficiency; it also allows us to attach special significance or meaning to certain groups of sets.

The foregoing is a formal description of the major features of the information base. It accounts for both the horizontal integration (via in-sets) and vertical integration (via resolution levels) of information into a single mechanism. In order to illustrate the use of resolution levels, we apply the above formalisms to the problem (see Winograd[7]) of representing information about cars. In this problem cars are to be described in terms of color and weight; in addition we would like to denote that color and weight are properties. Suppose we have record occurrences as shown in Figure 1a; these are identified by the respective labels in X_0. The set R_0 is also shown; by inspection we see that R_0 satisfies the needed conditions as given at the beginning of this section. The function r_1 determines whether or not an element of X_0 is of the type color. Similarly r_2 is associated with the type weight and r_3 is associated with the type car. In our implementation each r_i defines (and is defined by) a linked list of occurrences of its type. Given X_0, R_0, and P_0 we apply the rule for defining X_1 to obtain the result shown in Figure 1b. R_1 is also given and clearly satisfies the necessary conditions for its definition. Application of r_4 to elements of X_1 can be used to determine which elements are vehicle properties. Figure 1c gives the X_2 that follows from the definition. If we take $R_2 = \emptyset$, then $X = X_2$.

The occurrences and their "vertical" relations with each other are diagrammed in Figure 2. Also depicted are two in-sets: f_1 and f_2. Using the definitions of Q_1, Q_2, and Q_3 given in Figure 2, $f_1 : Q_3 \rightarrow Q_1$ and $f_2 : Q_3 \rightarrow Q_2$. The arrows in the diagram point from the owner of the in-set to the member; i.e., each arrow points in the direction opposite to that in the notation of its corresponding functional relation. Using the formalisms introduced here it is a simple matter to represent an extended problem including other kinds of vehicles,[13] more properties, subclassifications of properties (e.g., structural, functional, etc.) and even properties of properties.

An information base for water quality management

More detailed discussions of the water quality management problem may be found in References 14 and 15. The objective of the example presented in this section is to demonstrate the applicability of the information base as a

$$X_o = \{ \overset{\text{RED}}{1} , \overset{\text{BLUE}}{2} , \overset{\text{1-TON}}{4} , \overset{\text{2-TON}}{5} , \overset{\text{CAR-1}}{15} , \overset{\text{CAR-2}}{16} \}$$

$R_o = \{ r_1, r_2, r_3 \}$ with labels $P_o = \{ 8,9,11 \}$

where $r_1(x) = \begin{cases} 1 & x < 4 \\ 0 & \text{otherwise} \end{cases}$

$r_2(x) = \begin{cases} 1 & 4 \leq x \leq 7 \\ 0 & \text{otherwise} \end{cases}$

$r_3(x) = \begin{cases} 1 & x \geq 15 \\ 0 & \text{otherwise} \end{cases}$

a.

$X_1 = \{ 1,2,4,5,15,16,8,9,11 \}$

$R_1 = \{ r_4 \}$ with label $P_1 = \{14\}$

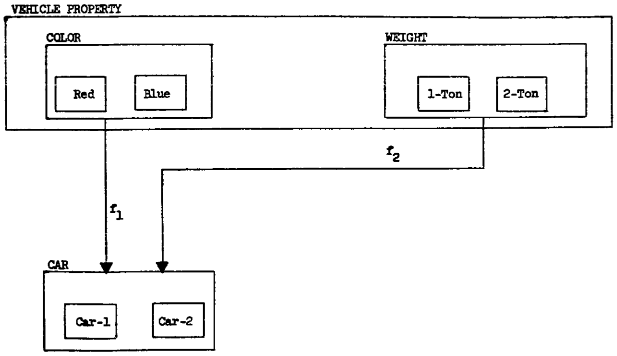
where $r_4 = \begin{cases} 1 & 8 \leq x \leq 10 \\ 0 & \text{otherwise} \end{cases}$

b.

$X_2 = \{ 1,2,4,5,15,16,8,9,11,14 \}$

c.

Figure 1—Resolution levels for representing information about cars



Let:

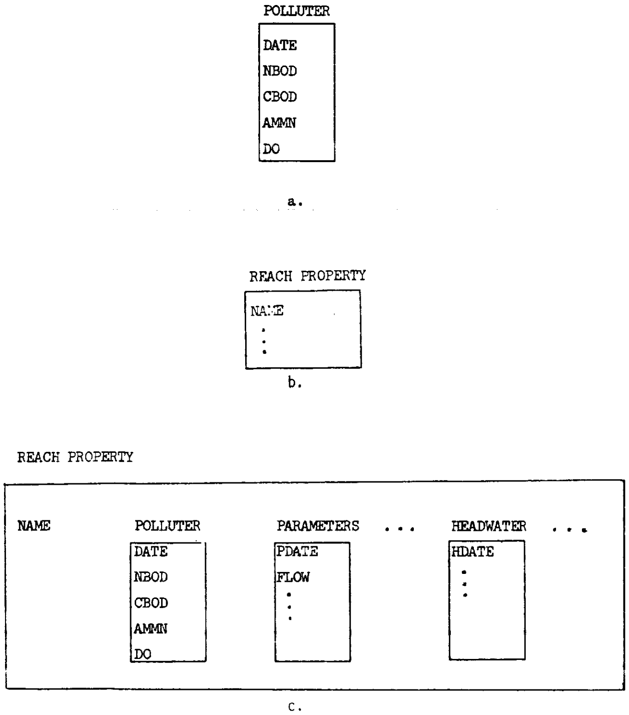
$$Q_1 = \{x \in X \mid r_1(x) \neq 0\}$$
$$Q_2 = \{x \in X \mid r_2(x) \neq 0\}$$
$$Q_3 = \{x \in X \mid r_3(x) \neq 0\}$$

Figure 2—Occurrences in car information base

device for capturing the semantics used to support practical decision problems. At this point, we presume that the reader has a sufficient concept of what an information base entails to obviate the need for complete formalistic description. So for the sake of economy, the following example is presented in a less formal manner than the previous one. It will be used to depict certain implementational details (e.g., languages in which the information base is specified and with which it is utilized).

Consider the record type POLLUTER, displayed in Figure 3a. This aggregate of data item types represents measures of types of polluter activity for a given date. So occurrences of this record type correspond to measurements taken on various dates. In order to build a semantic network, we must indicate how this concept of POLLUTER fits into the pattern of knowledge concerning water quality management. A polluter is properly characterized as being a property of a river reach. Other properties of a reach include reach parameters, headwater, incremental flow, and treatment plan. So a reach is characterized in terms of these properties as follows: a reach is a portion of a river in which certain water quality parameters are relatively invariant; which has no more than one (point-source) polluter, one incremental flow or one headwater; and which must possess treatment plans. This could be represented in the information base by occurrences of the REACH PROPERTY record type displayed in Figure 3b. However, observe that each occurrence of the data item NAME (e.g., "POLLUTER," "HEADWATER," "PARAMETER," etc.) is also the label of a record type which is itself an aggregate of item types and which may have numerous occurrences. So, for instance, "POLLUTER" denotes an occurrence of REACH PROPERTY; but it also denotes a record type (shown in Figure 3a). The same circumstance holds for the other reach properties, though their record types are not depicted here. The resultant logical structure is illustrated in Figure 3c; a record type enclosed by another record type indicates that the enclosed record type is also an occurrence of the enclosing record type.

We continue the example by examining general water quality modeling characteristics. In order to simulate water quality we need information about the following: the rivers involved, the reaches which are in each river, each reach's properties, junctions, piping plans, and model parameters. This is shown in the structure of Figure 4. Note that the record type GMC has two item types: CHAR (characteristic) and IMPT (a measure of the relative importance of each characteristic). Five occurrences of GMC are shown: RIVER,REACH,JUNCTION,MODELandPIPEPLAN. General Modeling Characteristic is not the only property of



a.

b.

c.

Figure 3—Attributes of a logical structure

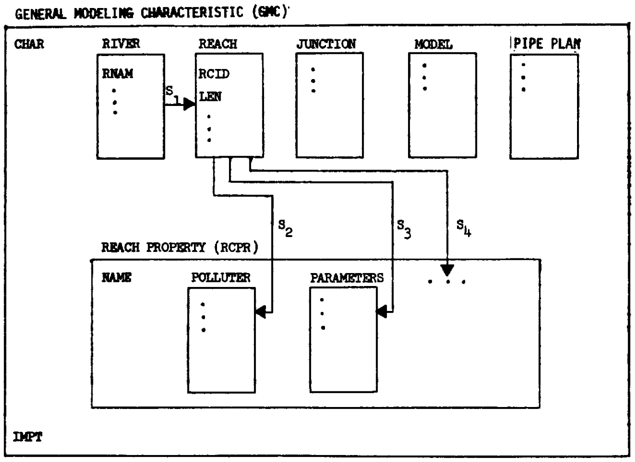


Figure 4—Example of logical structure (GMC)

a segment that needs to be represented; Local Modeling Characteristics (LMC) are also needed. (The term segment is used to indicate a particular area of a river basin.) The details of the high level record type LMC are not shown here, but they describe information about non-point sources of pollution, permits for point-source pollution, treatment plant construction status, permit violation data, etc. As shown in Figure 5, GMC and LMC are occurrences of SEGMENT PROPERTY which is itself an occurrence of the record type WQMA; BASIN and SEGMENT are also occurrences of this record type. The information base could be further extended to incorporate aspects of land use planning since they influence and are influenced by water quality management.

The foregoing logical structures are initially defined in terms of an Information Description Language (IDL). Use of the IDL to define the logical structure of Figure 4 is presented in Figure 6. The specification shown is largely self-explanatory. Each record type is followed by the item types which compose it. If the record type is of a high level, then its item types are followed by a specification of those record types which are its occurrences. Definition of an in-set must be preceded by specifications of its owner and member record types. For simplicity, details of the type and size of items are not shown; also the ordering criterion of each in-set is not shown.

A LANGUAGE FOR DECISION SUPPORT

The reader will observe from the preceding discussion that the decision support system has two basic components: an information base and a query language. Clearly the usability of a semantic network depends upon implementation of a language with which one can extract (insert) meanings that are held in the semantic net. Not only are semantics conveyed by a particular language, they are limited by it as well. The language is used to express meanings, but it also delineates the kinds of meanings which are expressed. We can devise arbitrarily complex semantic networks, but their usability is (from the practical standpoint) constrained by the languages (and language processors) which can be interfaced with them. Observe then that there is a fundamental duality of (1) the language in which ideas are expressed, and (2) the structural representation of ideas in an information base. On the other hand the semantic mechanism must be capable of taking full advantage of the language's power. In the case of the

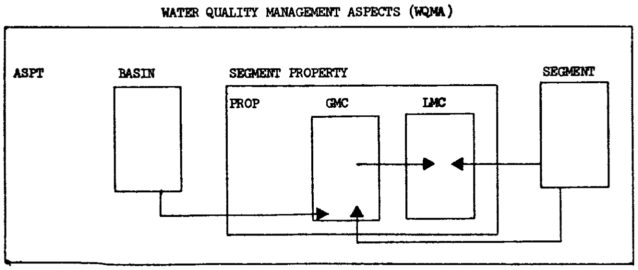


Figure 5—Example of logical structure (WQMA)

```
RECORD      GMC
ITEM        CHAR
ITEM        IMPT
  ⋮
IN          GMC
            RECORD      RIVR
            ITEM        RNAM
              ⋮
            RECORD      RECH
            ITEM        RCID
            ITEM        LEN
              ⋮
            RECORD      JUNC
            ITEM        JID
              ⋮
            RECORD      MODL
            ITEM        MID
              ⋮
            RECORD      PIPE
            ITEM        PPID
              ⋮
SET         S1
OWNER       RIVR
MEMBER      RECH
RECORD      RCPR
ITEM        NAME
  ⋮
IN          RCPR
            RECORD      PLTR
            ITEM        DATE
              ⋮
            RECORD      PARA
              ⋮
            RECORD       HDW
              ⋮
SET         S2
OWNER       RECH
MEMBER      PLTR
SET         S3
OWNER       RECH
MEMBER      PARA
  ⋮
```

Figure 6—Example of IDL for Figure 4

implementation described in this paper, the query language is the constraining factor since it is intended primarily for the practical support of decision activities of managers in both the public and private sectors.

Implementation of a natural language (e.g., English) processor is certainly a noble objective. It is our experience that the typical decision maker neither uses, nor needs, a complete facility for conversing in a natural language. It often happens that phrases or clauses are sufficient to convey an idea; there are grammatical constructs (e.g., reflexive, passive) which are not particularly germane to the decision activities of information access, model formulation, and analysis. In addition the decision maker is more prone to desire information conveyed in a tabular or graphical fashion than in a narrative mode. It has also been found that the user sitting at a computer terminal has a tendency to use abbreviations and concise mathematical notation.

With these factors in mind, the query language to be outlined here has been designed to meet the needs of decision makers for flexibility and brevity of expression, while at the same time being easy to learn and utilize. The query language is effectively a subset of English that has been extended to include standard mathematical operators (i.e., relational, arithmetic, and univariate and multivariate functions). The focus here is upon use of this language for interrogation, though it may be used for data creation and modification as well.[6]

The standard framework of the language consists of a collection of operators (used in the capacity of verbs, adverbs and adjectives) relating to operations typically performed by most decision makers; these operators are of two kinds: commands (e.g., LIST, PLOT, STAT, REGRESS, etc.) and mathematical operators (e.g., MAXIMUM, AVERAGE, =, +, <, etc.). In addition to this standard framework the user may define arguments (used in the capacity of nouns), synonyms for arguments and operators, and any further operators (i.e., programs to be integrated into the information base) that are mundane to the particular decision making application to be supported. This definition is effected in terms of an Information Description Language (IDL) which establishes the context(s) of all data, arguments and operators. That is, it defines the semantic net.

Details of the query processor are not discussed in this paper but may be found in References 16–18. Briefly, the query language has a context-sensitive grammar; inverse transformations are used to take a surface structure query into a deep structure expression in a language having a context-free grammar. This deep structure expression is compiled using well-known methods of syntax-directed analysis. Parts of the compiled expression are used as input to network traversal routines which make extractions from the information base for use in analysis indicated by the query's command (verb).

The query's syntax appears as follows:

⟨COMMAND⟩⟨FIND clause⟩⟨CONDITIONAL clause⟩

or alternatively,

⟨CONDITIONAL CLAUSE⟩⟨COMMAND⟩⟨FIND CLAUSE⟩.

So some sample queries are:

LIST REACH.NAME,REAERATION.PARAMETER AND REAERATION.EXPONENT. FOR DATE=110175 AND REACH.LENGTH<.9

WHEN DATE=110175, PLOT REACH.NUMBER VERSUS AMMONIA.CONCENTRATION AND DO.CONCENTRATION/LOG(TEMPERATURE)

LIST GENERAL.MODELING.CHARACTERISTIC IF IMPORTANCE>3

The language allows any meaningful configuration of arguments and mathematical operators to appear in the FIND and CONDITIONAL clauses.

PROCESSING HIGHER LEVEL RECORD TYPES

Upon receipt of a query, the query processor generates appropriate commands for traversal of a multi-level network. These commands are operators in an Information Manipulation Language (IML). We use the term IML to distinguish from the Data Manipulation Language (DML) proposed in the CODASYL DBTG Report.[1] The DML is intended to permit access, modification and retrieval for a single level network data base. The IML has the more extensive function of furnishing tools for manipulation of the information base. Thus the IML contains operators for handling traditional DML functions[16] and operators for processing higher level record types. The latter are discussed here.

In the DML, routines exist for creating a record occurrence at a unique location denoted by its key. The IML includes analogous routines for specifying that an existing record occurrence be treated as a record type as well. There are four such commands:

CRTK—Create Record Type based on a given Key

CRTR—Create Record Type based on the current occurrence of another Record type

CRTO—Create Record Type based on the current Owner of a given set

CRTM—Create Record Type based on the current Member of a given set

Another traditional DML operator (AMS) adds a specified record occurrence as a Member of a given Set. A similar IML operator is used for adding an existing occurrence of one record type as an occurrence of another record type. Note that utilization of this operator must be preceded by a generalization of the definition of a record type which was introduced above (i.e., the definition is generalized by removing the restriction that $\Sigma_i\ r_i(x)\leq 1$, $\forall x \in X_k$, where $r_i \in R_k$). This operator is AORT, Add Occurrence to Record Type, and it uses the key of the occurrence to be added. In conjunction with commands for the logical restructuring of a network data base,[6] AORT provides the ability to add and delete higher level record types and add existing occurrences to higher level record types; and this is accomplished without dumping and reloading data.

Finally operators are needed for determining the key of a record type, given an occurrence of the record type. These commands are:

GKRR—Get Key of the Record type for the current Record occurrence of that type

GKRO—Get Key of the Record type whose occurrence is the current Owner of some set

GKRM—Get Key of the Record type whose occurrence is the current Member of some set

These operators provide the capacity to proceed from a lower level occurrence to a higher level occurrence, when used in conjunction with traditional DML operators.

It must be emphasized that the typical user of the query system needs to have no knowledge of the IML operators, for they are automatically set up and executed by the query processor in response to a user query.

ADVANTAGES OF THE RESOLUTION LEVEL FACILITY

We contend that the concept of resolution levels effectively adds a new dimension to the field of information storage. The preceding discussion has suggested a means for operationalizing this concept as an extension to the traditional single-level network approach. One advantage is that multi-level semantic networks may be stored without introducing asymmetry in the interpretation and processing of in-sets and record types. Since a record type may also be defined to be an occurrence of a higher level record type, the addition of a record type is treated by creating a new record occurrence at the next higher level. That is, we remove the distinction between data values and the structural pattern according to which data is organized. In other words, the terms "attribute" and "value" are recognized as being relative, so that what is a value on one level is an attribute on another and vice versa.

From one viewpoint this abolishes the special status of an IDL specification by permitting record type definition to be a dynamic process. That is, the creation of a new record type is synonymous with the creation of a new record occurrence of a higher level record type. Thus the IDL specification of the highest level of resolution is effectively reduced to the definition of three record types (one describing information about record types, another relating to information about sets, and one with various system information[6]) and some in-sets between them. This definition is always the same regardless of the content and structure of lower resolution levels.

A second advantage, already mentioned in connection with integration of programs into the information base, concerns a mechanism for handling levels of abstraction in software. A third advantage is that higher level record types may be used to characterize areas of an information base by assigning record types of a particular area to be occurrences of a higher level record type; these areas may be defined for a variety of reasons (e.g., for information security, to denote scenarios, to delimit functional areas—which may overlap, etc.). As the information base becomes large and varied in content, this technique may also be used to realize efficiencies in path determination processing by limiting the scope of network traversal to a particular information base area.

THE INFORMATION BASE AS A DEVICE FOR SEMANTIC REPRESENTATION

With the foregoing background, we can now address the three criteria proposed by Woods,[5] which must be satisfied by a notation used for semantic representation. First observe that the information base is a tool for the representation of a semantic network (i.e., a single mechanism with both the ability to store factual knowledge and the ability to model associative connections which render certain parcels of information accessible from certain others).

The first criterion of a notation for semantic representation is logical adequacy. The notation must provide an exact, formal and unambiguous representation of any particular interpretation that may be given to a sentence. Recall that the sentences with which we are concerned are those allowed in the query language for decision makers.

The information base allows a given query to have a multitude of interpretations. The query specifies a group of data items which may be related to each other in many ways via vertical and horizontal linkages in the information base. Each path of linkages on which these items lie corresponds to a particular interpretation of the query. Upon receiving a query which is subject to multiple interpretations the query processor prompts the system's user in order to ascertain which interpretation (i.e., path) is intended. Details of the manner in which this has been implemented may be found in References 16 and 18.

The second criterion is that there must be an algorithm for translating an initial query into the notation of the information base. This is the central function of the query processor whose operation has already been described; implementational details appear in Reference 13. The third criterion, concerning algorithms capable of using the semantic representation, has also been addressed in the discussion of the query language. Observe that the IML provides the means for interfacing algorithms with the semantic representation.[16] Algorithms which have been used range from relatively commonplace report generators to large scale water quality simulation models.[15]

CONCLUSION

In the beginning we observed that it is the patterning of symbols which can convey information; a datum's meaning derives from its context, from its relationships with other data. Thus when considering the design and implementation of systems for decision support, a crucial point is the power of available tools for representing contexts. The value of such systems is constrained by the "richness" of patterning allowed by their data structure mechanisms. Observing the progression from relatively impoverished linear structures to trees and networks, we note that each stage has provided a more powerful and flexible tool for semantic representation. In this paper we have introduced the notion of an information base as a natural step forward in the continuing evolution of data structures. An outstanding feature of the information base is its accommodation of both the horizontal and vertical integration of information parcels into a single mechanism. An information base implementation which builds upon network concepts was discussed. A topic for future research is the investigation of an information base implementation which builds upon the relational data base notions.[19] A second distinctive feature of the information base, namely the integration of operators into its structure, was briefly described. The information base is utilized by a non-procedural, English-like query language, that has been designed for decision support applications. This language, in conjunction with the information base, satisfies the requirements for a notation for semantic representation.

ACKNOWLEDGMENTS

The authors are indebted to Professor Walter Reitman of the University of Michigan, Professor Tibor Vamos of the Hungarian Academy of Sciences and Dr. Bertram Raphael of SRI for useful discussion on semantic network and data base management.

REFERENCES

1. CODASYL, Data Base Task Group Report, ACM, April 1971.
2. Miller, J., "Living Systems: The Organization," *Behavioral Science,* Vol. 17, January 1972.
3. Kneitel, A. M., "Hard vs. Soft Information," *Management Datamatics,* June 1976.
4. Albarda, J. D., *Structures and Relations in Information,* Growningen, Druk: V.R.B. Offsetdrukkeriji, Rotterdam, 1974.
5. Woods, W. A., "What's in a Link: Foundations for Semantic Network," in *Representation and Understanding* (ed. Bobrow, D. G. and A. Collins), Academic Press, New York, 1975.
6. Bonczek, R. H., C. W. Holsapple, and A. B. Whinston, "Extensions and Corrections for the CODASYL Approach to Data Base Management," *International Journal of Information Systems,* Vol. 2, 1976.
7. Winograd, T., *Understanding Natural Language,* Academic Press, New York, 1972, pp. 23–27.
8. Dijkstra, E. W., "Notes on Structured Programming," T.H.E. Report No. EWD-248, 70-WSK-03, 2nd Edition, April 1970.
9. Donaldson, J. R., "Structured Programming," *Datamation,* December, 1973.
10. Miller, E. F. and G. E. Lindamood, "Structured Programming: Top-down Approach," *Datamation,* December 1973.
11. Heidorn, G. E., "Automatic Programming Through Natural Language Dialogue: A Survey," *IMB Journal of Research and Development,* July 1976.
12. Bonczek, R. H., C. W. Holsapple and A. B. Whinston, "Implementation of a Decision Support System for Regional Water Quality Planning," Krannert Institute Paper No. 570, Purdue University, West Lafayette, Ind., September, 1976.
13. Bonczek, R. H., "Theoretical Description of Access Language for a General Decision Support System," Doctoral Dissertation, Purdue University, 1976.
14. Haseman, W. D., C. W. Holsapple and A. B. Whinston, "Implementation of a Large Scale Water Quality Data Management System," *Socio-Economic Planning Sciences,* Vol. 10, March 1976.
15. Holsapple, C. W. and A. B. Whinston, "Decision Support System for Area-wide Water Quality Planning," *Socio-Economic Planning Sciences,* Vol. 10, 1976.
16. Haseman, W. D. and A. B. Whinston, *An Introduction to Data Management,* Richard D. Irwin Co., Homewood, Illinois, 1977.
17. Bonczek, R. H., W. D. Haseman and A. B. Whinston, "Structure of a Query Language for a Network Data Base," Technical Report, Krannert Graduate School of Management, Purdue University, April 1976.
18. Bonczek, R. H., W. D. Haseman and A. B. Whinston, "Automatic Path Determination in a Network Data Base," Technical Report, Krannert Graduate School of Management, Purdue University, April 1976.
19. Codd, E. F., "A Relational Model of Data for Large Shared Data Bases," *Communications ACM* 13 (6), June 1970.

Laboratory automation via a VM/370 teleprocessing virtual machine

by A. A. GUIDO and J. CONSIDINE
IBM Thomas J. Watson Research Center
Yorktown Heights, New York

ABSTRACT

A mechanism called the Teleprocessing Virtual Machine (TPVM) has been designed to provide remote intelligent sub-systems with the ability to access and utilize the power of the IBM VM/370 environment. This paper describes the TPVM implementation and operation in a laboratory environment at the Yorktown Research Center. However, the TPVM mechanism is applicable to a much broader spectrum of usage.

INTRODUCTION

Previous publications[1–4] by members of the Thomas J. Watson Research Center's Laboratory Automation group have described an implementation of a hierarchical system capable of providing a high degree of performance and availability for laboratory applications. These publications have stressed the considerations and requirements applicable to a distributed system.

Briefly stated, laboratory automation is considerably more than the acquisition of data. Laboratory automation encompasses major requirements for extensive data processing, data management, graphical display, and document preparation. It is a development tool wherein flexibility is of major importance, while at the same time the reliability of the data gathering apparatus is critical to the success of the experiment. The problem is to reconcile the requirement of constant availability for experiments of durations varying between minutes and days, with the need for large amounts of space to store the data accumulated over extended periods of observation, and for high-speed processing power to analyze and display the data most effectively.

Dedicated computer systems abound[5] wherein most, if not all, laboratory automation requirements are met. However, proliferation of such systems may be extremely costly. The approach taken in the referenced papers is to assign the tasks of data collection and experimental control to a small very reliable self-contained processor (e.g., IBM System/7), and the tasks of data storage, analysis, and formatting to a powerful central facility capable of serving a number of experiments. The work of these authors indicates that this arrangement meets all the laboratory automation requirements at a substantial reduction in cost to the end user.

This paper describes the VM/370[6] implementation of this computer hierarchy and in particular focuses upon the Teleprocessing Virtual Machine (TPVM) which is the heart of the hierarchy.

TPVM

Figure 1 illustrates the first two levels in the computer hierarchy or distributed system. Level 1 concerns itself with the real-time event driven operations and is comprised of an intelligent controller, such as the System /7, coupled to one or more experiments and user consoles. Level 2 is the time-sharing level operating in a VM/370 environment, and provides the data processing and data base requirements for the distributed system. The link between the two levels is made via a VM/CMS[7] machine called TPVM.

TPVM acts as the interface between the remote intelligent controllers and the VM/370 environment. It is trans-

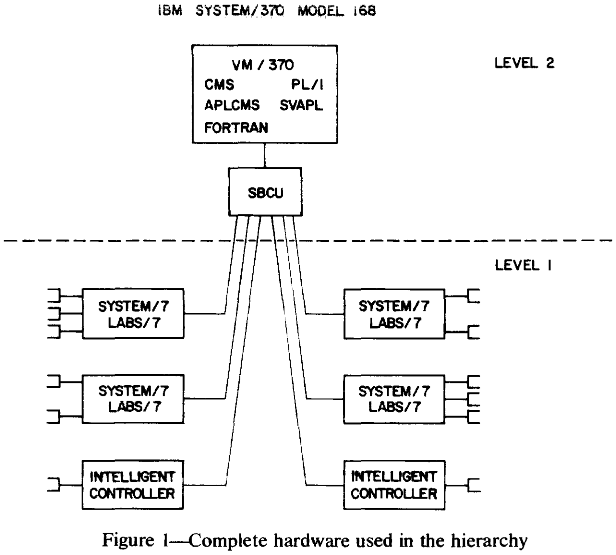


Figure 1—Complete hardware used in the hierarchy

parent to the remote controller user and acts as a slave to the controller. User-requested tasks, other than the movement of data or programs between levels, are assigned to ancillary VM/CMS machines by TPVM. VM/CMS filespaces are linked to users, thus extending any local controller data base. Additionally, TPVM makes the interactive facilities of VM/370 directly available to the remote controller consoles.

Because of TPVM's modular construction, the communications link between levels may be start/stop at speeds from 134.5 to 50K baud, bisync at speeds from 2400 to 9600 bps, or coaxial connections at 277K bytes/sec. via a Sensor Based Control Unit.[8]

TPVM DEVELOPMENT

The concept of distributed intelligence, while not new, gained tremendous impetus in laboratories with the advent of comparatively inexpensive intelligent controllers. Selection of the VM/370 environment came about naturally for the following reasons:

1. Thomas J. Watson Research Center is VM/370 oriented,
2. previous successes with laboratory automation in a CP/67 environment[1],
3. flexibility and language facilities such as APL, PL/1, FORTRAN, etc.,
4. ability to create a tailored machine with testing and debugging facilities such that other users would not be affected,
5. VM/CMS command structures directly applicable to our implementation,
6. interactive facilities directly extendable to remote controller consoles.

TPVM IMPLEMENTATION

TPVM implementation was carried on in a step-wise fashion. Our initial implementation consisted solely of providing a data base extension for the remote controller. TPVM is given the privilege of writing or reading data or programs stored in VM/370 filespaces only for that group of users tied to each controller. (A password file is maintained which prevents unauthorized access.) For this purpose, the remote controller user may issue an OPENIN/OPENOUT request to gain access to his VM/370 filespace. Subsequent READ/WRITE requests permit the user either read or write access to his filespace. VM filespaces are closed upon user request or when any error is detected.

Those familiar with VM/CMS will quickly realize the similarity to the FSOPEN, FSREAD, FSWRITE and FSCLOSE commands. In fact, the File Control Block (FCB) used in the execution of the CMS commands is filled with the necessary parameters as passed to TPVM from the remote controller.

The second step in TPVM implementation was to permit the remote controller users access to their VM/CMS machines for batch mode execution. This step required the addition of a DIAGNOSE command which permitted TPVM to logon a user VM/CMS machine and request execution of a user specified program with desired parameter set. (Since the implementation of VM/370 Version 3.3, this DIAGNOSE command has been discontinued. The function is now provided by the 'CP AUTOLOG' command.) *Notice that task execution is performed in a user machine independent of TPVM.*

Synchronization of VM/CMS and remote controller tasks is achieved via VM filespaces or extended external interrupts. When the user VM/CMS machine operating in batch mode completes its task, it writes a special STATUS file containing necessary information for the remote controller which can be retrieved via the OPENIN/READ/CLOSE mechanism.

The third and final step, bringing the complete VM/CMS interactive facilities directly to the remote controller, required changes to both the CP and CMS environments. Some of these changes are now part of Release 3 of VM/370. A group led by Alex Chandra designed a mechanism[9] whereby privileged machines could logon copies of user VM/CMS machines, and trap the console messages sent or received by the VM/CMS machine via extended external interrupts. TPVM fields these interrupts and passes the messages between the controller and user VM/CMS machine.

The following sections will further describe in greater detail the inner workings of TPVM. An example of this implementation using IBM System/7's[10] as intelligent remote controllers will also be described. Note however, that any intelligent controller conforming with the TP line protocol or interface can be used.

TPVM ORGANIZATION

The programs operating in TPVM are organized in a hierarchical fashion, and are modular in design. The basic functional components are an overseer or driver, an I/O communications package, and the operating portion which actually processes the requests from the controllers (see Figure 2).

In essence, the driver program activates the I/O communications package to receive requests from the controller(s). When such a request is received, the driver identifies the type of service requested and passes the request with whatever additional data are required to the appropriate functional routine. On completion of processing by the functional routine, the driver passes the results to the communications package for transmission to the controller.

With LABS,[11-13] now known as the Event-Driven Executive (EDX), as the operating system in a System/7, there are two basic types of requests that are created. The first of these is the isolated request. In this case, a task acquires control of the System/7 teleprocessing facility, sends one request, and then releases control. The other type is what we call the "stream of requests," as, for instance, in

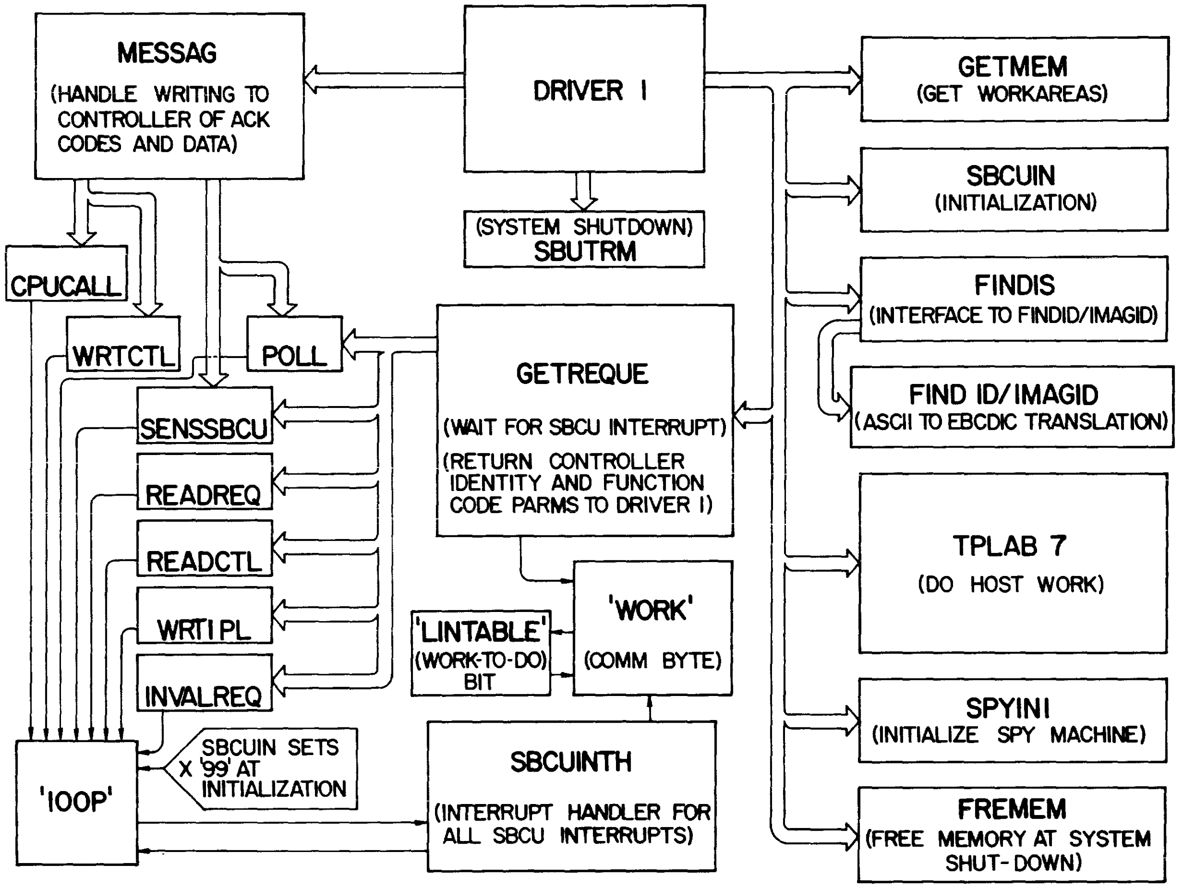

MESSAG
(HANDLE WRITING TO CONTROLLER OF ACK CODES AND DATA)
DRIVER I
GETMEM
(GET WORKAREAS)
SBCUIN
(INITIALIZATION)
(SYSTEM SHUTDOWN)
SBUTRM
CPUCALL
FINDIS
(INTERFACE TO FINDID/IMAGID)
WRTCTL
POLL
GETREQUE
(WAIT FOR SBCU INTERRUPT)
(RETURN CONTROLLER IDENTITY AND FUNCTION CODE PARMS TO DRIVER I)
FIND ID/IMAGID
(ASCII TO EBCDIC TRANSLATION)
SENSSBCU
READREQ
READCTL
WRT1PL
INVALREQ
TPLAB 7
(DO HOST WORK)
'WORK'
(COMM BYTE)
'LINTABLE'
(WORK-TO-DO) BIT
SPYINI
(INITIALIZE SPY MACHINE)
SBCUIN SETS X '99' AT INITIALIZATION
'IOOP'
SBCUINTH
(INTERRUPT HANDLER FOR ALL SBCU INTERRUPTS)
FREMEM
(FREE MEMORY AT SYSTEM SHUT-DOWN)

Figure 2—TPVM organization

sending a file to the host machine. In this case a task takes control of the teleprocessing facility in a controller and maintains that control for a sequence of requests, starting, in this case, with specifying the identity of the file to be created, continuing through sending all the data to be written, and concluding with a request to close the file. The details of the operation depend on what type of communication connection has been established. For dedicated connections to controllers via devices of the $270x^{14,15}$ or $370x^{16}$ type, each controller communicates with its own 270x/370x port, and has a TPVM for its individual use. This was the initial form of the implementation (see Figure 3). In this case, the TPVM processes requests from the controller essentially one at a time. The request comes in from the controller, the driver determines the function, and directs the request to the appropriate processing routine. After the request is processed, the results are passed back to the communication routine and then to the controller. The telecommunication facility of the controller is dedicated to a single task in the controller for the duration of the request (isolated or stream). Since there is only one controller communicating with each TPVM, the processing routines operate on one request for one controller at a time and process essentially sequentially from the beginning of a request to the end.

When the Model 5098-N5 Sensor-Based Control Unit (SBCU) is introduced as the communication mechanism with the controllers, the situation becomes rather more complex. Up to 64 different controllers can be attached to the SBCU, which occupies only one address on a selector channel of the host, and is therefore served by a single TPVM (see Figure 4). Requests can arrive from a number of different controllers and the TPVM must process each of them in timely fashion. This fact has implications for all of the components of the TPVM code.

In the course of converting to the SBCU environment, with more than one controller to be serviced by the same TPVM, the original single-strand implementation was analyzed carefully to identify those portions of the code which would be sensitive to the possibility of having more than one thing to do concurrently. At the same time, the original functions were made much more modular so that the underlying structure would be as clear as possible. As a result of this analysis it was possible to divide the original processing routines into one portion which could be made shareable among more than one request and another portion which was inherently sequential, and therefore would have to be executed for each request in turn.

One of the valuable tools of this analysis was a process called Basic Testing. This was applied to the key processing

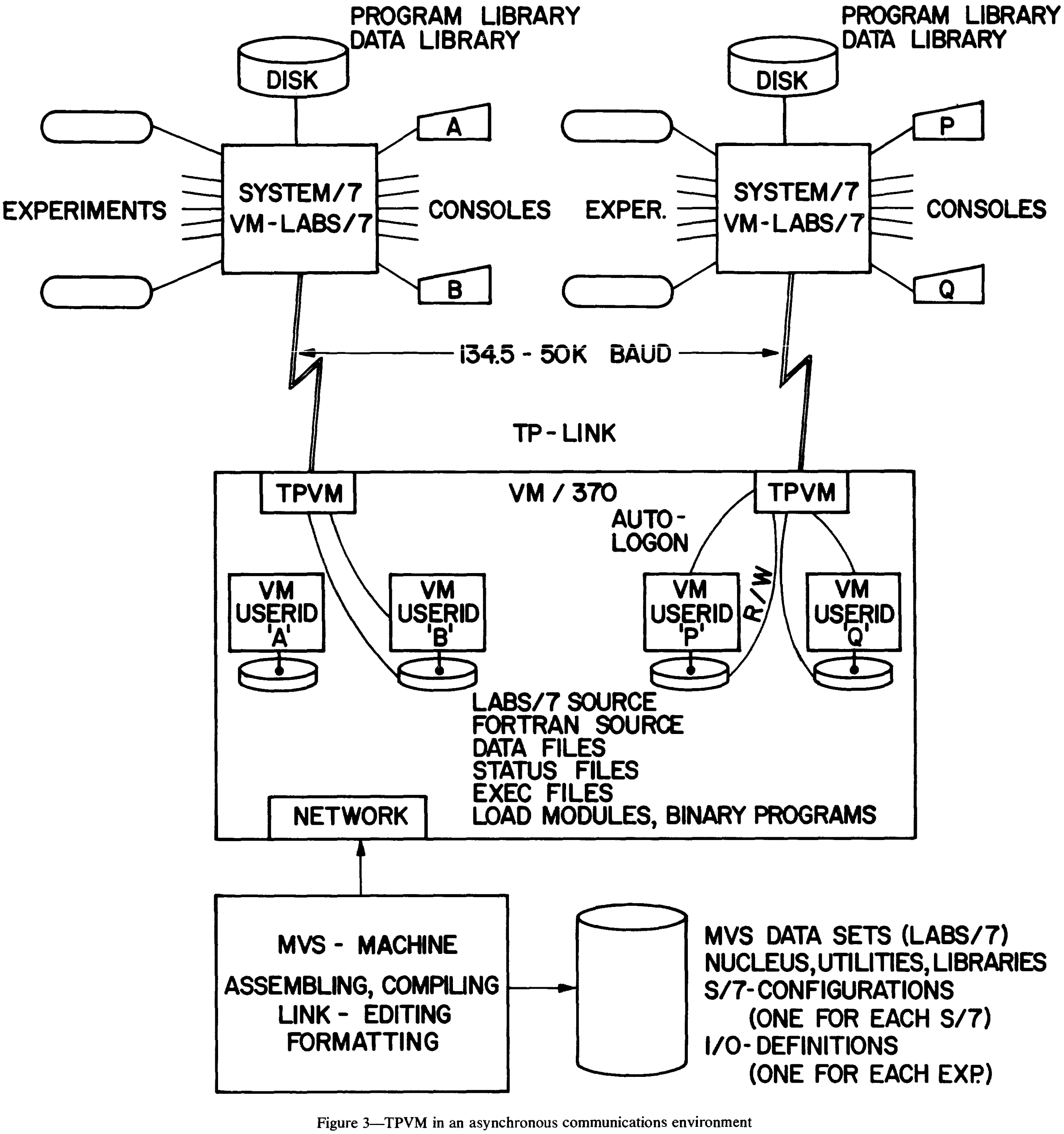


Figure 3—TPVM in an asynchronous communications environment

module. It consists of an exhaustive and detailed analysis of the logic of a program, beginning by identifying each block of code which has one entry and one exit point. These blocks of code are often as small as one or two lines of code. Next the flow between these blocks is examined carefully. This enables the identification of two different significant types of blocks. The first of these is so-called "dead code," which is code which can never be referenced either because there is no flow at all to it, or because the logic of the tests and branches is such that the code could never be entered. Removal of such blocks immediately contributes to the clarification of the remaining code. The other major type of block is the very frequently used code which is branched to from several different places in the module, and is a prime candidate for subroutinization. The main purpose of the Basic Testing approach is to clarify the structure of the program by linearizing the flow of execution for a particular function within the program. Having identified the blocks which are used to perform the function, it is often possible by rearranging blocks and creating subroutines to insure that the execution of the function takes place in a compact linear way, rather than by a series of branches all over the module. As a result of our analysis, the program was rearranged for greatly increased clarity, while at the same time, a number of errors were detected without having had to actually execute the code.

The three basic components of the TPVM were mentioned above; i.e., the driver, the communications package, and the processing routines. We will now examine the effect on these components of the introduction of the

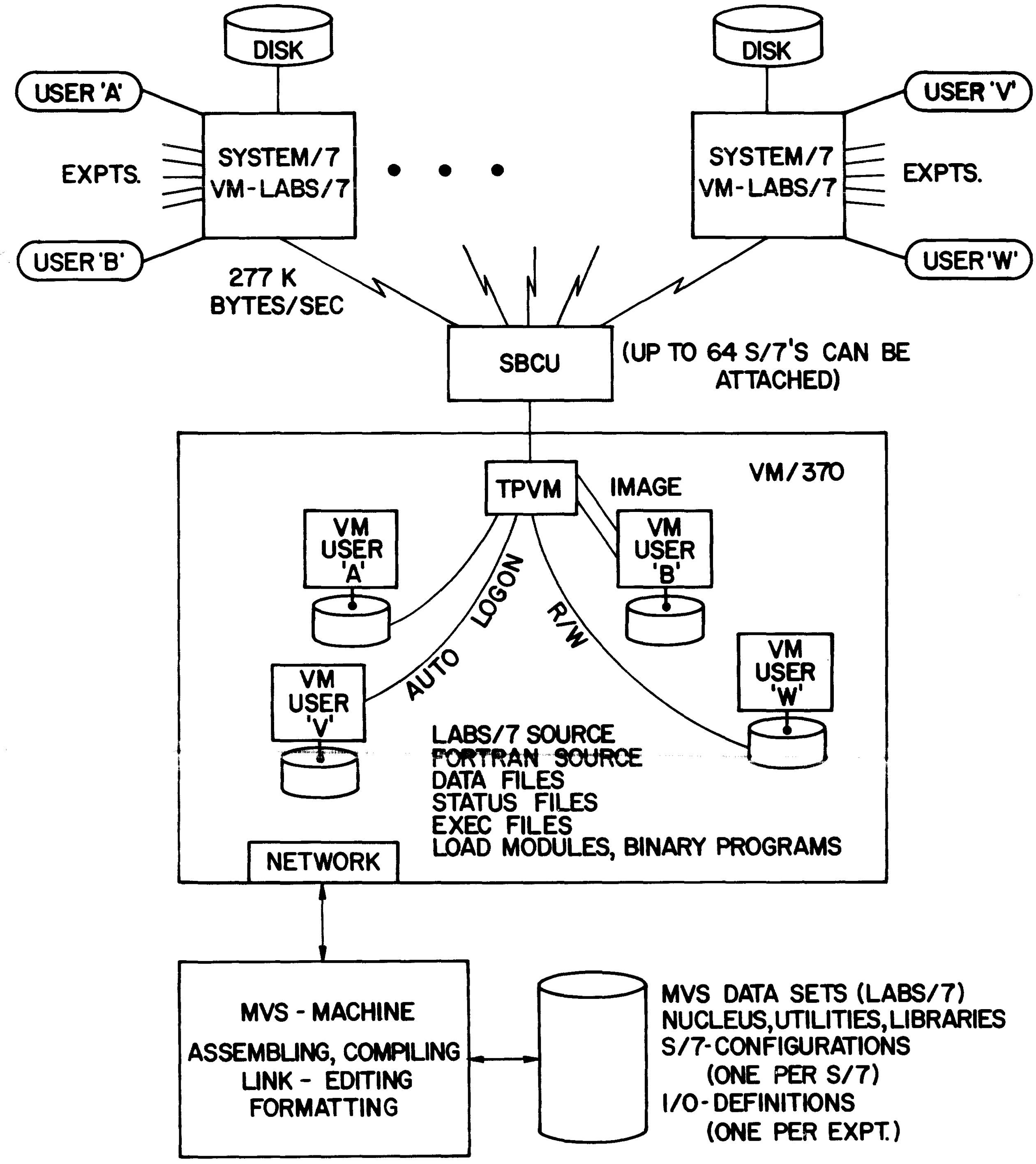


Figure 4—TPVM in an SBCU environment

SBCU as the communication between host and controllers. For the driver program, aside from having to call a different set of routines for communication, the main difference was that it now became necessary for the driver to be aware of the identity of the controller making the request and to be prepared to communicate that identity to the processing routines if required and to the communication routines for sending messages to the controller. The communications package of course was completely new. The SBCU is supported by IBM standard software under OS but not under VM. Thus it was necessary to write the basic device handling routines from scratch to work with the device under CMS. In addition, because of the possibility of requests coming from different controllers, interface routines were written to stand between the actual I/O commands and the driver program, to isolate the latter from the SBCU-dependent portions of the package.

The main modification to the processing routines was to make them capable of processing concurrent streams of requests from different controllers. They still process individual requests in the same way, with the code being executable on behalf of more than one controller without conflict. Since the processing of requests is what the TPVM is all about, let us begin by looking at these routines in greater detail.

Requests from the controller are identified by a halfword function code that is transmitted to the host. In TPVM for each distinct function code, there is a distinct processing routine. These routines are clearly delineated although some of them share common subroutines. Upon receipt of a request from the controller, the driver uses the function code to determine the routine to be called and passes the code and the controller identifier to the processing routine. As originally implemented for the single controller TPVM, these routines, in particular those which process stream requests, e.g., write a record in an already opened file, keep historical information (filename, filetype, record number, etc.) stored in the TPVM between calls. As long as only one stream of requests was in operation at a time this was quite appropriate.

However, in the SBCU operation, a request to write a record from one controller could easily be followed by a request to read a record for another controller. Thus some way had to be found to keep the information relevant to each controller separate. We wanted to do this with a minimum of modifications to the previous routines. We decided therefore to establish a workarea for each controller. This workarea would be an exact replica of the data area used by the single controller code, including all the constants and data areas referred to by the processing functions. The original data definitions were then used as a template by the processing programs to locate information in the workareas. The basic modification was to pass to the processing routines, in addition to the function being requested, the location of the workarea for the particular controller requesting service. The result was that each of the processing routines was able to operate on distinct data areas without having any change to the code itself. This enabled a simple extension of the single thread (uni-programming) code to the multi-thread (multiprogramming) situation.

We will here briefly summarize these processing routines. In the original implementation there were nine: OPENOUT, OPENIN, SUBMIT, READ, WRITE, CLOSE, FETCH STATUS, SET STATUS, and RELEASE STATUS. Functionally, OPENOUT, WRITE, and CLOSE form one stream request, to create a file and write data into it. Similarly, OPENIN, READ, and CLOSE form the analogous stream request for reading a file from the host. SUBMIT requests initiating of batch processing on the host machine. The three STATUS commands are related to determining the progress of the batch job and controlling some facets of its execution.

Subsequently four more function codes were defined: $LOGON, $FTCHCON, $WRTCON, and $ATTCON. These were in support of the CMS console functions being provided to the System/7 user at his System/7 console. $LOGON begins the CMS session, $FTCHCON reads a line from the user's CMS machine console output and sends it to the System/7, $WRTCON sends a line of input to the CMS machine from the System/7, and $ATTCON enables the System/7 user to provide his CMS machine with an attention interrupt from its console. These four basic functions enable the System/7 user to interact with a CMS machine exactly as if he were logged onto the system directly.

It should be pointed out here that these functions have been created without direct reference to the properties of the System/7, or indeed, of the SBCU. The input to the routines consist of a standard message block, which will be described in more detail later. This message block could be produced by any kind of communication mechanism, and, as part of the testing procedure, was read in from the virtual card reader of the TPVM.

The processing routines are all combined into a module called TPLAB7. This facilitates the sharing of subroutines with a minimum of overhead, but in no way constrains their independence or modularity.

By looking at Figure 2, one can see that the driver program is the central control and dispatcher for the TPVM. When the TPVM is initiated, the DRIVER program reads input parameters (number and identities of controllers) and calls routines to allocate (GETMEM) and initialize the workspaces (SBCUIN), initialize the SBCU device and interrupt handler (SBCUIN), activate the extended message facility (SPYINI), and wait for a request from one of the controllers (GETREQUE).

If the request originates in an IBM System/7, the data will be recorded in ASCII code. When the request comes in and is passed to the DRIVER program, it determines whether translation is required by examining the function code being requested. If translation is required, the FINDIS routine is called to carry it out. Then the function code is used to identify the appropriate processing routine, and the appropriate entry point to TPLAB7 is called. On return from the processing routine, the DRIVER program calls the MESSAG routine to send back to the controller the results of the request. The results can be either a simple return code indicating the outcome or a buffer full of information if that was what was requested. Upon completion of the call to MESSAG, the DRIVER program calls the GETREQUE routine again to retrieve the next request for service. And so it goes, round the loop, round the clock.

SBCU INTERRUPTION HANDLER

As mentioned above, the SBCU required a complete set of I/O routines to be written in the CMS environment. By having the SBCU attached to the TPVM as a dedicated device, we were able to have our own channel programs executed with a minimum of CP control program intervention. The SBCU is a hardware modification of the 2841[17] control unit, which was used to control 2311[18] disk drives. By specifying the SBCU as a 2311 in the VM system generation, we were able to get the appropriate control blocks created.

It would be appropriate at this point to describe the operation of the SBCU briefly. Up to 64 controllers can be attached to this device. The control unit polls the controllers looking for requests for service. If it finds one, it signals the host CPU with an attention interruption. At the same time, stored in the control unit is a request block, which identifies the issuing controller and contains a 16-byte message from the controller. Upon detection of the interruption, which suspends the polling of the control unit, the host CPU issues a Read Request Block command, to bring the contents of the request block into storage. After that is done, different commands may be issued. The hardware polling may be resumed; specific controllers may be added to or deleted from the polling list; a message may be sent to a controller which is waiting for one; a special alert code may be sent to a controller which is not expecting a message; a message may be read from a controller which is prepared to send one to the host. These are the basic functions available to the software in processing the requests for service from the controllers. One special kind of message that may be sent to a System/7 is an executable System/7 core image. This is in response to the "IPL Host" request from the controller.

The operation of the interruption handler is to respond to interrupts from the SBCU and attempt to classify them as to type, and identify for the GETREQUE routine the type of I/O operation which is required next. Because of the design of VM/CMS, it is not advisable to initiate further I/O operations from within an interruption handling routine. Therefore, a signal byte is set by SBCUIN and interrogated by GETREQUE to select the next operation.

The action to be taken is based on the contents of the function byte in the Request Block received from the controller, or from the type of interruption as determined from examining the status bytes. In addition to the interruption handling routines, there are a set of I/O initiating routines which are called by GETREQUE and MESSAG to issue whatever channel commands are needed. These include Read Request Block (READREQ), Read From Controller (READCTL), Write To Controller (WRTCTL), Send Special Alert (CPUCALL), Write Initial Program Load To Controller (WRTIPL), Resume Polling (POLL), Issue Sense Command After Error (SENSSBCU), and Signal Invalid Request To Controller (INVALREQ).

In following a typical transaction from request to fulfillment, one would observe a sequence like this. First the SBCU is polling the controllers. One controller, number 2 perhaps, signals a message to be sent to the host. The SBCU signals the host via an attention interruption. The interruption handler identifies the interruption and interprets it correctly as indicating there is a request block in the control unit waiting to be read. The communication byte is marked to indicate this. The GETREQUE routine gets control because it was waiting for the interruption, and issues the call to READREQ. The request block is then read. The interruption handler examines the request block function code to determine whether more I/O operations are required immediately. For instance, if the request is to open a file, there will be a message already waiting in the controller containing the identity of the file to be opened. If there is another operation required, the communication byte is modified to indicate 'Read from Controller Needed'. Once again the GETREQUE receives control and examines the communication byte. In this case a call is made to READCTL. The successful completion of this operation is signalled as an interruption with 'Channel End and Device End' status indicated. This information is used to indicate that the I/O operations are finished for the moment and processing of the request can commence. At this point the hardware polling is resumed with the controller waiting for service having been removed from the polling list.

When the processing has been completed, control passes to the MESSAG routine to request communication of the results to the originating controller. In our example of opening a file, the only information returned is a two-byte result code which indicates either successful completion or the reason for failure. This is transmitted via the Send Special Alert function, CPUCALL. This routine is called by MESSAG with the controller ID, and the two-byte code as parameters. Upon receipt of the interruption which indicates the completion of this operation, the requesting controller is returned to the polling list and polling is resumed.

TPLAB7 PROCESSING MODULE

The module TPLAB7 provides 13 different functions in response to requests from the controllers. These have been enumerated above and will be discussed here in some detail.

OPENOUT is a function which opens a file on a user's CMS minidisk. The user supplies from his System/7 a userid, a filename, and a filetype; the file is presumed to reside on his 191 disk, the standard address for user files. The OPENOUT routine first determines if the file being requested is for the same userid as the TPVM itself. If not then LINK must be made via CP to the disk of the user owning the file, with WRITE access. The password required is kept in a file in the TPVM's own CMS filespace. After a successful LINK the disk is accessed, and made available for subsequent processing. Next the user's file is OPENed for output using standard CMS access mechanisms. When this is completed, the function is complete and the message code is set to reflect the result. The OPENOUT routine then returns to the DRIVER program. Note that it is possible for a user to create a new file or to add to an existing one using the OPENOUT request.

The processing for OPENIN is similar except that READ access only is required to the user's filespace, and of course the file in question must already exist.

The READ request simply results in a buffer being filled with the requested information by an ordinary CMS FSREAD command. Then the READ routine returns to the DRIVER with its completion code set. Because the READ request is part of a stream request, the information as to user, filename, and filetype is preserved in the TPVM dataspace from the OPENIN through all successive READ's to the final CLOSE.

The WRITE request is again similar to the READ, except that the information is obtained from the buffer received from the controller and written to the user's CMS file via the FSWRITE instruction.

The CLOSE request simply closes whatever file is currently open, if any.

The SUBMIT function prepares to have a user's virtual machine logged on to carry out some processing requested by the user. This function presupposes that the desired processing will have been previously expressed as a command procedure, or EXEC file, in CMS parlance, and that this file resides either on the user's own filespace or on a system library filespace to which all users of this package are linked at Logon time. The contents of the request from the System/7 in this case are the USERID, the name of the requested EXEC procedure, and any parameters to be input to it. First, the existence of the requested procedure either on the user's own filespace or on the system library disk is verified. Next a special file on the user's filespace is created. This file which is itself an EXEC procedure consists of one line, namely a command to execute the procedure requested by the user with the parameters supplied from the System/7. This special file is executed whenever the user's virtual machine is logged on by the TPVM. When the preparations are complete, the user's virtual machine is logged on by CP in response to the command 'CP AUTOLOGON'. Thereafter the processing of the user's request proceeds in his own virtual machine and is charged to his account. Upon successful AUTOLOGON the SUBMIT processor exits to the DRIVER program with an appropriate return code.

Communication between the user's System/7 program and his task executing in his own CMS virtual machine takes place through special files called STATUS files. The task running in the virtual machine, by convention, has as its last step the creation of a file of filetype STATUS and filename the same as the procedure name being executed. Into this file is written a record reflecting the result of the execution of the procedure. The System/7 program interrogates the TPVM for the existence and contents of the STATUS file via the FETCH STATUS and RELEASE STATUS service requests. The RELEASE STATUS request additionally erases the STATUS file after it has been read. In addition the System/7 program can communicate information to the host programs by creating such a STATUS file via the SET STATUS request.

The operation of the FETCH STATUS and RELEASE STATUS requests is similar, except that since RELEASE erases the file it requires WRITE access to the user's filespace. The routines receive the USERID and the procedure name (PROCNAME) as input parameters. First the user's filespace is linked to and accessed with the appropriate access (READ or WRITE). Next the file "PROCNAME STATUS" is OPENed for input. If this is successful, a READ is done to retrieve the contents, and the file is CLOSEd. Then if it is a RELEASE request, the file is erased. Return is then made to the DRIVER program with a return code and the contents of the file in place for transmission to the System/7. If the STATUS file does not exist, then this is an indication that the requested procedure has not as yet completed execution. This is signalled by a different return code, and exit is again taken.

The SET STATUS request is the equivalent of an OPENOUT, a WRITE, and a CLOSE. The parameters are USERID and procedure name (PROCNAME), and the file created is "PROCNAME STATUS" on the user's filespace.

The preceding nine requests constitute the basic complement of the initial implementation of the TPVM-System/7 controller package. We will now proceed to the last, and probably most complex development, the provision of full CMS console function at the System/7 console.

REMOTE CONSOLE INTERACTIVE FUNCTION

As mentioned above, this development makes use of work done in VM by Alex Chandra and his group in the Computer Sciences Department at the Research Center. This work gives certain authorized virtual machines the ability to create (activate) other virtual machines as images of users already in the directory, but using different, unique USERID's. These other virtual machines operate as if they were interacting with an ordinary CMS console, but instead receive their console input from, and direct their console output to, the creating machine via a new CP extended external interruption facility.

Briefly, external interruptions are a class of System/370 interruptions which are not specifically related to events going on within the processor at the time the interruption occurs. They ordinarily include such events as timer signals, the depression of the "Interrupt" and "System Restart" keys of the system console, and direct CPU-CPU signals in a multiprocessor environment. Chandra and his colleagues extended the VM/370 simulation of such interruptions to include a new type created and reserved for generalized communication between virtual machines; in a sense it is a logical extension of the Write Direct feature of the multiprocessor hardware environment.

The mechanism was originally conceived as a tool for measurement and observation of virtual machines carrying out prescribed tasks. The CMS console of the creating machine provided console input not only to the creating machine but to the image machines as well. Similarly output from the image machines went directly to the creating machine.

It seemed to us that, with this facility already available, if we authorized TPVM to create such image machines for users at remote controllers, it would take relatively little additional work to transfer the console messages from the image machines to the remote controller rather than display them at the TPVM console. We decided to proceed in this direction. We were able after some investigation to reduce the process of creation of an image machine, and interaction with it to four primitives, which together would provide the full range of communication between remote controller console and image machine running on VM. These were:

1. start an image machine ($LOGON),
2. retrieve a message from the image machine ($FETCH),
3. send a message to the image machine ($WRITE), and
4. send a console attention interruption to the image machine ($ATT).

The standard process would consist therefore of a $LOGON request, followed by a number of $FETCH requests until all messages written to the console at LOGON time were retrieved, then an open keyboard presented to the remote controller keyboard, and a $WRITE request sent to the host with whatever the remote user had entered, followed by another series of $FETCH's until all the messages generated had been retrieved. In addition, the remote controller program would be interruptible to send a $ATT request so that the user could interrupt processing in the image machine via a simulated console attention interruption. Such a $ATT would be followed in general by a series of $FETCH's again until no more messages remained to be read. The remote user would conclude his session in the usual way by typing LOGOFF and the image machine would terminate in the usual way. This is the scenario we envisioned for the interactive process.

Programs were of course prepared for the remote controller to handle the interaction with the user at that end. We will concentrate here on the additions to the TPVM to carry out the image machine functions. First the number of valid function codes, which had previously ranged from 1 to 10 was extended to 15 to include the four new functions, plus one to grow on. This merely required modification of the error checking to accept codes as high as 15, modification of the driver program to select the appropriate routines for processing these new requests, and the creation of the new processing routines. Because of the hierarchical structure and modularity of the original design, previously existing processing routines required no modifications whatever to handle these new functions. The new routines were simply inserted into the TPLAB7 processing module as independent components. The new functions were coded as general purpose subroutines, not dependent in any way on the TPVM environment for their operation. The idea was that they should be available to anyone wishing to design a package accepting input from any source and using it to create and communicate with image machines.

There were several considerations kept in mind as we designed this package. For one thing to avoid a proliferation of image machines for a given user, we decided to allow no more than one image per user per controller. This was achieved by forming the image machine USERID from a unique combination of the user's normal CMS USERID and the controller identifier. Also we wanted to maintain the communication as continuously as possible across failures of both System/7 and the TPVM in the host. To this end, the identities of active image machines are maintained in the TPVM in a threaded list, each entry containing information necessary to identify the image machine, its corresponding controller (controller ID), and any outstanding message still to be processed.

If there is a TPVM failure and this table is lost, at the next request from the remote controller for communication with a particular image machine, after the TPVM has been reinitialized, a query is issued by TPVM to see if that machine is still logged on. If so, the table entry is re-created and communication resumes.

The structure of the Chandra modifications to CP and CMS allows the special external interruptions to be queued in the CMS External Interruption Handler, for retrieval on request by the CMS processing program. In fact they can be retrieved by specific image machine ID, as well as simply in the order in which they arrived. A number of image machine oriented subroutines have been written for the use of the $LOGON, $FETCH, $WRITE, and $ATT processing programs.

The ADDUSER routine, called by $LOGON, carries out the setting up of entries in the linked list and calls a CREATE routine, which issues the CP commands necessary to have an image machine created with a specific image ID, and establish the communication link between this machine and its creator, the TPVM.

The READUSER routine is called by $FETCH to determine whether there are any messages already queued for the particular image machine requested (either output from the image machine or a request for input to it). If there is a message, the entry includes a message identifier which is used by the CP facility to retrieve the specific message from its message queue, and place the contents in a specified buffer in the TPVM. If there are no messages queued, READUSER queries the system to see if the image machine is still logged on. If it is, a code is returned to indicate that there is nothing pending. This would happen if the image machine was compiling a program and had no output for some period of time. If the image machine is not logged on, a different code signals this fact. The corresponding entry in the linked list is deleted and the interaction is terminated. Finally, the last thing that READUSER can determine is that there is a request for input from the image machine pending. This is communicated by still another return code. This code when communicated to the remote controller normally results in a $WRITE request followed by a call to WRITUSER, an entry point in the READUSER routine. WRITUSER does the same list search to find the message identifier which is then used together with the message supplied to the routine as a parameter to reply to the image machine's request for input.

As mentioned earlier, the one additional function required to effectively simulate the virtual machine console at the remote controller console is the ability to generate ATTENTION interruptions at the image machine. The $ATT routine is called with a parameter of either 1 or 2 depending on whether one wants the attention to take one into CMS or CP on the image machine. The appropriate CP command developed by Chandra is issued and the attentions are sent.

We will conclude the discussion of the TPVM internals by a look at the communications interface routines, GETREQUE and MESSAG. These routines form the link between the driver program and the specific I/O routines for handling the communications link to the remote controllers. We have spoken briefly about GETREQUE earlier, and will do so at slightly greater length here.

GETREQUE is called by DRIVER after all initialization is complete, and is intended to return with a request, designated as a controller ID and a function code. It comes by these in the following way. Upon entry it immediately issues a WAIT on the communications device, in our case the SBCU. When an interruption is received by the Interruption Handler and is processed, this WAIT is satisfied. The GETREQUE routine is activated and proceeds to determine what action is required of it. First it examines the signal set by the interruption handler to see if there is more input/output to be done. The interruption handler does the analysis and leaves an explicit setting in a communication byte which GETREQUE interprets. GETREQUE then calls the particular I/O routine needed (READREQ, READCTL, etc.) and returns to the WAIT. When all the information needed to process a request has been obtained, as signalled by the interruption handler, it remains for GETREQUE to identify the controller and the request. A table is maintained by the interruption handler containing an entry for each active controller. This entry contains some flags, the function code of the last request received from this controller, and the address of the workarea for this controller. GETREQUE examines the flags for each entry till it finds one which indicates that I/O is complete, and the request is ready for processing. Having found one such, the controller ID of the requesting controller, and the function code are returned to the DRIVER program.

At the other end of the processing cycle is the MESSAG routine which controls the transmission of results to the remote controller. The DRIVER program, after completion of the processing of the request, calls the MESSAG routine with the controller ID and the function code. The MESSAG routine takes appropriate action depending on the function code to call I/O routines to transmit the results to the controller. For all requests the special alert (CPUCALL) function is used to transmit the two-byte completion code which results from all requests. If in addition information is to be sent, as for instance in response to a READ request from the controller, the CPUCALL is followed by a call to the Write to Controller function to send the data back to the waiting controller.

This concludes the discussion of the TPVM internals. The point of the design has been from the beginning to create a highly structured modular package with intelligible and independent code. The modularity of the design affords a great degree of flexibility in that a change in the implementation of one function, such as the communications routines, has no effect on the processing routines, and vice versa. For instance, the halfword allowed by the hardware for function codes provides up to 32,768 different possible codes. The implementation of the DRIVER program and others means that the processing routines for some additional codes (perhaps not 30,000 of them) could be added in a very simple straightforward way. The interfaces between the various levels for the most part are data content oriented rather than a very involved structural construct.

TPVM RESPONSE TIME

Measurements made, using the one millisecond resolution timer in the S/7, indicate that a complete trivial operation, that is, a blank line followed by a carriage return on the terminal, from time of request to end of response can be performed in as little as seven (7) milliseconds. These measurements were made by modifying the $LOGON utility within the S/7 to extract the time of day when a keyboard entry was completed and ready for transmission to the user VM system. When the response to the keyboard request was received by $LOGON the time of day clock was again read and the difference between clock times recorded. These measurements were made for many transactions. Typically for interactive operations, such as listing or typing a file an average of 40 milliseconds per transaction was recorded. Note that the TPVM mechanism was in no way operating as a privileged machine. Its dispatching and presence in the queues was that of any ordinary VM/CMS user machine.

USAGE DESCRIPTIONS

Thus far in this paper we have described the TPVM implementation and its facilities. We now turn our attention to some of the user experiments for which TPVM was originally created. A partial listing includes:

- superlattice fabrication of thin film devices,
- x-ray scattering measurements,
- ellipsometry,
- surface interface measurements of MOS capacitors,
- conductivity measurements of sodium ions, and
- Auger spectrometry.

Rather than describing each of these experiments, we will illustrate, with an experiment designed to control an Auger spectrometer,[19] the interplay between the user System/7 (S/7) and the TPVM mechanism.

COMPUTER-CONTROLLED AUGER SPECTROMETER

For those unfamiliar with the technique, an Auger spectrometer is a device which permits the scientist to analyze thin films and surfaces using electron beams. The Auger spectra are generally measured in a derivative form and then doubly integrated to obtain the electron energy distribution curve.

Historically, the experiment was manually operated with a technician setting up the various spectrometer controls. Spectra data were then recorded onto an X-Y plotter; only a single sweep could be made per measurement. The data were then examined visually to discern the peak heights of the various constituents. Manual calculations were then performed to permit re-plotting of the reduced data. As a result of this traditional method quantitative rather than qualitative analysis is achieved.

While the 'automation' of this experiment, control and observation, could be achieved by coupling to a stand-alone mini-computer, the laboratory automation functions are far greater than mere device control and data capture; data analysis is of prime concern. Program preparation and execution, generation of reports, and data storage for historical purposes are of major importance. This is best illustrated by Figure 5. Attempting to perform all these tasks within a single dedicated computer becomes difficult to achieve and may be quite costly in manpower and capital.

An hierarchical system as described in References 2, 3, and 4 provides for interface standardization via a dedicated mini-computer, the IBM System/7, and a large VM host. The mini-computer is a bare bones unit with a minimum of peripherals—in our case solely a terminal and disk—dedicated to serving several experiments simultaneously for data acquisition and device control. It performs the real-time tasks—those not possible with batch or time-sharing systems. Our users tend to view the S/7 in this hierarchy as an intelligent controller with a transparent link to a higher much more powerful computing system, VM/370. TPVM provides the link and brings to the user the full data processing facilities of a large host such as editors, compilers, assemblers, and the full range of peripherals. The Auger experiment is but one of many experiments sharing the System/7, TPVM, and the facilities of the host.

As shown in Figure 6, the Auger spectrometer is controlled by an IBM System/7 sensor based computer, which is coupled to the TPVM mechanism, residing in a VM/370 Model 168, via a 277K byte teleprocessing link.

The S/7 controls the real-time requirements of the Auger spectrometer, i.e., the control, measurements, calibration, and real-time decision making (e.g., process interruption or variation in operational sequence based upon externally measured parameters).

The spectra data are then transferred via the teleprocessing link to user filespace in the 370/168 VM system by TPVM. In the event that the host VM system is not operational, the System/7 with its disk storage can be operated independently. The System/7 can control experiments and accumulate substantial amounts of data in its own storage. It thus provides the experimenter with the security that he can leave his experiment in operation without concern for the interruptions in service, scheduled

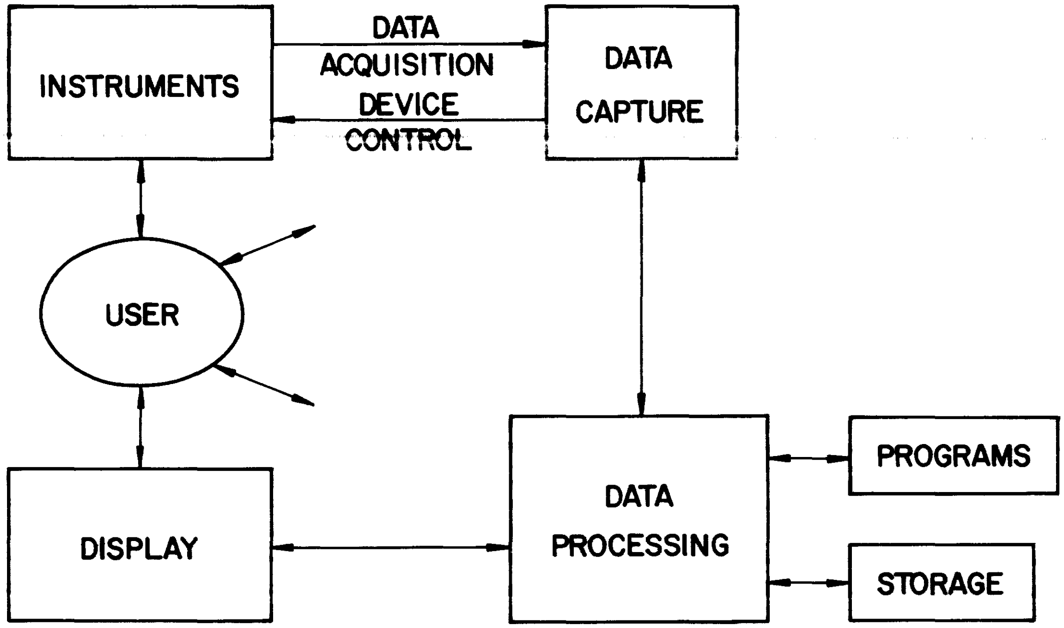


Figure 5—Functions required for laboratory automation

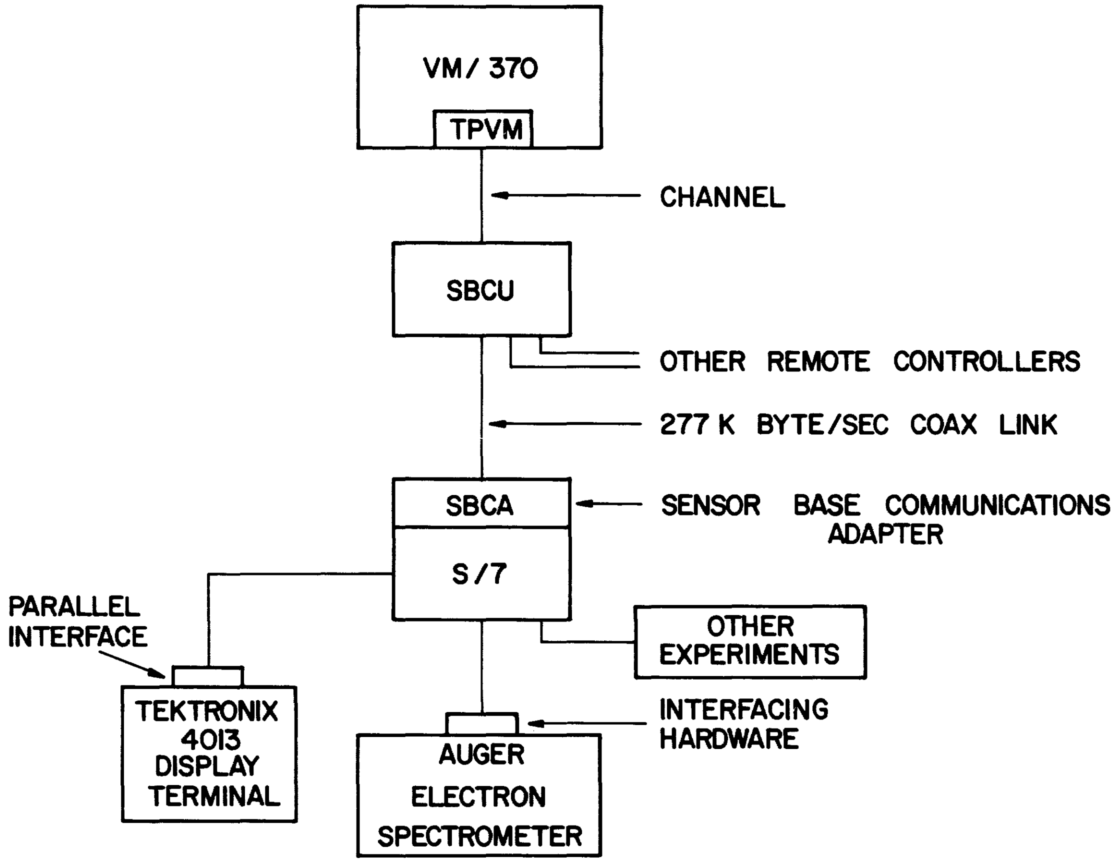


Figure 6—Auger spectrometer experiment configuration

or otherwise, on the host system. When the TPVM is reactivated after an interruption, it signals the controllers of its presence. This signal is interpreted by the controllers as a request for them to reissue any requests for teleprocessing that are outstanding.

Thus while it is generally necessary to have the host available for program preparation, once an experiment is set up according to the experimenter's specifications, and the control programs written and in place on the System/7, the experiment itself can proceed independently of the host machine. It is also possible to prepare programs to do some preliminary analysis on the System/7, while reserving the bulk of the analysis for the host machine.

Having transferred the data, the user may select background batch mode or foreground interactive mode for data reduction, e.g., time averaging, background subtraction, double integration, etc. on the VM facility.

For background batch mode, the user issues a SUBMIT request from the S/7 to TPVM. The SUBMIT request is comprised of an operation code and a message. Messages consist of a USERID, an EXEC procedure name and parameters.

The EXEC procedure specified contains the commands to carry out the desired processing. TPVM logs on the user VM/CMS machine via a CP AUTOLOGON, and causes the requested EXEC procedure to be executed. While the user's CMS machine is executing on his behalf, the controller can continue to carry out his interactions (other than submit requests) with the TPVM.

The execution of the EXEC procedure proceeds as follows. Each directory entry contains a request for automatic IPL of CMS. The user PROFILE procedure contains a test which determines whether the logon was performed with or without the user terminal. For AUTOLOGON a branch is taken which requests CMS to execute $SBMTL7 which in turn invokes the user specified procedure issued with the SUBMIT command. The user program will then execute with the desired parameters.

When the user machine has completed its assigned task, it returns to the user EXEC procedure with a return code. The EXEC procedure creates a STATUS file with the name of the EXEC procedure and then logs off the user machine. Periodic interrogation by the user S/7 program via the FETCH/RELEASE requests will retrieve the STATUS file after it has been created. The results of the computation and reduction are brought back to the S/7 by TPVM for display via the graphics terminal coupled to the S/7.

For interactive operation mode, the user issues a $LOGON request from the S/7 console and all succeeding steps are those for a user logging on directly to the VM host. Now the data reduction, which can be accomplished in any language available within the VM host, is performed interactively and the results immediately displayed at the console.

To summarize, the entire system, i.e., the Auger spectrometer, the S/7, the interactive graphic terminal, TPVM and the VM host, is used as a unique, integrated, and powerful instrument for Auger spectroscopy.

CONCLUSIONS

The TPVM mechanism was designed as an integral part of a hierarchical approach to the support of laboratory automation, or computer augmented experimentation. The thrust of the approach has been to allocate the functions required for this process to different computers, based on matching the characteristics of the computer to the function required. Thus the intelligent controller is assigned the tasks requiring dedicated resources, rapid response, and high reliability. The larger host system supplies the large data storage and processing resources, the facilities for program preparation, and other user-oriented features of large general-purpose interactive computing systems (for example, high quality text editing facilities). The TPVM is the basic means of coupling these controllers in an efficient way to the particular host system we have chosen, VM/370.

The applications mentioned herein have been on System/7's communicating with VM/370 and TPVM at the Yorktown Research Center. However, any controller which conforms to the TPVM hardware and software interface requirements may be used.

ACKNOWLEDGMENTS

The authors wish to thank their many colleagues at the Research Center for their help and support. We are particularly grateful to P. Greier for his documentation efforts, to Drs. H. Cole, H. Freitag, and R. P. Kelisky for their encouragement and guidance, and to N. J. Chou, T. Hickmott, C. S. Cargill, L. Alexander, E. Irene, S. La Placa, K. D. Hardman, and R. Pollak, the experimenters, for their application descriptions.

REFERENCES

1. Guido, A. A., "Laboratory Automation In A Virtual Machine Environment," IBM Research Report RC3917, 1972.
2. Hultzsch, H., A. A. Guido, and H. Cole, "Laboratory Automation In A Novel Computer Hierarchy," IBM Research Report RC4714, 1974.
3. Cole, H., "System/7 in a hierarchical laboratory automation system," *IBM Systems Journal* Vol. 13, No. 4, 1974, pp. 307–324.
4. Cole, H., A. Guido, and A. Bednowitz, "Laboratory Automation—Current Status and Future Trend," *Japanese Journal for the Society of Instrument and Control Engineering,* Vol. 14, No. 10, 1975, pp. 714–724.
5. Perone, S. P., "Computer applications in the chemistry laboratory—a survey," *Analytical Chemistry* Vol. 43, No. 10, August 1971, pp. 1288–1299.
6. *IBM Virtual Machine Facility/370—Introduction,* IBM Systems Library, Order Number GC20-1800.
7. *IBM Virtual Machine Facility/370—CMS User's Guide,* IBM Systems Library, Order Number GC20-1819.
8. *IBM System/7 Sensor-Based Control Unit (SBCU)—Planning Guide,* IBM Systems Library, Order Number GC34-1522.
9. Hsih, Shirley C., "Inter-Virtual Machine Communication Under VM/370," IBM Research Report RC5147, 1974.
10. *IBM System/7—System Summary,* IBM Systems Library, Order Number GA34-0002.
11. Raimondi, D. L., et al, "LABS/7—a distributed real-time operating system," *IBM Systems Journal* Vol. 15, No. 1, 1976, pp. 81–100.
12. Raimondi, D. L., "Multiprogramming Monitor for Laboratory Automation," IBM Resarch Report RJ1075, 1972.
13. *Laboratory Applications Based System—Program Description/Operations Manual,* IBM Systems Library, Order Number SH20-1363; also, Event Driven Executive—Program Description/Operations Manual, Order Number SH20-1819.
14. *IBM 2701 Data Adapter Unit—Component Description,* IBM Systems Library, Order Number A22-6864.
15. *IBM 2702/2703 Transmission Controls—Original Equipment Manufacturers' Information,* IBM Systems Library, Order Number A27-3012.
16. *IBM 3704 and 3705 Communications Controllers—Principles of Operation,* IBM Systems Library, Order Number GC30-3004.
17. *IBM System/360—Component Description-2841 and Associated DASD,* IBM Systems Library, Order Number GA26-5988.
18. *IBM 2311 Disk Storage Drive—Original Equipment Manufacturers' Information,* IBM Systems Library, Order Number A26-3567.
19. Chou, N. J., R. Hammer, and A. Bednowitz, "Computer-controlled Auger spectrometer," *Review of Scientific Instruments,* Vol. 47, No. 5, 1976, pp. 559–564.

Computer typesetting of technical journals on UNIX

by M. E. LESK and B. W. KERNIGHAN
Bell Laboratories
Murray Hill, New Jersey

ABSTRACT

A UNIX-based system for typesetting technical papers for high-quality output was evaluated by measuring use of computer and economic resources. Five manuscripts submitted to *Physical Review Letters* were typeset at Bell Laboratories, after preparation of programs to handle the equations, tables, and layout problems of this journal.

Computerized typesetting is substantially cheaper than typewriter composition. The primary cost of page composition is keyboarding and the aids provided by UNIX to facilitate input of complex mathematical and tabular text reduce input time significantly. Typing and correcting articles on UNIX, with a single experienced typist, is between 1.5 and 3.3 times as fast as typewriter composition. Input on UNIX averaged 2.4 times as fast as conventional methods. The composition cost per camera-ready page using a full-scale UNIX-based system producing 200 finished pages per day would be about $10 per page as compared with typewriter composition costs of $30 per page.

INTRODUCTION

UNIX[1] is a general purpose time-sharing operating system for the PDP-11 family of minicomputers. It is used extensively for the preparation of technical documents, ranging from internal technical memoranda in diverse fields to patent applications and books. The basic tools that UNIX provides for this are:

(1) A file system for long-term storage of information on the computer.
(2) A text editor for input and modification of text.
(3) Formatting programs for producing output on standard ASCII terminals, line printers, or on a phototypesetter.
(4) Specialized programs for formatting mathematics and tables.
(5) Mechanical aids for proofreading and format checking.
(6) An especially convenient mechanism for connecting programs together to perform complex tasks.

The document preparation facilities of UNIX are used heavily on a daily basis by scientists, secretaries and typists, preparing material of all sorts, but with the emphasis on typesetting mathematics.

This paper describes an experiment performed to evaluate the cost and performance of UNIX as a production system for computerized typesetting of technical papers for a primary technical journal, *Physical Review Letters*. This project was done in cooperation with the American Physical Society (APS), which supplied the test materials.

TEXT FORMATTING LANGUAGE

The basic tool for document formatting is a general-purpose text formatting program called *troff*.[2] *troff* provides standard facilities for formatting text into lines, with justification, hyphenation, size and font control, and the like. In addition, it has a macro capability, text and arithmetic variables, numerical computation and testing, and conditional branching. In effect, *troff* is a programming language, albeit of an unconventional sort.

However, these programming operations are generally not used directly by typists. Instead, a higher level descriptive language is used in which the typist indicates the content of each section of the input, by typing commands such as ".TL" before the title, or ".AU" before the author name(s). *troff* programs are written to interpret these descriptive commands and generate appropriate low-level typesetter requests for each particular journal style (e.g., that of *Physical Review Letters*). The macro instructions in the layout program for each journal enforce the local style rules; for example, the title of an article in *Physical Review Letters* is centered and set in bold-face. Different journals may have different styles for this, but the article need not be retyped or even edited to print it in a different style; only the program used to interpret the commands is changed.

MATHEMATICS LANGUAGE

Within a document, mathematical expressions are written in a special language[3] which has been designed to be easy to learn and use by non-technical people like secretaries and typists. For example, in this language the display equation

$$\sum_{i=0}^{\infty} x_i \to \frac{\pi}{2}$$

is written as

```
.EQ
sum from i=0 to infinity x sub i -> pi over 2
.EN
```

The "formatting commands" .EQ and .EN signal the beginning and end of a displayed mathematical expression. In-line expressions like $\vec{x}$ are written as %x vec%, where the "%" is a user-chosen character.

Essentially all details of sizes, fonts, etc., are handled automatically by this system; handfiddling is hardly ever needed.

The mathematical parts of a document are interpreted by a separate program called *eqn*. *eqn* operates as a preprocessor which converts the mathematical parts into the rather complicated *troff* commands necessary to actually position and print the expressions properly. The non-mathematical parts of the document are passed through *eqn* untouched.

TABLE LANGUAGE

Tables within a document are handled in a fashion quite analogous to mathematics: a special language,[4] again designed to be easy to learn and use, permits quite complicated tables to be entered by typists. The program does all the computations necessary for lining up columns, leaving space for the widest elements, drawing lines, etc. For example, in this language, the input

```
.TS
center, box;
c s
c c
l n.
Weight: English to Metric
Name (tab)  Grams
pound (tab)  453
ounce (tab)  28.349
grain (tab)    0.0648
.TE
```

will produce the output

Weight: English to Metric	
Name	Grams
pound	453
ounce	28.349
grain	0.0648

As with *eqn*, a program *tbl* interprets table specifications, generating appropriate *troff* commands. *tbl* is also a preprocessor. Tables may contain mathematics; in this case, *tbl* is run into *eqn*, and the output of *eqn* is passed in turn to *troff.*

PAGE LAYOUT

APS/AIP journals normally set all pages in double column, except for very wide equations and the text surrounding them, which are set full width, i.e., a single wide column. Figures and tables are placed only at the top or bottom of columns, and may be either narrow or wide depending on their content. Lines are not justified (ragged right margin) but all columns are the same length. Figure 1 shows one of the sample pages, as produced by our system. Figure 2 shows the same page taken from the published journal. Notice the order in which the material is read.

All material is typed in with the UNIX text editor, using ordinary ASCII terminals with no special characters. The typist inserts commands in the text as needed to specify different formatting actions. The major commands are those which delimit equations, tables and figures; the identification of the title, authors, authors' institutions, and abstract; the beginning of each paragraph; and the commands "begin double column" or "begin wide single column." Figures may be marked as either wide or narrow; captions and figure size are placed in the text at the point where the figure is referenced. Of course within tables and equations there are commands appropriate for those processes. Figure 3 shows the beginning of the input used to prepare the output of Figure 1.

There are four stages in the procedure that formats a document, although the user only types one command. The UNIX system permits different programs to be linked at command level by connecting the output of one program to the input of the next program; the program network constructed is called a "pipeline." This pipeline facility is used very heavily by the typesetting system, since we could not run as one program all of the pieces of our typesetting software. Running them separately not only avoids the space limitations of our system but minimizes the interaction between the authors of the different programs.

(1) The file is preprocessed to measure the length of the various one- and two-column sections of the paper. This information is needed by the layout programs to avoid extremely short double column sections. There are two ways of performing this step: we can use a procedure that goes through the internal steps of producing galleys for the entire paper, but instead of printing the galley proofs merely records the length of each section. Usually, however, we simply count the number of characters in each section and assume a standard number of input characters per column inch. This can be done much faster by the computer and is accurate enough for papers of the style of *Physical Review Letters.*

(2) The table processing program *tbl* comes next, if the paper contains any tables. It ignores most of the manuscript, but extracts tables and arranges their

is

$$\langle\beta,b|H'|\beta',b\rangle=\langle\beta,b|H_{\rm diag}|\beta',b\rangle+\tfrac{1}{2}\sum_{\alpha}\left\{\frac{\langle\beta,b|H_{ba}|\alpha,a\rangle\langle\alpha,a|H_{ab}|\beta',b\rangle}{E_a(\alpha)-E_b(\beta)}+\frac{\langle\beta,b|H_{ba}|\alpha,a\rangle\langle\alpha,a|H_{ab}|\beta',b\rangle}{E_a(\alpha)-E_b(\beta')}\right\},\qquad(10)$$

where the labels β and α index the bonding ($|\beta,b\rangle$) and antibonding ($|\alpha,a\rangle$) eigenfunctions of $H_{\rm diag}$. The valence and conduction bands are now decoupled to order $(V_1/V_2)^2$. In lowest approximation, $[E_a(\alpha)-E_b(\beta)]=2|V_2|$ and the sum over intermediate states can immediately be evaluated. The total valence-electron energy (including magnetic-field-dependent terms) is the trace of Eq. (10) in the bonding representation. This expression contains both paramagnetic and diamagnetic terms. The former arise from the matrix elements of $H_{ab}+H_{ba}$; hence they depend upon bond angles and crystal coordination. The latter emerge from the $H_{\rm diag}$ term.

To extend this procedure to next order in (V_1/V_2), we keep higher powers of T in Eq. (8), and expand the energy denominator of Eq. (10) about the "average" gap $E_g\equiv 2|V_2|$. The result is a series of commutators of the form $H_{ba}[H_{\rm diag},[H_{\rm diag}\ldots,[H_{\rm diag},H_{ab}]]]$ which again can be evaluated with trace methods. To order $(V_1/V_2)^2$,

$$\chi=\frac{-Ne^2}{4mc^2}\sum_{j=1}^{4}\left\{\langle r_\perp^2(j)\rangle_{\rm local}+\langle r_\perp^2(j)\rangle_{\rm overlap}\left[1-\frac{3}{4}\left(\frac{V_1}{V_2}\right)^2\right]\right\}+\frac{\tfrac{1}{2}N(e\hbar/mc)^2}{E_g}\left[1+\frac{1}{4}\left(\frac{V_1}{V_2}\right)^2\right].\qquad(11)$$

Since $(V_1/V_2)<0.5$,[5] the second-order terms in Eq. (11) are small and may be ignored. Equation (11) was derived on the assumption of zero overlap between orbitals that form a bond. With overlap, the expression is modified as follows[10]:

$$\chi=\frac{-Ne^2}{4mc^2}\left(\frac{1}{1+S}\right)\sum_{j=1}^{4}\left[\langle r_\perp^2(j)\rangle_{\rm local}+\langle r_\perp^2(j)\rangle_{\rm overlap}\right]+\frac{N}{2}\left(\frac{e\hbar}{mc}\right)^2\left(\frac{1}{1-S^2}\right)\frac{1}{\tilde E_g},\qquad(12)$$

where $\tilde E_g$ is the energy gap modified for overlap. Equation (12) is our final result, which we compare with the HKF model. Their diamagnetic and paramagnetic terms can now be identified with well-defined, gauge-invariant quantities: the orbital area $\langle r_\perp^2(j)\rangle_{\rm local}$, the overlap area $\langle r_\perp^2(j)\rangle_{\rm overlap}$, the overlap integral S, and the energy gap $\tilde E_g$. $|M|^2$, which depends only on the geometrical arrangement of the bonds, is given by

$$|M|^2=(1-S^2)^{-1}(e\hbar/mc)^2N_0.\qquad(13)$$

Here N_0 is Avogadro's number. This result implies that $|M|^2$ is constant for all covalent tetrahedrally bonded materials, as observed by HKF. With the value $S=0.5$,[11] Eq. (13) gives $|M|^2=1.7\times10^{-4}$ eV cm^3/mole, in good agreement with the experimental values[1] 1.8 ± 0.6 (diamond), 1.8 ± 0.3 (Si), and 2.2 ± 0.2 (Ge) in units of 10^{-4} eV cm^3/mole.

To evaluate the diamagnetic terms in Eq. (12), we have used Herman-Skillman wave functions[12] to calculate $\langle r_\perp^2(j)\rangle_{\rm local}$. It can be shown[10] that $\langle r_\perp^2(j)\rangle_{\rm overlap}\lesssim(0.15)\langle r_\perp^2(j)\rangle_{\rm local}$. Our results[13] for $r_\perp$ agree with experimental values[1] (indicated in parentheses): diamond 0.84 Å (1.04 ± 0.15 Å), Si 1.23 Å (1.32 ± 0.1 Å), Ge 1.25 Å (1.48 ± 0.06 Å).

In conclusion, we have derived a particularly simple expression for the susceptibility of tetrahedral semiconductors in terms of gauge-invariant quantities characterizing the chemical bonding and the spatial structure of the solid. Work is presently under way to extend our formalism to differently coordinated solids and to amorphous materials.

It is a pleasure to thank Professor M. Kastner and ~~Dr. S. Hudgens for stimulating our interest in this~~ problem, and for many helpful suggestions and discussions.

* Research supported in part by the U. S. Air Force Office of Scientific Research, Air Force Systems Command, under Contract/Grant No. AFOSR-71-2010.

† IBM Predoctoral Fellow. Present address: Vanderbilt Hall, Harvard Medical School, Boston, Mass. 02115

[1] S. Hudgens, M. Kastner, and H. Fritzsche, Phys. Rev. Lett. 33, 1552 (1974).

[2] J. C. Phillips, Rev. Mod. Phys. 42, 317 (1970).

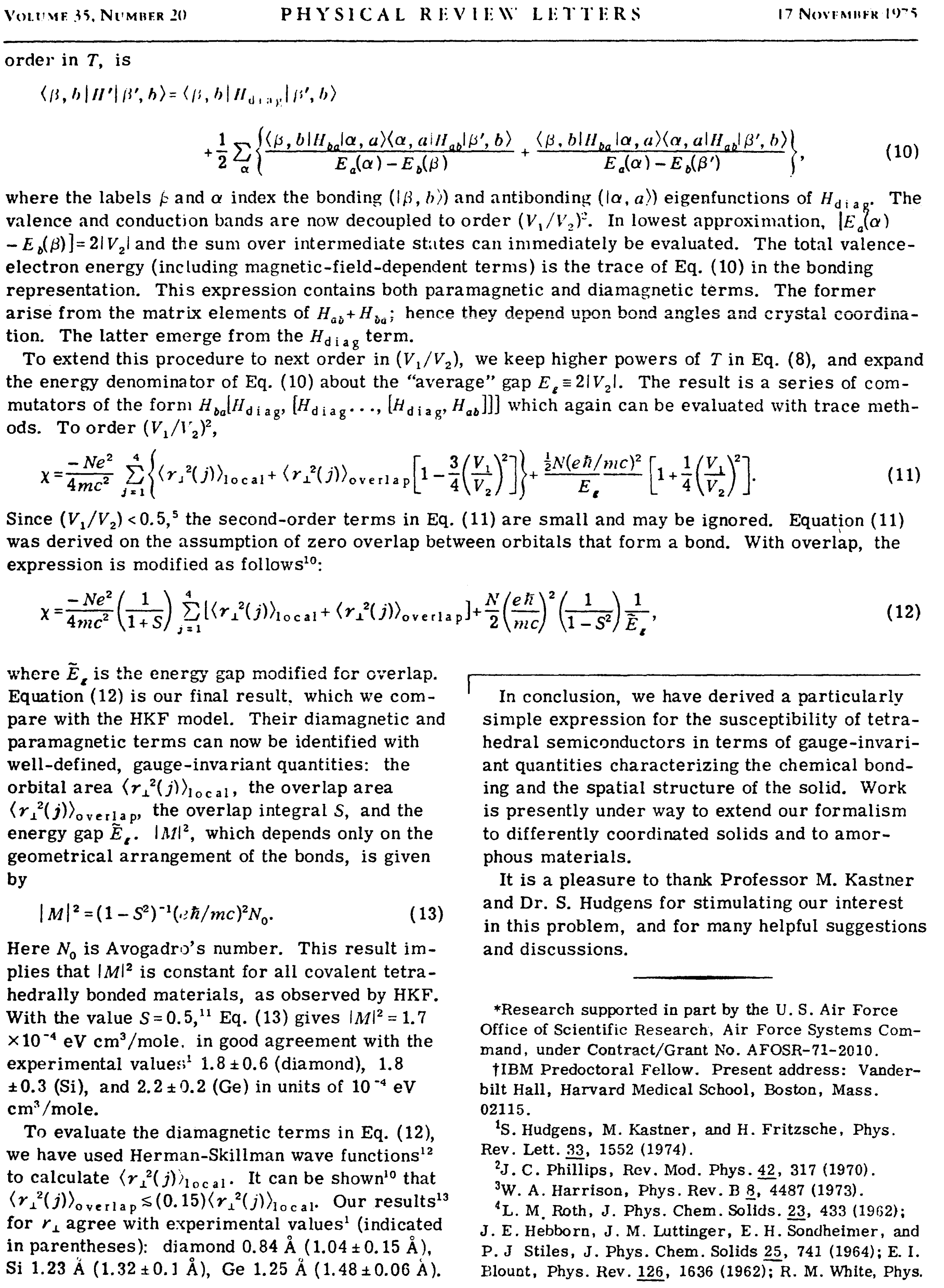
[3] W. A. Harrison, Phys. Rev. B 8, 4487 (1973).

[4] L. M. Roth, J. Phys. Chem. Solids 23, 433 (1962); J. E. Hebborn, J. M. Luttinger, E. H. Sondheimer, and P. J. Stiles, J. Phys. Chem. Solids 25, 741 (1964); E. I. Blount, Phys. Rev. 126, 1636 (1962); R. M. White, Phys. Rev. B 8, 3426 (1974).

[5] D. Weaire and M. F. Thorpe, Phys. Rev. B 4, 2508 (1971).

[6] G. G. Hall, Philos. Mag. 43, 338 (1952); and 3, 429 (1958).

Figure 1



order in T, is

$$\langle \beta, b|H'|\beta', b\rangle = \langle \beta, b|H_{\rm diag}|\beta', b\rangle + \frac{1}{2}\sum_{\alpha}\left\{\frac{\langle \beta, b|H_{ba}|\alpha, a\rangle\langle \alpha, a|H_{ab}|\beta', b\rangle}{E_a(\alpha)-E_b(\beta)} + \frac{\langle \beta, b|H_{ba}|\alpha, a\rangle\langle \alpha, a|H_{ab}|\beta', b\rangle}{E_a(\alpha)-E_b(\beta')}\right\}, \tag{10}$$

where the labels β and α index the bonding ($|\beta, b\rangle$) and antibonding ($|\alpha, a\rangle$) eigenfunctions of $H_{\rm diag}$. The valence and conduction bands are now decoupled to order $(V_1/V_2)^2$. In lowest approximation, $|E_a(\alpha) - E_b(\beta)] = 2|V_2|$ and the sum over intermediate states can immediately be evaluated. The total valence-electron energy (including magnetic-field-dependent terms) is the trace of Eq. (10) in the bonding representation. This expression contains both paramagnetic and diamagnetic terms. The former arise from the matrix elements of $H_{ab}+H_{ba}$; hence they depend upon bond angles and crystal coordination. The latter emerge from the $H_{\rm diag}$ term.

To extend this procedure to next order in (V_1/V_2), we keep higher powers of T in Eq. (8), and expand the energy denominator of Eq. (10) about the "average" gap $E_g \equiv 2|V_2|$. The result is a series of commutators of the form $H_{ba}[H_{\rm diag}, [H_{\rm diag} \cdots, [H_{\rm diag}, H_{ab}]]]$ which again can be evaluated with trace methods. To order $(V_1/V_2)^2$,

$$\chi = \frac{-Ne^2}{4mc^2}\sum_{j=1}^{4}\left\{\langle r_\perp^2(j)\rangle_{\rm local} + \langle r_\perp^2(j)\rangle_{\rm overlap}\left[1-\frac{3}{4}\left(\frac{V_1}{V_2}\right)^2\right]\right\} + \frac{\frac{1}{2}N(e\hbar/mc)^2}{E_g}\left[1+\frac{1}{4}\left(\frac{V_1}{V_2}\right)^2\right]. \tag{11}$$

Since $(V_1/V_2) < 0.5$,[5] the second-order terms in Eq. (11) are small and may be ignored. Equation (11) was derived on the assumption of zero overlap between orbitals that form a bond. With overlap, the expression is modified as follows[10]:

$$\chi = \frac{-Ne^2}{4mc^2}\left(\frac{1}{1+S}\right)\sum_{j=1}^{4}[\langle r_\perp^2(j)\rangle_{\rm local} + \langle r_\perp^2(j)\rangle_{\rm overlap}] + \frac{N}{2}\left(\frac{e\hbar}{mc}\right)^2\left(\frac{1}{1-S^2}\right)\frac{1}{\tilde{E}_g}, \tag{12}$$

where $\tilde{E}_g$ is the energy gap modified for overlap. Equation (12) is our final result, which we compare with the HKF model. Their diamagnetic and paramagnetic terms can now be identified with well-defined, gauge-invariant quantities: the orbital area $\langle r_\perp^2(j)\rangle_{\rm local}$, the overlap area $\langle r_\perp^2(j)\rangle_{\rm overlap}$, the overlap integral S, and the energy gap $\tilde{E}_g$. $|M|^2$, which depends only on the geometrical arrangement of the bonds, is given by

$$|M|^2 = (1-S^2)^{-1}(e\hbar/mc)^2N_0. \tag{13}$$

Here N_0 is Avogadro's number. This result implies that $|M|^2$ is constant for all covalent tetrahedrally bonded materials, as observed by HKF. With the value $S = 0.5$,[11] Eq. (13) gives $|M|^2 = 1.7 \times 10^{-4}$ eV cm^3/mole, in good agreement with the experimental values[1] 1.8 ± 0.6 (diamond), 1.8 ± 0.3 (Si), and 2.2 ± 0.2 (Ge) in units of 10^{-4} eV cm^3/mole.

To evaluate the diamagnetic terms in Eq. (12), we have used Herman-Skillman wave functions[12] to calculate $\langle r_\perp^2(j)\rangle_{\rm local}$. It can be shown[10] that $\langle r_\perp^2(j)\rangle_{\rm overlap} \le (0.15)\langle r_\perp^2(j)\rangle_{\rm local}$. Our results[13] for $r_\perp$ agree with experimental values[1] (indicated in parentheses): diamond 0.84 Å (1.04 ± 0.15 Å), Si 1.23 Å (1.32 ± 0.1 Å), Ge 1.25 Å (1.48 ± 0.06 Å).

In conclusion, we have derived a particularly simple expression for the susceptibility of tetrahedral semiconductors in terms of gauge-invariant quantities characterizing the chemical bonding and the spatial structure of the solid. Work is presently under way to extend our formalism to differently coordinated solids and to amorphous materials.

It is a pleasure to thank Professor M. Kastner and Dr. S. Hudgens for stimulating our interest in this problem, and for many helpful suggestions and discussions.

*Research supported in part by the U. S. Air Force Office of Scientific Research, Air Force Systems Command, under Contract/Grant No. AFOSR-71-2010.

†IBM Predoctoral Fellow. Present address: Vanderbilt Hall, Harvard Medical School, Boston, Mass. 02115.

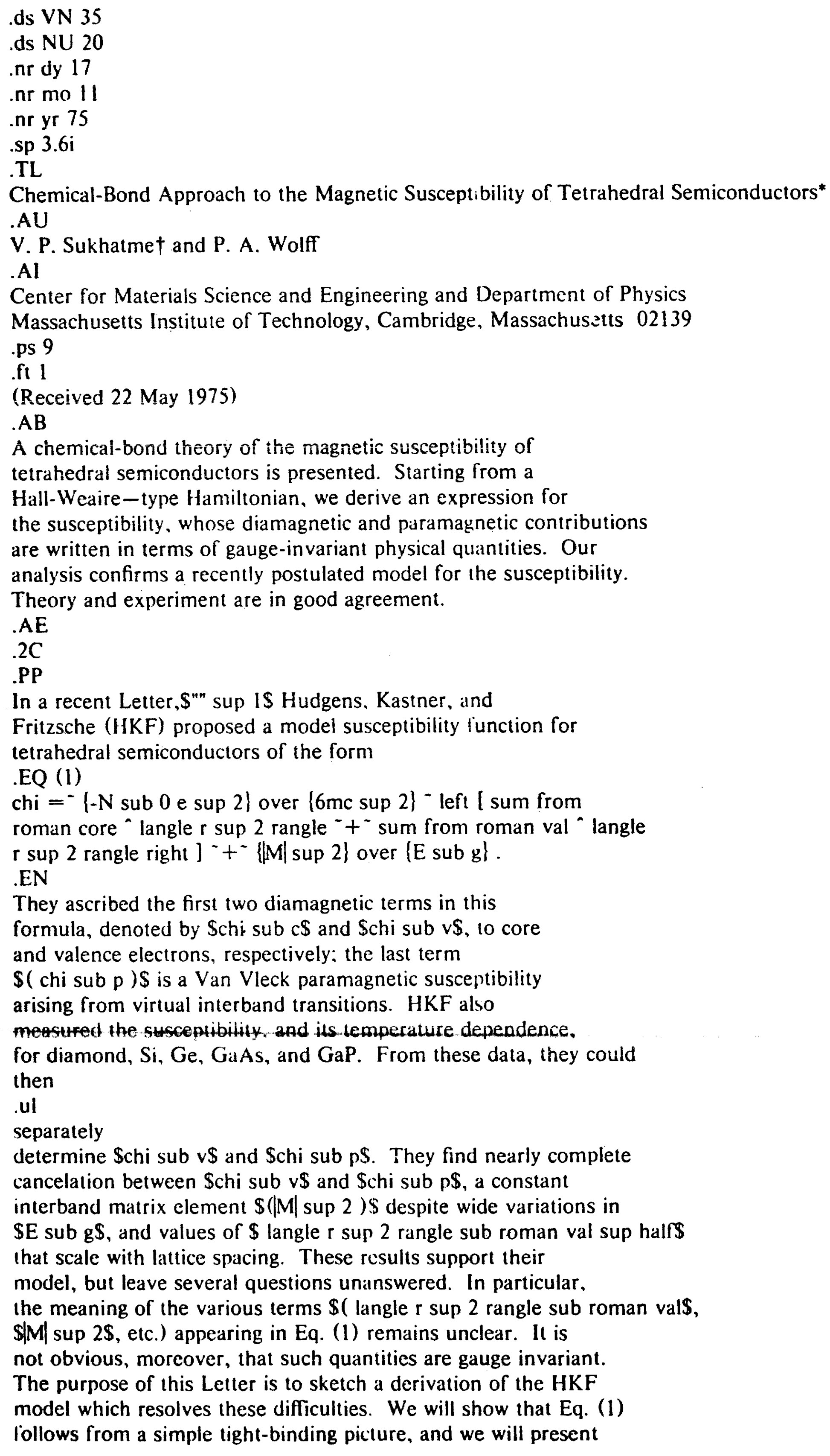
[1] S. Hudgens, M. Kastner, and H. Fritzsche, Phys. Rev. Lett. 33, 1552 (1974).

[2] J. C. Phillips, Rev. Mod. Phys. 42, 317 (1970).

[3] W. A. Harrison, Phys. Rev. B 8, 4487 (1973).

[4] L. M. Roth, J. Phys. Chem. Solids. 23, 433 (1962); J. E. Hebborn, J. M. Luttinger, E. H. Sondheimer, and P. J Stiles, J. Phys. Chem. Solids 25, 741 (1964); E. I. Blount, Phys. Rev. 126, 1636 (1962); R. M. White, Phys.

Figure 2



```
.ds VN 35
.ds NU 20
.nr dy 17
.nr mo 11
.nr yr 75
.sp 3.6i
.TL
Chemical-Bond Approach to the Magnetic Susceptibility of Tetrahedral Semiconductors*
.AU
V. P. Sukhatme† and P. A. Wolff
.AI
Center for Materials Science and Engineering and Department of Physics
Massachusetts Institute of Technology, Cambridge, Massachusetts 02139
.ps 9
.ft 1
(Received 22 May 1975)
.AB
A chemical-bond theory of the magnetic susceptibility of
tetrahedral semiconductors is presented. Starting from a
Hall-Weaire—type Hamiltonian, we derive an expression for
the susceptibility, whose diamagnetic and paramagnetic contributions
are written in terms of gauge-invariant physical quantities. Our
analysis confirms a recently postulated model for the susceptibility.
Theory and experiment are in good agreement.
.AE
.2C
.PP
In a recent Letter,$"" sup 1$ Hudgens, Kastner, and
Fritzsche (HKF) proposed a model susceptibility function for
tetrahedral semiconductors of the form
.EQ (1)
chi =~ {-N sub 0 e sup 2} over {6mc sup 2} ~ left [ sum from
roman core ^ langle r sup 2 rangle ~+~ sum from roman val ^ langle
r sup 2 rangle right ] ~+~ {|M| sup 2} over {E sub g} .
.EN
They ascribed the first two diamagnetic terms in this
formula, denoted by $chi sub c$ and $chi sub v$, to core
and valence electrons, respectively; the last term
$( chi sub p )$ is a Van Vleck paramagnetic susceptibility
arising from virtual interband transitions. HKF also
~~measured the susceptibility, and its temperature dependence,~~
for diamond, Si, Ge, GaAs, and GaP. From these data, they could
then
.ul
separately
determine $chi sub v$ and $chi sub p$. They find nearly complete
cancelation between $chi sub v$ and $chi sub p$, a constant
interband matrix element $(|M| sup 2 )$ despite wide variations in
$E sub g$, and values of $ langle r sup 2 rangle sub roman val sup half$
that scale with lattice spacing. These results support their
model, but leave several questions unanswered. In particular,
the meaning of the various terms $( langle r sup 2 rangle sub roman val$,
$|M| sup 2$, etc.) appearing in Eq. (1) remains unclear. It is
not obvious, moreover, that such quantities are gauge invariant.
The purpose of this Letter is to sketch a derivation of the HKF
model which resolves these difficulties. We will show that Eq. (1)
follows from a simple tight-binding picture, and we will present
```

Figure 3—Beginning of input for Figure 1

columns to be left-adjusted, right-adjusted, centered, or aligned by decimal points, as required. Headings as requested by the typist are placed over the columns. The typist need only specify the form of each column and type the data separated by tabs; other processing is automatic.

(3) The next procedure is *eqn*, which identifies equations and processes the input language into the typesetting (or line printer) language. Equations within tables are handled properly.

(4) The actual typesetting and layout by *troff* now follow. In the process of this stage the input material is handled three times.

(a) First, sections of double and single column text are collected. Material specified as too wide for double-column must of course be set full width, but the program will change narrow material to wide if setting as requested would cause an unattractive section of very short columns. Up to an entire page of text is gathered and processed at once. There is no backup: a limitation of our typesetting software prevents us from ever changing the initial decisions about line length, etc., used with a particular piece of text. Instead, the estimates of the lengths of each section of text produced by the first program in the sequence permit the layout routine to decide the proper format for each section. For example, there are two cases in which the program will change narrow text to wide: if it is too close to the bottom of a page, or if the next change to wide text follows very closely. As the text is gathered, figures and out-of-text tables are collected but temporarily set aside. The expected location of each figure is identified. By preference, figures are first placed at the top of the current page. If that space is no longer available, figures are placed at the bottom of the current page. If there is no space on this page, figures are placed at the top of the next page. The figures are not actually written out until the text for the page is processed, however, since (for example) it is not possible to place a half-width figure in full width text and on occasion the program is forced to widen a figure to full width in order to place it on the current page. It is possible, but rarely necessary, for the user to indicate where a particular figure is to be placed.

(b) Second, columns are formed by taking the double column pieces of text and dividing them into two columns. This may be difficult (for example, the expected breakpoint may be in the middle of a half-inch high equation) and thus the program tends to be conservative. It tries to put slightly less on each page than will actually fit; for *Physical Review Letters* the typography is sufficiently simple that one line of leeway is usually adequate, whereas for a very mathematical paper from *Physical Review D* three lines per page were left. The user can override this default for any paper and for any page, if it matters. While forming the columns the program counts the number of breaks between paragraphs and around equations in each column, and also measures the lengths of the columns.

(c) Third, the actual page is written out. The figures, and appropriate margin, and the columns of text are placed. To even up the columns, extra space is placed between paragraphs and equations. This is generally not noticeable, although it does mean that the lines in the two columns are not aligned. This conforms to APS/AIP practice. We do not have the capability of adjusting column lengths by changing the spacing between each line.

(5) The output medium is now chosen: the page may be written either on the typesetter, on a scope (for checking layout) or on a terminal for proofreading. In any case, the input typed by a typist is completely independent of the output device ultimately used.

If the pages produced are not acceptable, the user can make adjustments in the document in several ways. Material may be changed from narrow to wide format; in fact, the program will diagnose any equation which is too big for its intended appearance. Figures may be moved around in the text or enlarged, and more white space can be added to pages. For *Physical Review Letters* such methods are not usually necessary; they were only resorted to for trivial changes in the five papers of the experiment. Specifically, in two papers one line of white space was dropped from a page; and in one paper a figure was moved.

Finally, multiple papers may be run back to back and any individual paper may be placed at any vertical position on a page. These facilities are necessary to produce an entire issue of a journal, although irrelevant to the cost measurements in this experiment.

THE MAIN EXPERIMENT

Five copy-edited manuscripts (i.e., papers as given to typists at APS) were obtained from the editorial offices of *Physical Review Letters* through the courtesy of APS. Before looking at these manuscripts, eleven papers from the published journal were typed and typeset for practice and program debugging. Then the manuscripts were typed and typeset.

The experiment was done working entirely from manuscript; the published versions of the papers were not looked at until afterwards, and then only to supply one table omitted from one of the manuscripts and the date and initial page positions and volume numbers.

The sequence of operations performed by the typist for a paper is essentially this:

(a) original input of the paper (including a limited amount of "on-the-fly" correction of errors);
(b) rudimentary check of spelling, legality of equations and formatting commands (done by machine);
(c) fix any errors found;
(d) print a draft version on the typesetter;
(e) proofread draft;

(f) cycle through (c), (d) and (e) until the paper is in a satisfactory state.

Obviously some of these operations, particularly printing a draft, are normally overlapped with other activities, so the human time involved will not include (d) if there is enough typesetter capacity to keep typists busy.

The table below shows the data collected for the five papers, and the corresponding totals for each category. Each paper is characterized by its size in characters of raw input (including all mathematics, formatting information, etc.), the number and size of display equations like

$$S=\sum_{i=0}^{\infty} x_i$$

and the number and size of in-line mathematical expressions like π_i. The data on equations and in-line expressions give a rough measure of the complexity of the paper. Figure captions and tables are entered as normal input; their content is included in the size information. All times are in minutes.

Statistics on Five Sample Papers

	Browman	Lee	Keiser	Wolff	Tidman	Total
raw input (characters)	11236	15160	13366	15197	14004	68963
in-line expressions	154	130	118	113	134	649
(characters)	1256	1291	2913	1872	2083	9415
display equations	5	0	3	20	17	45
(characters)	318	0	385	3314	3058	7075
figures, tables	4, 1	3, 0	1, 1	0, 0	3, 0	11, 2
raw input time	56	63	66	77	84	346
subsequent editing time	14	21	22	20	21	98
total typing time	70	84	88	97	105	444
output pages	3	3.5	3	3	3	15.5
time per page	23	24	29	32	35	29

The average typing time per *final* page is thus 29 minutes. The statistics with regard to typing time do not take into account the unquantifiable fact that the typist, Ms. Carmela Scrocca, is extraordinarily fast and competent. For comparison, however, the total times recorded by APS for their typing and correction of the same papers are presented below.

Comparison of Typing Times

	Browman	Lee	Keiser	Wolff	Tidman	Total
APS	230	240	135	300	170	1075
UNIX	70	84	88	97	105	444
APS/UNIX	3.3	2.9	1.5	3.1	1.6	2.4

Although the total UNIX performance is 2.4 times faster, this does represent a spectrum of typists. The fairest comparison is probably with the papers by Keiser and Tidman, which were typed by the best typists at APS. If we assume that Ms. Scrocca and these typists are of comparable skill, then the UNIX system is 1.5 times as fast as typewriter composition. (A further experiment with a longer and more mathematical paper again showed a ratio of 2.4 to 1.)

There are two other significant costs for which we have some data—proofreading and page composition. According to APS, page composition takes approximately 15 minutes per paper on the average. Much of this effort can be eliminated by the automatic layout operations described above; it seems reasonable to believe that the computer system could reduce this to five minutes per paper of human time.

Proofreading times are of course much less objective. A careful proofreading by a technical reader took 94 minutes for the entire set of five papers, or about six minutes per page; this appears to be a fairly stable estimate, and is in reasonable agreement with values supplied by APS.

HARDWARE

The computer system used for these experiments is a PDP-11/45 running the UNIX operating system. The hardware for typesetting is a Graphic Systems C/A/T typesetter (not the current model) with four fonts and a range of point sizes selected by lens turret motion. The time required to set a page (8.5×11 inches) with this device ranges from about three minutes for pages with no point size changes to 15 minutes for the most complex material observed in *Physical Review Letters*. The articles from this experiment required on the average from eight to fifteen minutes per page depending on the amount of mathematics.

The actual computer time used for production of the papers is broken down as follows. First, initial input using the UNIX text editor proceeds at about 500 characters per CPU second; this accounts for about 140 seconds for the entire experiment. A similar amount would more than cover the subsequent editing and checking operations. Editing is certainly not the limiting cost.

CPU time required for typesetting is much larger, as shown in the next table.

	Browman	Lee	Keiser	Wolff	Tidman	Total
CPU time (seconds)	143	157	206	287	331	1124
pages	3	3.5	3	3	3	15.5

The cost per page then is about 74 seconds of CPU time. This is not the time per finished page. In this experiment, there were two complete drafts done before final copy, so in effect each page was done three times. It is our belief that with more careful proofreading of the first draft, this could probably be reduced to something much closer to two printings per page on the average.

The effort described above brought the papers into agreement with the APS copyediting instructions. Since we did not have access to an APS style guide a small amount of extra effort was expended to bring the manuscripts into exact conformance with APS rules by hunting around the library to find articles with similar features. The extra time involved in these changes is *not* included in the measurements.

COSTS

Although our typesetter averages about 12 minutes per page, it is an old model. The currently available Graphic Systems C/A/T (as timed at another Bell Labs computer installation) is twice as fast, using about six minutes per page. In a two shift day it could produce 160 pages of drafts. An extra 25 pages per day could be produced during an unattended third shift. At 2.5 to 3 tries per page, the finished page output rate would be 60–70 per day. Since a typist can comfortably sustain ten finished pages per day, six typists could keep one typesetter busy. The figure of five hours per day is deliberately conservative to allow for the Hawthorne effect, different typing skill levels, uneven workload, and non-typing time.

The cost of the basic computer hardware, exclusive of typesetting equipment, is about $150,000; this configuration could support (if there were no other demands on it) three or four typesetters at $15,000 each plus the twenty typists needed to keep the typesetters running. Realistically, two operators and a programmer would also be required. Amortizing the hardware (including three typesetters) over four years and 21 days per month makes a cost per day of $200 for equipment. The maintenance contract for the computer equipment costs about $60 per day. The Western Electric Company charges a one-time software license fee of $22,-000 for the UNIX software, including typesetting programs; amortized over four years, this amounts to about $20 per day. Personnel costs are of course highly variable. The following table shows estimated monthly salaries per person, multiplied by a factor of 1.5 to allow for overhead and divided on the basis of 21 working days per month.

Item	Cost/day
Computer hardware	$200
Computer maintenance	60
Computer software	20
20 typists ($850/mo)	1200
2 operators ($800/mo)	110
1 programmer ($2000/mo)	140
4 proofreaders ($1100/mo)	300
Supplies, etc.	50
Total per day	$2080

Since this arrangement produces about 200 finished pages per day, the cost per camera-ready page is about $10. This does not include copy-editing or the handling of figures. If overhead costs were taken as equal to basic salary, so that a factor of 2.0 replaced 1.5 in the calculations for the table, the cost per page would come out at $14 instead of $10.

Quoted costs of conventional operations vary widely. In all the figures below the date is given after the price; the UNIX costs, of course, are 1976 prices. APS itself quotes $40 per page for monotype (1970) and $29 per page for typewriter composition (1972) including illustrations.[5] Other quotes are $32 per 1000-word page for AIP (1973) and $28 per 1000-word page for ASCE (1973) for similar typewriter composition methods; some of the differences in costs are explained by the fact that AIP expects text to cost $25 per 1000 words while mathematics costs $65 per 1000 words, and the various journals differ in mathematical content.[6] The IEEE also reports a *difference* of $18 per page between mathematical and nonmathematical journals (1973).[7] A large survey reported "editorial" costs of $29 and $26 per page (1975) for two groups of 20 journals publishing about 40,000 pages per year, but exactly what is covered by this is unclear, especially as one group reported 20 percent of its costs as "remainder" while the other reported 5 percent.[8] A very low cost quote was given by SAE at $10.50 per page composed (1973) although this is not for a complete journal but rather for individual article publication. They itemize editing and proofreading separately at another $1.50 per page.[9]

Perhaps the most useful summary is to note that many sources agree that a typist can be expected to produce about 1000 pages per year or four pages per day with conventional typewriter composition.[10,11] Our typist, at half an hour per page, could easily do 2500 pages per year, which exceeds even the best typists working with conventional equipment.

The small scale of the UNIX system makes it very adaptable. An operation with fewer typists and pages than the one we have sketched, for example, would not have proportionally higher costs, since the majority of the costs are manual and not tied to the computer installation. A half-size (or ten-typist) shop would still operate at about $12 per page or so. Furthermore, the UNIX system is general-purpose, and a use might well be found for the surplus computer capacity. (The American Physical Society is currently installing a UNIX system which will be used for both typesetting and editorial management functions.) In this case even a two or three typist operation would be reasonable, as only the typesetter costs would have to be covered in full by the printing operation.

On the other hand, costs cannot be significantly reduced by enlarging the computer system. Most of the cost is in typists' salaries, and even a 50 percent reduction in the hardware costs would provide at most a 5 percent saving per page. In addition, a larger typesetter would involve substantial (perhaps a person-year) software costs to revise the formatter. Further development, instead, should emphasize additional aids to the input typists. In particular, we have no way on our hardware of handling drawings or figure contents.

CONCLUSIONS

Computerized typesetting of technical material is faster than typewriter composition, because it mechanizes those parts of the typing job which most slow down the typist. In particular, the effort to lay out complicated equations and tables is essentially eliminated, as is the need for inserting and removing keys for special characters.

If the experiment were to be continued, some of our operations could be improved. In particular, inadequate communication with APS before the experiment caused some confusion on our part about their copy-editing conventions and format rules; this in turn led to extra editing time. Additional training of our typist in the use of our text editor would have been desirable; the process of making changes was not as efficient as that of initial typing. Similarly, we are not equipped or staffed for large scale proofreading operations, and a production shop would undoubtedly have proofread more accurately with fewer delays than we did. Finally, the layout of figures still requires occasional manual intervention; these programs could be further improved. In particular, the program works well on text with relatively few figures or very wide equations. More than four figures per page, in fact, cannot be done with APS style rules obeyed, and the program is likely to produce unattractive results in this case.

The present system, however, has many advantages. In addition to the basic demonstration of feasibility and economic attractiveness, there are obvious side benefits of computerized composition. The text is available in machine-readable form for secondary services like indexing, information banks, or later publication via computer-generated microfilm. Additional uses of a machine-readable file will certainly appear.

The UNIX system also contains several examples of computer aids not found in ordinary printing operations. Two different spelling error checking programs are available; one operates by letter patterns and one by reference to a dictionary. The syntax of the commands to lay out equations and pages can be checked automatically. To determine the number of column inches required for each article we had a special layout program which went through the steps required for setting galleys, although no printing was actually done. Many other programs for text handling are also available: for example, it is easy to scan multiple files for occurrences of particular words.

The quality of the output is higher than with typewriter composition. The copy is camera-ready, except for figures. Pages rather than galleys are produced, including complete header and footer lines. The output is attractive, with more fonts than are economic with a typewriter. Right-justified margins are available if desired. Recently many scientific journals have lowered their typographical standards to reduce costs; computers may make this unnecessary.

Copy-editing costs should also be reduced by a computer-based system. Changes are easier to make; presently copy-editors tend to mark repetitive changes at each point in a manuscript whereas computer editors can easily change all instances at once. It is also easy to number footnotes and equations automatically if desired. Finally, as computer editing systems spread, and more and more authors are able to provide the original manuscript in machine-readable form if desired, copy-editing can be done entirely by the use of a computer editor.

We conclude, finally, that a UNIX-based system would be an appropriate way now for a small scientific journal, using a reasonably simple typographic style, to compose its papers. It would also leave a publishing company in an excellent position to take advantage of future improvements in computing systems.

ACKNOWLEDGMENTS

We must thank Ms. Scrocca, whose typing is the basis of this research; J. F. Ossanna, who wrote the typesetting program *troff* and modified it at our request; and the editorial staff of the American Physical Society, which supplied the material. Without them none of this work could have been accomplished.

REFERENCES

1. Thompson, K. and D. M. Ritchie, "The UNIX Time-Sharing System," *Comm. ACM* Vol. 17, pp. 365–375, 1974.
2. Ossanna, J. F. *Troff User's Manual,* Bell Laboratories internal memorandum.
3. Kernighan, B. W. and L. L. Cherry, "A System for Typesetting Mathematics," *Comm. ACM,* Vol. 18, pp. 151–157, 1975.
4. Lesk, M. E. *Tbl—A Program for Formatting Tables,* Bell Laboratories internal memorandum, 1976.
5. Marks, R. H. and A. W. K. Metzner, "Typewriter Composition Cuts Journal Costs, Speeds Publication," *IEEE Transactions on Professional Communication,* **PC-16,** pp. 73–79, 174, 1973.
6. Herschmann, A. and P. Parisi in a discussion session, *IEEE Transactions on Professional Communication,* **PC-16,** p. 165, 1973.
7. Gannett, W., *ibid.*
8. Sanders, J. W., C. M. B. Anderson, and C. D. Hecht, *Scientific Publication Systems: An analysis of past, present and future methods of scientific communication,* Toronto University report to the National Science Foundation, NTIS number PB 242 259, June 1975.
9. Staiger, D. L. "Separate Article Distribution as an Alternate to Journal Publication," *IEEE Transactions on Professional Communication,* **PC-16,** pp. 107–108, 177, 1973.
10. Marks and Metzner, *loc. cit.* Also confirmed by conversations with the American Physical Society.
11. Staiger, D. L. *In-House Photo Composition of Technical Manuscripts for Mid-Range (2000–5000) Society Publications,* presented at the CESSE Annual Meeting, Washington, D. C., 1975.

The computer in manufacturing—Reduction of scrap by computer monitoring

by P. E. GOBER

Westinghouse Semiconductor Division
Youngwood, Pennsylvania

ABSTRACT

A computer furnace monitoring system was implemented as the first stage of a computer process monitoring system designed to provide better control of the process used to manufacture high power semiconductor devices. The purpose of the furnace monitoring system is to reduce scrap resulting from furnace malfunctions that are otherwise not detected in time to salvage the product. The system also improves reproducibility by maintaining a tight control of the elevated furnace temperatures (±2°C at 1250°C). Additional results of the system are greatly improved operation visibility during the run, increased furnace utilization as a result of computer assisted scheduling, and improved correlation of results among different furnaces.

The Ⓦ 2500 computer provides the process I/O necessary to monitor furnace temperatures as measured by thermocouples, sound alarms when deviations from the spec occur, store information for later analysis, plot furnace behavior, and assist in scheduling by calculating cycle times. It also provides the furnace operators with the ability to quickly and accurately determine furnace temperature at any time during the cycle. The real time foreground/background operating system of computer, based on a strict priority system, allows data analysis programs to run without disturbing the real time monitoring of physical parameters such as temperature.

The furnace operator's interface with the computer, a set of related programs accessed from a teletype by a single command, is also described. The system is user-oriented, and employs a conversational question and answer format that guides the operator through the various procedures. The system also incorporates error detection and correction methods to prevent mistakes from improperly entered data.

The computer is an effective tool for the reduction of cost resulting from scrap. It also provides the basis for an integrated monitoring system encompassing the entire manufacturing process.

INTRODUCTION

The computer has long been used by industry to handle payrolls, accounting, and various records, and by scientists for mathematical analysis, modeling, and simulation. A less explored area, however, is that of the computer as a manufacturing tool: the computer can play an important role in day-to-day operations in a manufacturing environment, where features such as memory technology, innovative architecture, and language syntax must pale in significance when compared to the manufacturing user's concern with the computer's effect on his product quality, amount and cost of defective material produced, and consequences of missing a production schedule.

The Westinghouse Semiconductor Division in Youngwood, Pennsylvania, is such a manufacturing facility. The production of high power semiconductor devices involves processing raw silicon in rod form through several diffusion, alloy, metallization, and passivation operations before the fabrication is complete, and the device is tested, packaged, and ready for use in its final form. The most advanced technology, however, cannot make a good product unless the many parameters which define the process recipe are accurately measured and controlled. Parameters such as furnace temperatures, gas flows, belt speeds, and humidity are critical to the manufacturing process, and even the most sophisticated gauges are useless unless someone can continuously observe them.

After investigating several alternatives, we decided to purchase a minicomputer to help reduce D.A. (defective apparatus) by monitoring some of these vital parameters. The monitoring of diffusion furnace temperatures was selected as the initial application because of relative ease of implementation (reading voltages generated by thermocouples) and the large potential return on investment. The furnace monitoring system would be the first step in establishing a real-time and historical data base of product characteristics and performance. Other important parameters, such as gas flows, could be added to the system, and the multi-tasking capability of the computer chosen would permit the execution of data analysis and design programs in a background model while real-time monitoring continued in the foreground.

The computer selected was a Westinghouse Ⓦ 2500. Important features include well-developed process-oriented hardware that can be fully controlled under a high level language (FORTRAN IV), and a multitasking operating system which provides the foreground/background capability mentioned above. The computer was installed in October, 1974, and has been expanded to its present configuration of 64K core, 3.75 million (16-bit) words disk storage, incremental plotter, line printer, card reader, CRT computer console, TTY furnace system console, and process hardware consisting of a 40-point-per-second analog input system, external interrupt system, contact closure input system, and contact closure output system. Approximately 16K of core is reserved for the furnace monitoring software, and the computer operating system occupies another 12K. The rest of core is available for other real time and batch programs. The computer is capable of running 1,024 tasks on a priority basis.

The name chosen for the computer, "DARIN," stands for "Defective Apparatus Reduction and Information," and reflects the computer's purpose: D.A. reduction and improved product by providing information that was otherwise either unavailable or difficult to obtain.

THE DIFFUSION PROCESS

The first step of the manufacturing process is the diffusion of dopants into the sliced silicon to impart the desired electrical characteristics. From three to six separate diffusion operations are required for each device, and errors in this step of the process are usually irreversible.

Diffusion operations are carried out in large furnaces at elevated temperatures (1100 to 1250°C). The temperature must be maintained within a tolerance of ±2.5°C for extended periods (two to 40 hours). This soak cycle is followed by a six hour slow cool to quench the diffusion. Each furnace is equipped with a timer set to maintain the peak temperature for the prescribed time and then switch over to a programmed slow cool. If the timer or furnace controller malfunctions, the junction may be driven too deep. A run that is damaged in this manner cannot be salvaged. Since a run contains from 200 to 1000 slices, such furnace malfunctions are very costly, not only in terms of scrap generated, but also in production time lost when a replacement run must be started from the beginning of the four to six week process.

A closely related problem is that of scheduling the furnaces. It is very difficult for the foreman to keep track of the conditions of 51 furnaces, used for 11 processes, each with different behavior characteristics, and very few set for the same time cycles.

COMPUTER IMPLEMENTATION

Computer monitoring was instituted for half the furnaces in June, 1975. It has since been expanded to include all the furnaces. The system was designed not only to flag furnace malfunctions, but also to be a useful tool for the furnace operators.

As an example, one of the simplest features of the system, that of temperature reading, has proven extremely valuable. Before the computer was installed, furnace temperatures were checked by a slow and often inaccurate procedure: a thermocouple, attached to a chart recorder, was inserted into the center zone of the furnace, and allowed to stabilize for approximately 15 minutes, before the temperature was read as a voltage, which was then converted to degrees by a table. This method could only be used with empty furnaces; there was no way to read the furnace temperature while a run was loaded in the furnace.

In addition, the chart recorders, although calibrated weekly, were prone to drift, and could drift as much as 15°C without detection. Such errors were unknowingly passed on to the furnaces profiled with those recorders, and were an additional source of D.A.

SYSTEM DESCRIPTION

The furnace monitoring system is centered around a set of tables occupying approximately one-fifth of the core reserved for the system. The tables describe the real time state of each furnace, as well as define temperature specifications and furnace characteristics. Each time a furnace is loaded, the tables are updated to reflect cycle information such as the times the run should enter cool down, be unloaded, and a projection of the time the furnace will be reheated and ready for a new run.

The system is controlled by a master scheduling routine, MAST, which references the computer's internal 60 Hz clock and issues calls to tasks performing the following functions:

1. Calibrate analog-to-digital conversion system every 15 seconds.
2. Read and store furnace temperatures every 15 seconds.
3. Compare temperatures to specs every 60 seconds.
4. Record out-of-spec data on disk every 60 seconds.
5. Sound alarms as they occur.

TEMPERATURE MEASUREMENT

Furnace temperatures are measured with a type S (Platinum/Platinum-10% Rhodium) thermocouple located on the outside of the liner of each furnace. This configuration provides accurate detection of temperature behavior (±0.5°C), minimizes thermocouple exposure to corrosive elements in the furnace, and does not interfere with furnace loading or unloading.

The thermocouples are connected to the computer by screw terminals in a thermally insulated compartment used as a cold junction box (CJB). A resistance temperature detector (RTD) mounted within the CJB measures room temperature. This temperature is converted to millivolts and is added to the millivolt measurement of the type S thermocouple before the thermocouple measurement is converted to degrees centigrade.

The relationship between temperature and voltage for a type S thermocouple is linear in the ranges 0°-30°C and 1000°C to 1300°C. The diffusion furnaces have soak temperatures between 1135°C and 1250°C. Therefore, an equation of the form Y=mX+b can be used to convert room temperature, as measured by the RTD, to millivolts on a type S thermocouple scale. A second equation of the same form is used to convert millivolts, as measured by the thermocouple with reference to the 0° established by the RTD, to degrees centigrade. The equations are detailed below.

$$RTdC=(RTDmV/0.7114)+25 \tag{1}$$

$$RTmV=(RTdC*0.0061)-0.01 \tag{2}$$

$$FTdC=(FTmV+RTmV)*83.25+205.2 \tag{3}$$

RTdC =Room temperature in °C
RTDmV =(milli-) Voltage detected by RTD
RTmV =Room temperature in millivolts (for type S thermocouple)
FTdC =Furnace temperature in °C
FTmV =Furnace temperature measured in millivolts (type S thermocouple)

The constants in equation (1) are specific to the Model S4 RTD used. The constants in equations (2) and (3) are derived from a least squares fit of temperature versus voltage, using tables from the National Bureau of Standards (1971).

Although temperature readings are constantly updated, they are only compared to spec temperatures while the furnace is loaded. A set of flags informs the comparison program, CHEK, of the status of each furnace and, therefore, of the action to be taken. The possible furnace states are:

1. Empty
2. Shut down (for maintenance or cleaning)
3. Loaded, in soak cycle, and in spec
4. Loaded, in cool down cycle, and in spec
5. Ready to be unloaded
6. Loaded and out of spec

CHEK examines the status flag of each furnace and then performs the appropriate check. No check is made for conditions 1 and 2. Furnaces in soak (3) are checked for temperature within ±2.5°C of spec, for time to turn off gas flows, and for the beginning of cool down. Furnaces in cool (4) are monitored to maintain a cooling rate of at least one degree C per minute until 800° is reached.

When a furnace goes out of spec (6), a timer is started. If it returns to spec within five minutes, normal monitoring continues; if it remains out of spec for five minutes, an alarm is sounded and a message is printed on the TTY, notifying the furnace operators of the furnace, its temperature, and the spec. If corrective action does not bring the furnace back in spec within ten minutes, the alarm is rung again. The alarm is also sounded if a furnace does not enter the cool down cycle within ten minutes of the prescribed time.

While a furnace is out of spec, the data is logged on a disk file once a minute. This provides a record of furnace behavior that can be used to determine corrective action, interpret results, or study furnace characteristics. Any furnace, regardless of condition, can be flagged to log data in this manner, providing a means of studying reheat cycles and recovery times. This data is summarized once a day in a table showing furnace number, the time it went out of spec, how long it remained out, the spec temperature, and the minimum, maximum, and average temperature during that period. If more detail is needed, the data can be printed as a simple chronological list or displayed on the x-y plotter as a graph of temperature versus time.

FURNACE OPERATOR'S INTERFACE

The furnace operators interact with the computer through a conversational task, CON1, which runs on the TTY at the operator's loading station. The operator specifies the desired action, such as loading a run, and the task calls in the appropriate program or subroutine. All interaction is in the form of questions from the computer and answers from the operator. Error detection and correction methods are included in the programs. The programs were designed to be user-oriented, and to make the computer a useful tool for the furnace operators, requiring a minimum of operator response.

The operators use the computer to load and unload runs, to read furnace temperatures, and to check furnace availability. The alarms and printed messages alert them to problems and help them determine the necessary corrective action. Sample interactions are shown in Figures 1 and 2.

```
     TYPE OPTION NO.
  03
PROCESS?
  PHOS
FURNACE?
  04
STANDARD TEMP=0   SPECIAL=1
  0
PHOS—DEP TIME?
  2.0
DRIVE TIME?
  5.9

     VERIFY DATA

PROCESS = PHOS       FURNACE = 4       SOAK TIME = 7   53
STARTED AT 22     8     41
COOLDOWN AT          (DATE—HR—MIN) 22.—16.—35.
RUN ENDS             (DATE—HR—MIN) 22—22—35
FCE AVAILABLE        (DATE—HR—MIN) 22—23—35

PHOS OFF AT (DATE—HR—MIN) 22—10—41

IF DATA IS CORRECT, TYPE 0 TO START RUN.
IF NOT, TYPE 1 TO RE-ENTER DATA.
  0
BYE
```

Figure 1

```
TIME TO TURN OFF WATER IN FURNACE 34  CLOCK = 8:44

RUN IN FURNACE 34 IS DONE—CLOCK = 9:05

    TYPE OPTION NO.
 04
FURNACE?
 34

    RUN ENDED IN FURNACE 34 AT 9:5
FURNACE?
 00
BYE

    TYPE OPTION NO.
 06
FURNACE?
 26
FURNACE 26 READS     1249.5

FURNACE?
 27
FURNACE 27 READS     1251.0

FURNACE?
 24
FURNACE 24 READS     1135.5

FURNACE?
 34
FURNACE 34 READS     1150.1

FURNACE?
 47
FURNACE 47 READS     1100.0

FURNACE?
 48
FURNACE 48 READS     1104.6

FURNACE?
 00
BYE
```

Figure 2

RESULTS

The results of the furnace monitoring system have exceeded the initial goal of reducing D.A. by catching furnaces that fail to go into cool. It was projected that, to be cost-effective, the computer should catch at least one furnace malfunction of this type a month. The actual savings have been 3 to 4 furnace loads a month, a considerable amount of product. The computer also provides instantaneous and accurate temperature readings, which not only saves time over the previous manual method, but also enables the operators to determine temperatures at any stage in the furnace cycle instead of only while the furnace is empty. The operators and foreman are able to obtain up-to-date information on furnace status, including projections of next available furnace for a given process. Sample displays of this information are shown in Figures 3 and 4.

```
    TYPE OPTION NO.
 05
BY FURNACES = 0    BY OPERATIONS = 1
 1
PROCESS?
 PHOS

   PROC. = PHOS                    SPEC = 1205.00
FURNACE NO.   NOW                 LATER      OUT OF SPEC.

  5             ×
  6                              23- 5:50
  7                              22-21:54
  8                              22-21:52
  9                              22-22:42
 10                                              ×
  4                              22-23:35
 52                              22-23:35
 25                              22-16:42
 28                              22-14:28
 31                              19- 8:36
 33                                              ×
 41                              22-15:10
 50                              22-12:35
 35                               0- 0: 0
BYE
```

Figure 3

The computer's ability to track furnace behavior has been used to study the reheat time of furnaces, which resulted in increasing throughput by 3-5 hours in some cases, and also provides data used to determine which furnaces should be replaced.

Potential benefits are even greater. The computer provides more extensive and current information on the furnace system than was available before. We are currently expanding the system to include formation of a data base that will be used to correlate furnace behavior, device characteristics determined by the diffusion operations, and performance at final test. Data is also being collected on the frequency and type of maintenance required by each furnace, with the goal of scheduling regular preventive maintenance. Work-in-process inventory in the diffusion area is also being tracked by the computer, and provides the foreman with more current information than was obtainable with the previous cumbersome handcount method.

CONCLUSIONS

The small computer has become a valuable production tool in the manufacturing environment of Youngwood. The improved process control, combined with the computer's versatility in the areas of data gathering and analysis, offer almost unlimited potential. In order to be effective, however, such a system must be designed to fit the manufacturing process. It must be specific enough to meet the peculiar needs of each application, yet the programs must be structured to allow for changes in the environment; in our system, a furnace may be converted from one process to another, experimental runs may require a non-standard temperature, etc.

FURN	FLAG	LODTYP	TEMP	SPEC	COOL	HOUR	MIN	DONE	HOUR	MIN	ALRM
1	1	DRIVE	1150.32	1150.00	23	18	0	24	0	0	F
2	1	STMDRI	1150.05	1150.00	23	16	15	23	22	15	F
3	1	ALGABN	1250.59	1250.00	24	1	50	24	7	50	F
4	4	PHOS	944.29	1121.00	23	13	23	23	19	23	F
5	4	PHOS	693.01	954.00	23	10	3	23	16	3	F
6	1	PHOS	1203.51	1205.00	23	15	9	23	21	9	F
7	−1	PHOS	777.09	917.00	23	7	59	23	13	59	F
8	0		1205.54	0.00	0	0	0	0	0	0	F
9	4	PHOS	1093.82	1162.00	23	14	11	23	20	11	F
10	0		924.51	0.00	0	0	0	0	0	0	F
11	7		682.64	0.00	0	0	0	0	0	0	F
12	7		−192.04	0.00	0	0	0	0	0	0	F
13	7		221.04	0.00	0	0	0	0	0	0	F
14	7		824.68	0.00	0	0	0	0	0	0	F
15	7		800.63	0.00	0	0	0	0	0	0	F
16	7		−541.31	0.00	0	0	0	0	0	0	F
17	0	ALSEAL	1249.14	1250.00	22	14	0	22	20	0	F
18	0		1249.26	0.00	0	0	0	0	0	0	F
19	0		1249.90	0.00	0	0	0	0	0	0	F
20	0		1250.04	0.00	0	0	0	0	0	0	F
21	0		1253.91	0.00	0	0	0	0	0	0	F
22	0		1002.97	0.00	0	0	0	0	0	0	F
23	7		329.20	0.00	0	0	0	0	0	0	F
24	0		1136.97	0.00	0	0	0	0	0	0	F
25	1	PHOS	1205.05	1205.00	23	18	35	24	0	35	F
26	4	DRIVE	1193.42	1227.00	23	14	34	23	20	34	F
27	4	DRIVE	1203.64	1228.00	23	14	35	23	20	35	F
28	0		830.34	0.00	0	0	0	0	0	0	F
29	1	DRIVE	1250.05	1250.00	23	17	44	23	23	44	F
30	1	DRIVE	1249.72	1250.00	23	17	46	23	23	46	F
31	1	PHOS	1204.59	1205.00	23	21	35	24	3	35	F
32	0		1249.91	0.00	0	0	0	0	0	0	F
33	0		1253.61	0.00	0	0	0	0	0	0	F
34	1	OXIDAT	1149.30	1150.00	23	19	15	23	19	15	F
35	7		1126.65	0.00	0	0	0	0	0	0	F
36	0		950.94	0.00	0	0	0	0	0	0	F
37	7		1218.97	0.00	0	0	0	0	0	0	F
38	4	PDPDRI	1245.37	1251.00	23	15	8	23	21	8	F
39	1	PDPDRI	1250.18	1252.00	23	15	23	23	21	23	F
40	7		226.13	0.00	0	0	0	0	0	0	F
41	1	PHOS	1204.72	1205.00	23	19	23	24	1	23	F
42	1	PDPDRI	1253.84	1252.00	24	6	19	24	12	19	F
43	1	OXIDAT	1150.29	1150.00	23	19	30	23	19	30	F
44	1	OXIDAT	1149.86	1150.00	23	19	31	23	19	31	F
45	7		255.03	0.00	0	0	0	0	0	0	F
46	0		1251.06	0.00	0	0	0	0	0	0	F
47	0		1076.47	0.00	0	0	0	0	0	0	F
48	0		1110.82	0.00	0	0	0	0	0	0	F
49	1	DRIVE	1250.41	1250.00	23	15	5	23	21	5	F
50	0		933.63	0.00	0	0	0	0	0	0	F
51	0		1250.06	0.00	0	0	0	0	0	0	F
52	4	PHOS	811.90	1063.00	23	12	14	23	18	14	F
53	0		1251.06	0.00	0	0	0	0	0	0	F
54	7		839.22	0.00	0	0	0	0	0	0	F
55	7		825.80	0.00	0	0	0	0	0	0	F

Figure 4

Most important, the system must be oriented toward the user—in this case, hourly employees, supervisors, and engineers who are not computer operators or programmers. The conversational question-and-answer method has been very successful at Youngwood, allowing the user to "converse" with the computer in familiar terms. The user's feedback should also be used to improve the system. For example, initially the computer only alerted operators to problems, such as a temperature out of spec. After becoming familiar with the computer and with entering data on the TTY, the operators asked if the computer would be able to somehow print a message when it was time for them to turn off gas flows. Such a modification of the original system was well worth the program changes because it contributed not only to the objective of reducing D.A. in an area that had not been originally considered, but also made the computer a more useful tool, not just a ''glorified thermometer'' or an expensive new gadget that is more trouble than it is worth. To be truly effective, as a manufacturing tool, the computer system must be carefully designed, and simplicity of interface with the end user, regardless of his background, must not be dismissed as an unnecessary frill.

The furnace monitoring system was a test case for the computer at the Semiconductor Division. It has been successful, and the experience gained in solving the problems of implementation is being applied to the expansion of the system to include monitoring of other processes. The computer is becoming even more important as a manufacturing tool in areas not originally considered, such as work-in-process inventory and device design, and we are looking forward to continued expansion into other areas that affect the daily dollars and cents concerns of the manufacturing plant.

A methodology for multi-criteria information system design*

by JOHN S. CHANDLER and THOMAS G. DELUTIS
The Ohio State University
Columbus, Ohio

ABSTRACT

The design dilemma faced by the designer is to satisfy a set of conflicting user demands and resolve a set of conflicting resource requirements concurrently. In light of the complexity of modern systems, it is assumed that good system design need only produce satisfactory performance for both criteria. Current evaluative techniques, however, concentrate on either the user criterion or the system criterion aspect of the total design problem, but not both. A methodology has been developed that establishes a formal liaison between the evaluation of user goals as a function of system activity and the evaluation of resource utilization as a function of user demand, thereby creating a design/evaluation process that encompasses both criteria. The methodology employs three stages in an iterative manner to produce a "satisfactory" design. The IPSS simulator models and measures system activity, multiple goal programming evaluates both user and system goals, and heuristic procedures determine design modifications to improve performance. A functional description of the methodology and an example of its use will be presented.

INTRODUCTION

In March of 1973, ACM and NBS sponsored a Workshop** on computer performance evaluation. One of the major results of that Workshop was a consensus that there have been two separate approaches to the evaluation of information systems performance—one which focuses on the computer system domain and the other whose attention is directed at the application system (user) domain. Each have their own goals and measures: the computer system domain measures are based on resource queueing and utilization statistics and the user domain is evaluated through the performance of requested services. Measures such as throughput and response time are common for the latter. The Workshop also concluded that any performance analyses "should recognize both the costs of a computer installation and the needs of users for service."

The complexity of the design problem for modern computer based information systems has increased significantly over its predecessors due to:

a. the servicing of an expanding range of user or uses with corresponding diverse performance goals and resource requirements, and
b. the dynamic and unpredictable behavior of the system as a function of design decisions and load mix.

Thus, it is quite possible to improve the performance of the system with respect to one or more users at the expense of others. Likewise, because system resources are used by different users, improving the performance characteristics of one or more resources for the benefit of specific users may have an overall detrimental effect on performance. The problem presented to the designer is to configure a system which satisfies the user criterion while achieving system resource related performance criteria.

A computer based information processing system can be viewed as a symbiotic relationship between the system's users and its hardware, software and data resources. Ideally, the system will perform "optimally" when it achieves its user oriented objectives within a minimum cost system. However, optimal solutions are seldom achieved when systems are complex, ill defined or constrained for reasons outside the control of the designer, and thus, the designer usually settles for a satisfactorily behaving system. Hopefully, systematic procedures are employed to achieve system configurations which concurrently meet the user objectives while obtaining efficient utilization of its resources. Current evaluative technologies focus on only one criterion in the system design equation, either the user or the system's resource performance. The ability to simultaneously ascertain the impact of resource performance on user goals or vice versa is not readily achievable through these methodologies. The purpose of this paper is to describe a methodology which establishes a formal liaison between the evaluation of user goals as a function of system behavior

* This research is being conducted with the support of The National Science Foundation, Grant No. SIS75-21648.

** This was one in a series of Workshops sponsored jointly by ACM and NBS to examine the major issues involving computers. Performance evaluation was chosen as the topic of this Workshop because of its significant impact on computer usage. A summary of the conclusions appears in Reference 1.

and the analysis of resource performance as a function of user activity.

User oriented analyses with objective functions based on response time, throughput, and cost have been (and are continuing to be) reported in the literature. Most frequently, analytic approaches use queueing models as their basis (References 2-4 are representative of this type of analysis). Due to the necessity to maintain tractable models, many simplifications are required for a model's analytical solution. Simulatiqn models have also been applied to user oriented analysis.[5,6] Unfortunately, these models yield only average and/or aggregate measures of system response. As a result of these simplifications, the analyses produced by both of the approaches fail in many cases to identify the relationship between users and resources. Therefore, they are suspect when used to predict the impact on system performances of modifying the current environment.

Alternatively, performance analyses can be made from the system's standpoint, treating the user and his goals in the aggregate. The most common approach is a subsystem study, where a particular part of the information system complex is isolated, with the subsystem user(s) represented by a stochastic generator, both analytic and simulative. The most emphasized areas of research has been the I/O subsystem[7-10] and CPU utilization.[11-13] The problem with this level of evaluation is that, although providing valuable local intuitive insight, these models rarely relate to the ultimate information system user, and, therefore, do not provide realistic insight into global performance.

Examinations of complete systems have also been made. Exhaustive hardware/software measurements have been analyzed by Gonzales and Cantrell[14,15] while simulation models, including an aggregate user component, have been built by Reeves and Pooch, Norland, and Lum.[16-18] Although results of the evaluations include resource utilization statistics and user oriented measures such as response time, there is little attempt in these models to relate particular resource usage to the effort on user goal attainment. (Two exceptions are Lindsay's study of the KRONOS system[19] and Hall's data base investigations.[20]) But from practical experience it is evident that there is indeed a relationship between user goals and resource usage. In fact, Buzen[21] has recently proposed some fundamental laws for computer performance which relate resource activity to global system/user measures such as response time and throughput.

It is assumed that the objective of good system design is to satisfy both performance related criteria. However, in light of the complexity of modern systems, many design decisions tend to be made without proper supportive evidence on performance. The crux of the problem is to establish a causal relationship between user goal attainment and system resource expenditures. The methodology to be discussed has been designed to establish such a liaison and will be shown to allow for the collection of heretofore hard to obtain evaluative information. The methodology measures the impact of individual user classes on internal system performance and identifies system bottlenecks which inhibit the attainment of user goals. This is achieved by maintaining resource utilization statistics on a user class basis. This methodology presents an evaluative framework which is capable of eliminating many of the numerous nonsatisfactory designs by directing the designer to the most advantageous ones. This methodology is an iterative one with each iteration involving three separate but integrated stages. Figure 1 illustrates the activities for an iteration. Briefly the responsibilities for each stage shown in this figure are:

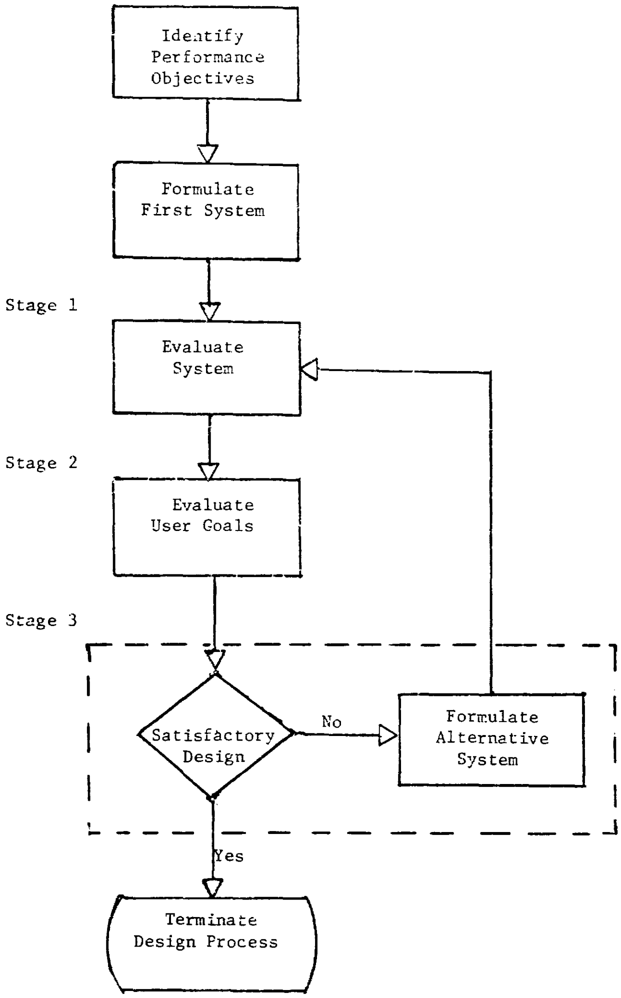


Figure 1—Stages in system design process

Stage 1: System Evaluation

This stage is responsible for evaluating the behavior of a specific information system model. It does this by associating the hardware, software and data activities belonging to a specific design with the system's user activities. The outputs of Stage 1 are performance statistics for the resources in the aggregate and for their behavior with respect to identified users (or uses). To perform this function, the IPSS Simulator is employed.†

† IPSS is a special purpose discrete event simulator whose development was conducted with the support of the National Science Foundation, Grant No. GN-36622.

Stage 2: User Goal Evaluation

Stage 2 has two purposes. The first is to ascertain whether the user goals are being either over or under achieved. The second purpose is to determine the "best" set of guidelines for altering the *current* system configuration in order to obtain the user goals with minimum penalty for either under or over achievement. Multiple goal programming is used for this purpose. As will be seen, "best" is a function of the assigned penalty coefficients in the goal programming objective function.

Stage 3: Design Evaluation

Stage 3 has two functions. The first is to ascertain whether or not the current design's performance is satisfactory with respect to both the user criteria and the system criteria. If the design is not satisfactory, then this stage's second goal is to define a new system based upon the current design, prior alterations, and the results of the Stage 1 and Stage 2 analyses. Heuristic procedures are currently employed for Stage 3.

The focus of this paper is on the Stage 2 formulation and its formal liaison to Stage 1. The paper also identifies the unique features of IPSS which permit this multi-stage multi-criteria methodology to be achieved. The paper concludes with a discussion of the use of the Stage 1 and Stage 2 results in the Stage 3 heuristics.

EVALUATIVE REQUIREMENTS

For the purposes of this methodology, an information system is viewed as the sum of its users and their goals, and the system's services and their subordinate activities. This is illustrated in Figure 2. It is assumed that the system's analyst can identify and classify the system's users according to their service request characteristics and according to the performance constraints imposed upon the system (i.e., goals) when honoring their requests. It is also assumed that the analyst can identify those information system activities which are critical to system performance. Obviously, the complexity of the problem is increased substantially when a system supports diverse users or provides a wide spectrum of services. Whether or not the system is complex or simple, the criteria for identifying system activities should be based upon the sensitivity of the system's performance with regard to changes in their behavior.

Information system services are viewed as being a series of distinct yet interconnected activities which are invoked during the processing of a stream of user requests for the

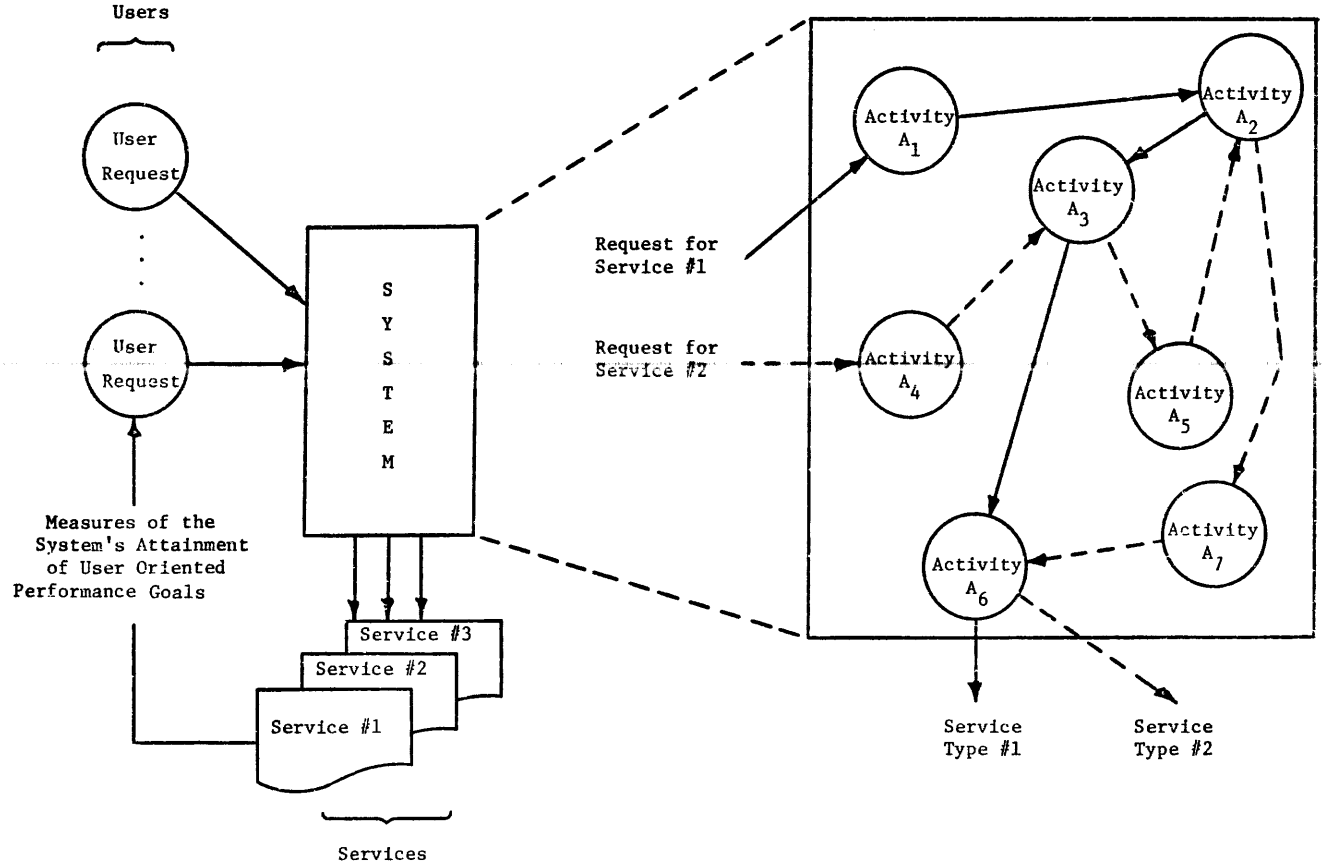


Figure 2—The methodology's view of an information system

service. Again, Figure 2 illustrates this view of a system. Most likely, system activities are aggregations of one or more traditional computer system functions that perform the following tasks:

1. request (job) scheduling,
2. task management,
3. resource allocation,
4. secondary storage I/O processing, and
5. application processing.

The choice of what constitutes an activity is part of the art of performance evaluation, however, a necessary condition for their selection is that they be measurable and that these measurements distinguish the service rates for separate classes of system services. It is also assumed that the role of performance measurement is to determine the current processing rate for the jth activity with respect to the ith service.

Figure 3 is a schematic of the functional composition of system activities. Each is viewed as an individual queueing system containing one or more priority queues and one or more identical servers. Additionally, the performance measure for the activity in processing a request type is the sum of both the queue performance and service functions of the activity. Throughout this paper, the variable $R_j(i)$ is employed to identify this performance of activity A_j, with respect to Service S_i. It is assumed to be the average of performance for all the executions of A_j for S_i. Also associated with each activity A_j is the performance factor β_j which is interpreted as the scaling factor to be applied to the $R_j(i)$ to obtain the level of performance for the jth activity which minimizes the goal programming objective function. It should be noted that the problem of identifying a "good" level of performance for activities, i.e., determining the appropriate values of the $R_j(i)$'s is compounded by the multiple use of the activity by different and possibly conflicting services. Therefore, the modification of an activity's processing rate to achieve one goal may be counterproductive to the attainment of another goal. It is on this

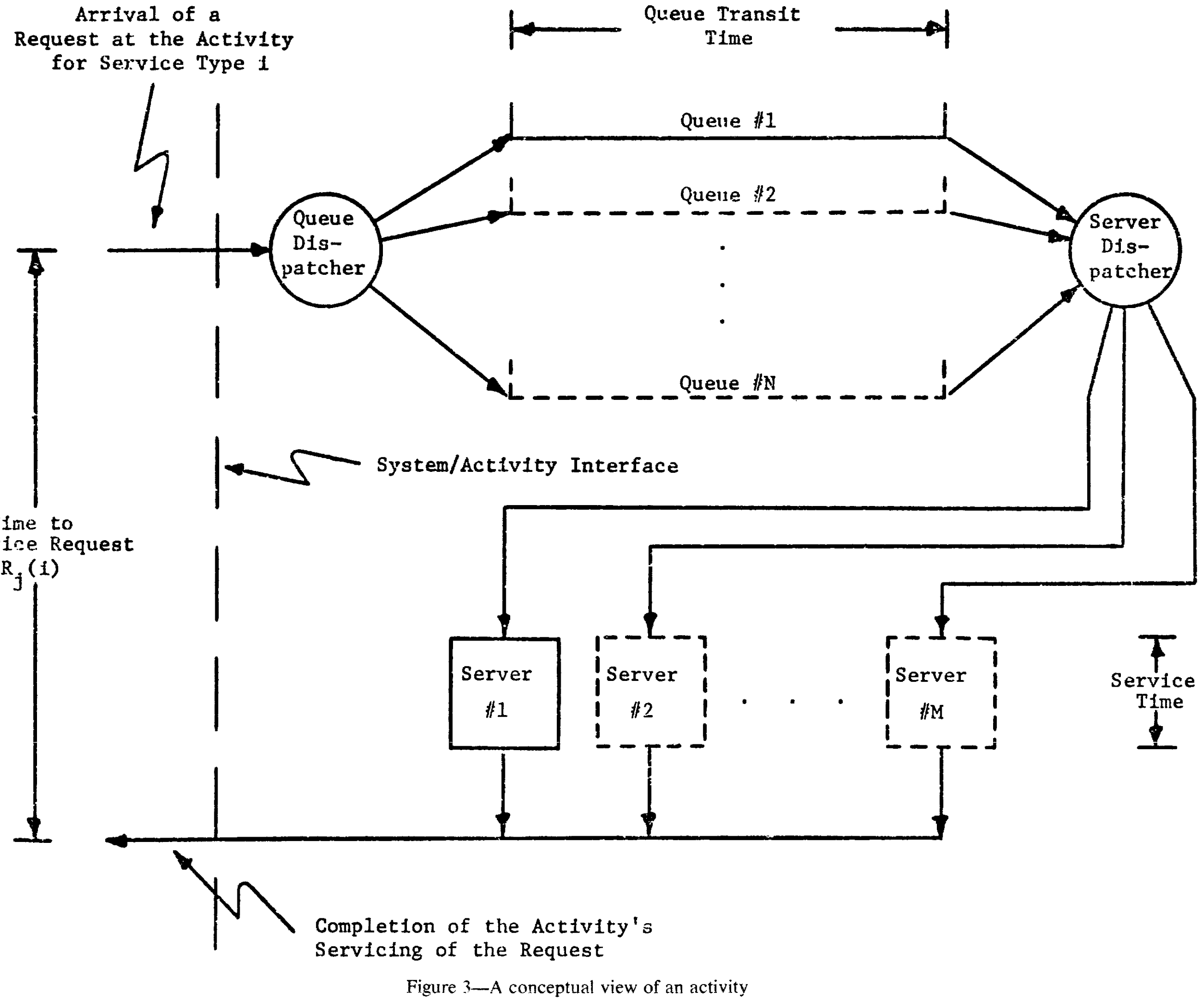


Figure 3—A conceptual view of an activity

possibility of multiple conflicting interactions and goals that this methodology is focused.

FORMULATION OF THE STAGE 2 EVALUATIVE PROCEDURE

Stage 2 is based on an evaluative procedure commonly called multiple goal programming (MGP). The procedure was first formulated by Charnes and Cooper[22] in 1961 to solve linear programming problems that had conflicting constraints. Ijiri[23] developed the details of the procedure within the framework of mathematical programming. This technique has been used to solve problems in the areas of strategic management planning such as accounting control,[23] advertising-media planning (Charnes and Cooper[24]), and resource allocation.[25] The employment of goal programming in conjunction with information system performance evaluation is a new use of the procedure.

There are three reasons for choosing multiple goal programming for use in this stage of the methodology. First, this approach can evaluate linear and ordinal multiple goal situations, both of which are inherent to information systems evaluation. For example, one user class may pay twice as much for its service, and, therefore, satisfaction of its goals may be worth twice as much as others; a linear relation. On the other hand, certain users, such as a critical patient monitoring application, may have incomparable importance relative to other classes; an ordinal relation. Second, multiple goal programming produces a solution that not only evaluates the total goal situation, but also evaluates each goal, individually. One of the purposes of this methodology is to determine the critical user classes and associated activities. Third, MGP derives the "best" design under the given goal constraints. Other design approaches such as weighting, sequential elimination, and spatial proximity,[26] are based on selecting the "best" design from a finite set of alternatives. The purpose of the overall methodology, however, is to *design* an appropriate system to satisfy the user and resource constraints. The standard formulation of a multiple goal programming problem is:

[A] Minimize $P \cdot D$
Subject to $A \cdot X + D = G$
where
A=a matrix of technological coefficients which can be thought of as the rates at which the ith service uses the jth resource
X=the array of resulting system resource allocation levels
G=the array of service goals
D=the array of discrepancies from these goals
P=the array of penalties associated with the discrepancies in D

and where the objective function is to minimize the product of the discrepancies and their associated penalties. The solution to a multiple goal programming problem represents the best set of levels for the resource allocation vector X such that the objective function is minimized. The remainder of this section discusses the specific formulation for the Stage 2 component of the methodology.

Figure 4 illustrates the relationship between MGP, the information system activities, and its servicing of user requests. The servicing of a request type i is a sequence of activities, $A_1, A_2, \ldots, A_n$, each assumed to be measurable by $R_j(i)$. In general, the measure can be a function $M(R_j(i))$ of the service time, however, just $R_j(i)$ will be employed in the following discussion.

The performance of the information system for service type i is given by the relation

$$T_S(i)= \sum_{j=1}^{n} (R_j(i)) \tag{1}$$

where $T_S(i)$ is the average system response time to service requests of type i. Assuming that the performance goal for the service is $T_G(i)$, then the discrepancy between performance and goal is given by

$$D(i)=T_G(i)-T_S(i). \tag{2}$$

The objective of MGP is to determine new performance levels for each activity in such a manner that the weighted discrepancy, $P \cdot D$, is minimized (hopefully to zero). Letting β_j be a scaling factor to be applied to the jth activity, then the new performance level for the activity is $R_j(i)\beta_j$. Incorporating the β_j's into equation (1) results in the following expression for the discrepancies:

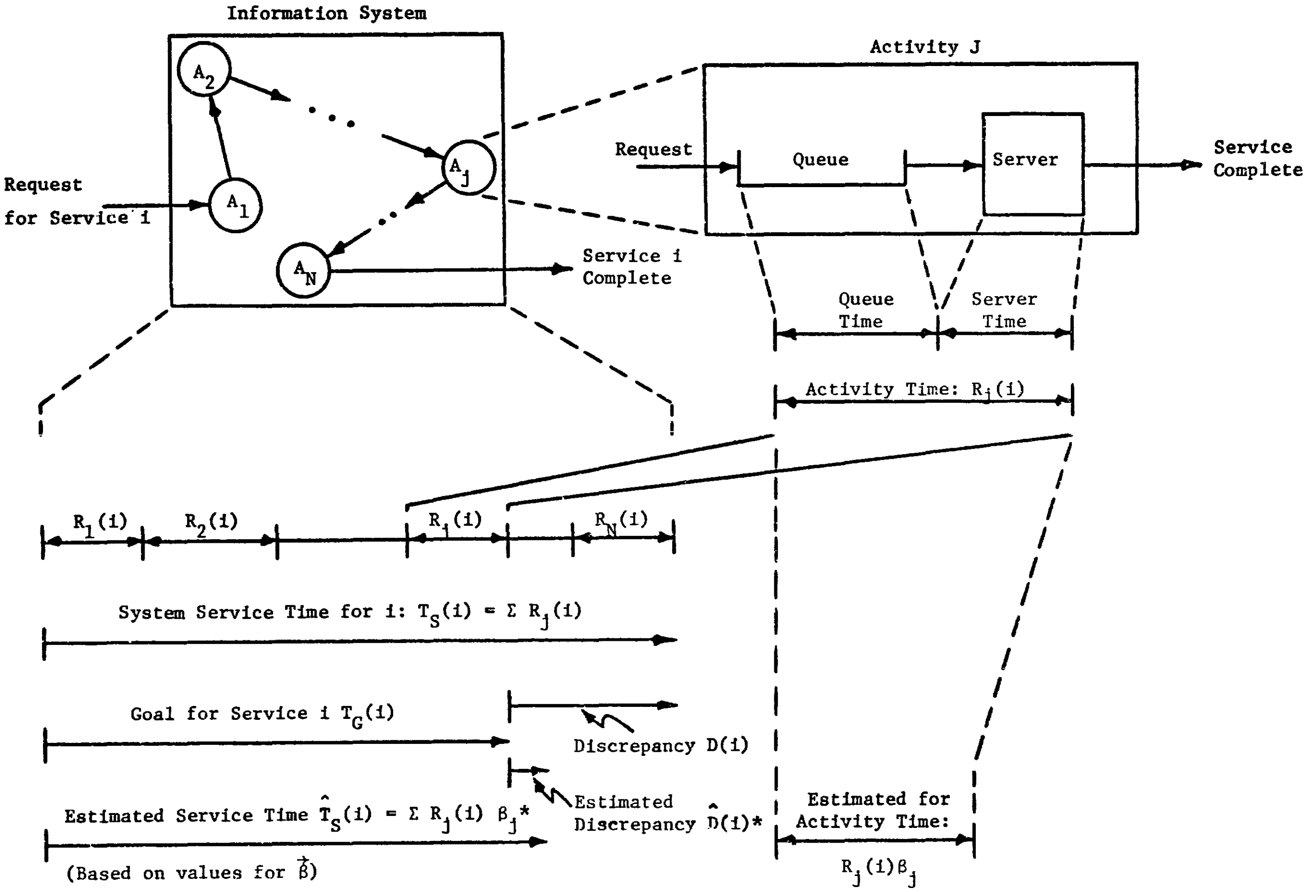
$$D(i)=T_G(i)- \sum_{j=1}^{n} (R_j(i) \cdot \beta_j). \tag{3}$$

Observe that both positive and negative discrepancies are possible, and, therefore, the formulation of the user goal evaluation as a MGP problem becomes:‡

[B] Minimize $P_D{}^+ \cdot D^+ + P_D{}^- \cdot D^-$
s.t. $R \cdot \beta + D^- - D^+ = G$
where
R is a matrix of service rates
β is the array of scaling factors
G^+ is the array of user goals
D^+ and D^- are the arrays of, respectively, the positive and negative discrepancies from the user goals
P^+ and P^- are the arrays of penalties associated with the corresponding positive and negative discrepancies.

The formulation serves two purposes: it evaluates goal achievement and produces the β_j's. By setting the values of the β's to reflect only the current configuration (i.e., $\beta_j = 1$),

‡ The complete derivation appears in a previous paper by the authors presented at the Annual Conference of the Computer Measurement Group, November, 1976.[27]



*Values of β_j's and D(i)'s are the outputs of Goal Program Optimization Problem

Figure 4—Relationship between goal programming and information systems characterization

the evaluation of the system's attainment of the user goals is accomplished.

Experiments with formulation [B] produced valid, but impractical sets of β's. The MGP problem as stated allowed for the possibility of solutions where a β_j could equal 0, clearly an unacceptable situation. In order to inhibit this type of solution, limits were placed on the range of possible β_j values. This was accomplished with the following set of additional constraints:

$$\beta_j+\mu_j^- -\mu_j^+ = L_j \qquad (4)$$

$$\beta_j+\nu_j^- -\nu_j^+ = H_j \qquad (5)$$

where

$$0<L_j\leq 1$$

$$1\leq H_j$$

L_j is used to restrict the alternative possibilities for the case that $\beta_j<1$ while H_j is used for those cases that $\beta_j>1$. In general, the set of all positive discrepancies for L_j's, $(\mu_1^+, \mu_2^+, \ldots, \mu_j^+)=M^+$ and likewise for $H_j(\nu_1^+, \nu_2^+, \ldots, \nu_j^+)=N^+$. ($M^-$ and N^- have similar definitions).

These constraints are reflected in the objective function in a manner different than previous constraints. Instead of minimizing both discrepancies, only one is minimized. In the case of L_j only μ_j^- is included, since, if μ_j^- is driven to zero, then $R_j(i)\beta_j-\mu_j^+=L_j$, implying that $R_j(i)\beta_j>L_j$, the desired condition. Similarly, for H_j only ν_j^+ is in the objective function because minimizing ν_j^+ results in $R_j(i)\beta_j<H_j$.

These added constraints also have a physical interpretation relative to the evaluation of the system. No activity can be eliminated from a system (i.e., $\beta_j=0$) since L_j must be greater than 0. In general, however, L_j represents the lower bound on the degree of reduction feasible for the current rate of usage for an activity. For example, $L_j=.25$ implies that the usage rate for activity j can be made, at most, four times faster, being reduced to 25% of its current rate. Similarily, H_j represents the upper bound on the degree to which an activity's rate can be increased (slowed down). It must be emphasized that these limits are only rough estimates, not exact values.

In order to reduce the number of alternatives one should minimize the number of modifications indicated per evaluation iteration. Since modifications are characterized by the production of β_j's not equal to 1, a secondary objective of Stage 2 is to produce as few $\beta_j \neq 1$ solutions as possible. This is accomplished by including the constraint equation

$$\beta_j+\epsilon_j^- - \epsilon_j^+ = 1 \qquad (6)$$

while minimizing both ϵ_j^+ and ϵ_j^-

Constraints of this type provide a default value of 1 for the multiple goal programming procedures in the case where an activity is neither critically inefficient or excessive. (Note: $(\epsilon_1^+, \epsilon_2^+, \ldots, \epsilon_j^+) = E^+$.)

As a result of these added constraints, the actual formulation of the MGP problem used in Stage 2 is given in formulation [C] below:

$$\begin{aligned} \text{[C] Minimize} \quad & P_D^+\cdot D^+ + P_D^-\cdot D^- + P_M^-\cdot M^- + P_N^+\cdot N^+ + P_E^+\cdot E^+ + P_E^-\cdot E^- \\ \text{s.t.} \quad & R\cdot\beta + D^- - D^+ = G \\ & \beta + M^- - M^+ = L \\ & \beta + N^- - N^+ = H \\ & \beta + E^- - E^+ = \bar{1} \end{aligned}$$

where

R is the matrix of service rates

β is the array of scaling factors

G, L, H, and $\bar{1}$ are the arrays of goals for the user criteria and the respective β constraints

$D^\pm$, $M^\pm$, $N^\pm$ and $E^\pm$ are the arrays of positive and negative discrepancies from the respective goals

$P_D^\pm$, $P_M^\pm$, $P_N^\pm$, and $P_E^\pm$ are the arrays of penalties for the associated discrepancies.

The solution variables for the MGP problem are the β's. They identify those activities that must be altered in order to improve user based or system based performance. If the value for a $\beta_j = 1$ then the service characteristics of activity j were adequate to satisfy all the user's criteria. If a $\beta_j < 1$, this implies that the current service rate of activity j is insufficient to meet the system's needs. The new service rate for the activity should be $R_j(\cdot) = (\beta_j) * (R_j(\cdot))$. If a $\beta_j > 1$ then the current service rate of activity j is faster than necessary and there exists the possibility of excess capacity. The new unit rate should be $R_j(\cdot) = (\beta_j) * (R_j(\cdot))$.

Assuming that an activity follows the characterization in Figure 3, then the analyst has three avenues of action when a $\beta_j \neq 1$. First, he can analyze the queue dispatching discipline in order to increase queue throughput (or possibly replace it with a simpler one if $\beta_j > 1$). Second, he can alter the service rate characteristics of the servers, e.g., slower hardware devices. And third, he can increase (decrease) the degree of parallelism among servers, for example, by adding (removing) a second channel, controller, etc.

LIAISON WITH STAGE 1

The critical factors in the Stage 2 evaluation are the values for the $R_j(i)$'s needed by the MGP formulation. These values are calculated in Stage 1 and are the statistical measure produced vis-a-vis the simulation. The specific model to be evaluated is the result of the heuristic procedures constituting Stage 3. The liaison is based upon the assumption that an information system can be viewed as a collection of resource allocation and task management activities and user oriented services. This view is supported by the literature, e.g., Madnick[28] and Zurcher and Randall.[29] IPSS also views the modeling of an information system in a similar manner, and thus, facilitates the development of the formal liaison with the MGP user evaluation.

The view of the system taken in Stage 2 (as illustrated in Figure 2) has an analog in IPSS. Its basic modeling concept is that of a service as shown in Figure 5. The service is classified in IPSS as a procedural facility and is capable of representing any information system activity including request (job) scheduling, task management, resource allocation, secondary storage I/O processing and application software. Since services are allocatable facilities in IPSS they have associated with them both queueing and utilization statistics. Furthermore, service behavior can be predicated on the quantity and characteristics of other IPSS hardware and software facilities. Therefore, the statistics associated with service facilities have the appropriate structure to service as the R_j's needed in Stage 2.

To complete the formal liaison between the two stages a second feature is employed. This is the Task facility. Through its use, the service facility statistics can be automatically segregated into service statistics by user. In this manner, the statistic $R_j(i)$, required by Stage 2, is collected. Thus, the Stage 2 users (indexed by i) and the activities (indexed by j) are, respectively, an IPSS model's Task and Service facilities. An $R_j(i)$ is the sum of queueing and utilization statistics gathered for Service i when executing Task j.

Stage 1 must also be adaptive to model changes dictated via Stage 3. Again, the IPSS model synthesis philosophy and language constructs permit the desired adaptiveness. This is possible for reasons too detailed to discuss in this paper. A complete description of IPSS is available in the document titled "The Information Processing System Simulator (IPSS): Language Syntax and Semantics."[30] Briefly, however, possible modifications to an existing and executing model without requiring complete reformulation include

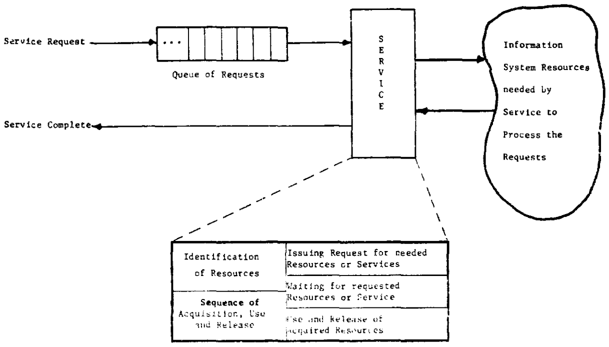

Service Request
...
Queue of Requests
SERVICE
Service Complete
Information System Resources needed by Service to Process the Requests
Identification of Resources
Sequence of Acquisition, Use and Release
Issuing Request for needed Resources or Services
Waiting for requested Resources or Service
Use and Release of acquired Resources

Figure 5—Functions of IPSS service entities

changes to:

1. timing and space characteristics associated with secondary storage hardware and storage media,
2. the secondary storage I/O configuration,
3. user usage patterns and service requirements,
4. file organization methods and space management policies,
5. the queueing disciplines associated with job scheduling, resource allocation and task management, and
6. memory management policies.

IPSS supplies the Stage 1 processing with a capability of being self-adaptive with respect to Stage 3 outputs. Currently, the methodology employs modeler assistance in Stage 3. Future research will be directed at providing more sophisticated heuristics for Stage 3 in order to provide a truly self-contained iterative methodology for the multiple criteria evaluation of information systems.

STAGE 3 ANALYSIS

The functions of Stage 3 of the methodology are to determine whether the current configuration is satisfactory and to formulate a new model in light of the data provided from Stage 1 and Stage 2. Figure 6 shows the information flow to Stage 3. New models reflect the performance goals of both the user and the system. Heuristics using Sutherland's* definition of a heuristic are employed in Stage 3.

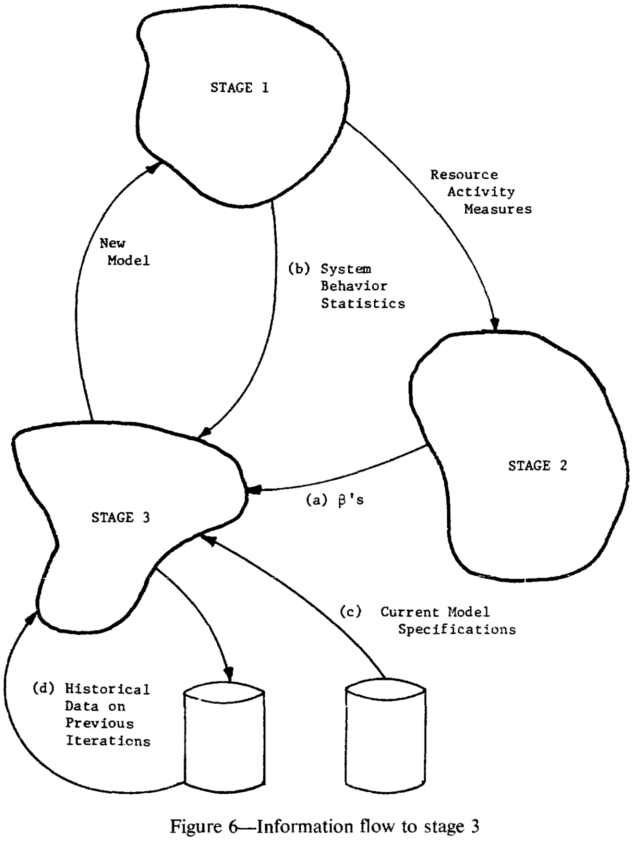
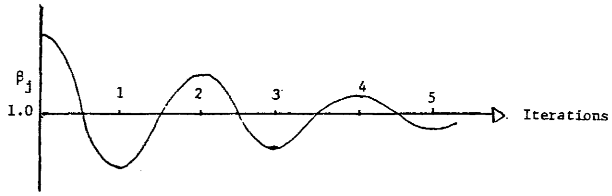

STAGE 1
Resource Activity Measures
New Model
(b) System Behavior Statistics
STAGE 2
STAGE 3
(a) β's
(c) Current Model Specifications
(d) Historical Data on Previous Iterations

Figure 6—Information flow to stage 3

The problems encountered are complex and unstructured. Determining if the current design is acceptable requires a mixture of objective and subjective reasoning. It would be a rare situation if all the user goals and system constraints were satisfied simultaneously. Generally, an extremely wide spectrum of acceptable performance levels and alternative designs are possible, at each iteration, to satisfy both user and system criteria. Trade-offs will dominate the decision processes. Many factors effecting suitable designs may not be included in the formulations and procedures of the first two stages. For example, there may be external political, organizational or economic considerations that are not directly related to the performance of the system, but may be a major factor in the final decision. The methodology does assume, however, that the heuristic procedures have access to this external criteria.

When it has been determined that another iteration is desirable, it is assumed that the heuristics will examine current and past designs. Whatever the heuristic employed, ideally its objective is to produce a sequence of models whose β_j characteristics (for all β's) behave as follows.



The emphasis of the current research is to provide insight into the decision process for improving the performance of information systems. Stage 3 is this decision process. It is aided by input from four sources within the methodology. These sources are: (a) the Stage 2 outputs, specifically the β_j's identified to improve user and system goal performance, (b) system behavior statistics from Stage 1, (c) the current model, and (d) historical data from prior iterations. It should be emphasized that at this juncture in the development of the methodology no formal heuristic procedures have been implemented. It is one of the purposes of this research, however, to investigate the appropriateness and success of various heuristic decision rules. Rules of thumb such as those proposed by Buzen[21] are possible avenues to be investigated.

AN EXAMPLE

In order to validate the procedures developed in this methodology and the liaison between Stage 1 and Stage 2, a test case was developed. This example was modeled and executed in IPSS to satisfy the Stage 1 requirements.

* A heuristic is "a disciplined trial-and-error process, . . . , an exercise in successive improvement, where we may learn from both success and failure and where the criteria for success and failure may vary with what we have previously learned" (p. 183, Reference [31]).

TABLE I—User Goal Evaluation

Goal	Over-Achievement	Under-Achievement
$USER_1$	26.6	0.0
$USER_2$	0.0	51.5
$USER_3$	0.0	258.5

Several iterations were applied, demonstrating the evaluative capabilities of the methodology. The following is a description of the problem, the corresponding model and the results of the first two iterations.

The example is a model of an on-line document retrieval system. There are three files associated with the system; an author/title index (A/T), a system document identification file (ID), and the document file itself (DOC). They are structurally related such that an entry in the A/T file points to one or more entries in the ID file and each ID file entry in turn is associated with only one DOC entry.

The model is designed so that a unique activity is associated with the accessing of each file; Activity 1 with the A/T file, Activity 2 with the ID file and Activity 3 with the DOC file. Each activity performs similar functions: obtaining and releasing devices, reading records, performing I/O techniques, but to different files.

It is assumed that the system supports three user classes, each with a different demand on the retrieval system and each characterized by a different combination of activities. The purpose of the first user class, $USER_1$, is to retrieve a document for a particular author, thus utilizing all three activities. Those in the second user class, $USER_2$, want to determine the existence/non-existence of an entry in the system for a given author, and therefore, need to use only Activity 1. The final user class, $USER_3$, already has the address of the ID entry and wants to retrieve the associated DOC entry requiring only Activity 2 and Activity 3.

In order to complete the formulation of the performance evaluation problem for this methodology, assumptions concerning the performance of the system were made. A summary of these assumptions for the user goals and β constraints and the penalties associated with the corresponding discrepancies in accordance to the requirements of formulation [C] is shown below.

$$
\begin{aligned}
G&=(\ 150.0,\ 100.0,\ 400.0)\\
L&=(\ .2,\ .2,\ .2)\\
H&=(\ 5.0,\ 5.0,\ 5.0)\\
\bar{I}&=(\ 1.0,\ 1.0,\ 1.0)\\
P_D^{+}&=(1000.0,\ 10.0,\ 1000.0)\\
P_D^{-}&=(\ 1.0,\ 0.0,\ 0.0)\\
P_M^{+}=P_N^{-}=P_E^{+}=P_E^{-}&=(\ 1.0,\ 1.0,\ 1.0)
\end{aligned}
$$

The model constructed in IPSS assumed a simple configuration of one processor and one bank of IBM 2314 type direct access devices. Under a given loading (which is not a controllable variable in this methodology) the resulting performance statistics, the values of matrix R, are shown below.

$Q_1(1)=0.0$	$Q_2(1)=0.0$	$Q_3(1)=11.4$
$S_1(1)=\underline{37.2}$	$S_2(1)=\underline{37.1}$	$S_3(1)=\underline{90.9}$
$R_1(1)=37.2$	$R_2(1)=37.1$	$R_3(1)=102.3$
$Q_1(2)=12.2$	$Q_2(3)=0.0$	$Q_3(3)=8.9$
$S_1(2)=\underline{36.3}$	$S_2(3)=\underline{39.1}$	$S_3(3)=\underline{93.5}$
$R_1(2)=48.5$	$R_2(3)=39.1$	$R_3(3)=102.4$

This performance information, coupled with the goal assumptions, was input to Stage 2. The evaluation of the current configuration's performance with respect to the set of user's goals is shown in Table I. It indicates that the goals of user classes 2 and 3 were satisfied with a good margin of slack, (which is not penalized in this example) while the goal of the first user class was not satisfied (over-achievement implying non-satisfaction).

In the second phase of Stage 2, the β's are allowed to be manipulated until they satisfy the user goal constraints and best suffice the system guideline constraints. The result is the identification of these activities whose performance can be, and need to be, improved with respect to one or both of the criteria. The values for the β's as calculated were

$$\beta_1=1.0 \qquad \beta_2=1.0 \qquad \beta_3=.74$$

These are interpreted as indicating that both Activity 1 and Activity 2 were adequate to meet the user demands put to them. Activity 3, however, was found to be insufficient to satisfy the requirements of user classes 1 and 3. The modification indicated is to reduce the present rate of usage for Activity 3 by at least 1/4 in order to satisfy the user goals, in particular, the first user class goal.

The determination of whether to cease the design loop by accepting this performance or to continue by modifying the existing model is made in Stage 3. Given the stated goal/penalty structure, it was assumed that the over-achievement of $USER_1$ goal was at an unacceptable level and the design process must continue if possible. By examining the queueing and service time statistics for the first iteration, one can eliminate some of the modification possibilities. The result of Stage 3 analysis was a decision to replace the IBM 2314 type device with a faster one, i.e., an IBM 3330 type device.

The original model of this example system was dynamically altered to reflect this modification. Under the same loading as before, the following performance statistics were accumulated.

$Q_1(1)=0.0$	$Q_2(1)=0.4$	$Q_3(1)=3.8$
$S_1(1)=\underline{11.9}$	$S_2(1)=\underline{11.3}$	$S_3(1)=\underline{27.6}$
$R_1(1)=11.9$	$R_2(1)=11.7$	$R_3(1)=31.4$
$Q_1(2)=3.8$	$Q_2(3)=0.0$	$Q_3(3)=0.0$
$S_1(2)=\underline{11.1}$	$S_2(3)=\underline{12.0}$	$S_3(3)=\underline{29.1}$
$R_1(2)=14.9$	$R_2(3)=12.0$	$R_3(3)=29.1$

Stage 2 analysis showed that now all three user class goals were satisfied (i.e., not over-achieved). The calculation of the β's, however, indicated that while Activities 1 and 2 were still adequate ($\beta_1=\beta_2=1.0$), Activity 3 now had the possibility of excess capacity ($\beta_3=4.0$). Although a slower and probably less expensive device for Activity 3 would be more appropriate, we had found in the first iteration that such a device was not able to satisfy all the user goals. Therefore, in future iterations, Stage 3 procedures had to examine more subtle methods of improving performance.

CONCLUSION

Modern information systems do not exist as entities unto themselves, but must interact with their environment, i.e., their users. The loading and mix of the users effect the performance of the system resources and likewise, the service characteristics of the system resources effect the satisfaction of user goals. In order to design such systems, one must satisfy a large set of users demanding a conflicting set of performance goals while operating within efficiency and minimum cost constraints. Thus, performance evaluation of information systems is a multiple criteria problem. Concurrently satisfying both of these sets of criteria is the goal of this methodology. Current available techniques, however, only address one side of the problem, either the user or the system. The methodology described in this paper establishes a formal liaison between the evaluation of user goals as a function of system behavior and the analysis of system resource performance as a function, of user demand, thereby, facilitating multi-criteria evaluation.

The methodology is iterative, comprising three separate but integrated stages. The first stage models and evaluates system behavior. The particular technique employed in the first stage is IPSS and it is able to collect the necessary statistic, $R_j(i)$. The second stage evaluates the user based criteria and provides evaluative insight into performance improvement. Solution of the MGP formulation in [C] produces a set of β's, the variables of Stage 2 which indicate inefficiencies and/or excesses in the current model. And finally, the third stage heuristically determines the current model's acceptability and need for modifications.

The evaluative procedures developed for this methodology have been shown to be valid in practice. Furthermore, this methodology provides an excellent basis for continued research into areas such as:

a. investigation into the causal relationships between user demand and system activity,
b. sensitivity analysis of these relationships,
c. investigation into suitable heuristics for Stage 3, either testing existing heuristics or development of new ones, and
d. development of heuristic/modification rules to close the design loop into an automatic self-modifying process.

REFERENCES

1. Boehm, B., and T. E. Bell, "Issues in Computer Performance Evaluation: Some Consensus, Some Divergence," *PER*, Vol. 4, No. 3, 1975, pp. 4-39.
2. Gaver, D. P., and G. Hunfeld, "Multitype Multiprogramming: Probability Models and Numerical Procedures," *Proc. of CPMME,* 1976, pp. 38-43.
3. Buzen, J. P., "Computer Algorithms for Closed Queueing Networks with Exponential Servers," *CACM*, Vol. 16, No. 9, 1973, pp. 527-531.
4. Neilson, J. E., "An Analytic Performance Model of a Multiprogrammed Batch Time-Shared Computer," *Proc. of CPMME,* 1976, pp. 59-70.
5. Conger, C. R., "The Simulation and Evaluation of Information Retrieval Systems," Report 352-R-17, April 1965.
6. Roehrkasse, R. C., and D. Smith, "Simulation of Operating Systems," Tech. Report GITIS-70-11, 1970, Georgia Inst. of Tech.
7. Abate, J., H. Dubner and S. B. Weinberg, "Queueing Analysis of the IBM 2314 Disk Storage Facility," *JACM*, Vol. 15, No. 4, 1968, pp. 577-589.
8. Nahourii, E., "Direct Access Device Simulation," *IBM Systems Journal*, Vol. 13, No. 1, 1973, pp. 19-31.
9. Sherman, S. W., and R. C. Bric, "I/O Buffer Performance in a Virtual Memory System," *Symposium on the Simulation of Computer Systems,* 1976, pp. 24-35.
10. Hellerman, H. R., and H. J. Smith, "Throughput Analysis of Some Idealized Input, Output and Computer Overlap Configurations," *Computing Surveys,* Vol. 2, No. 2, 1970, pp. 111-118.
11. Kleinrock, L., and R. R. Muntz, "Processor-Sharing Queueing Models of Mixed Scheduling Disciplines for Time-Shared Systems," *JACM,* Vol. 19, No. 3, 1972, pp. 464-482.
12. Agrawala, A. K., and R. L. Larsen, "Experience with the Central Server Model on a Lightly Loaded System," *Symposium on the Simulation of Computer Systems IV,* 1976, pp. 102-109.
13. Lewis, P. A. W., and G. C. Shedler, "A Cyclic-Queue Model of System Overhead in Multiprogrammed Computer Systems," *JACM*, Vol. 18, No. 2, 1971, pp. 199-220.
14. Gonzalez, G., "Using Covariance Analysis as an Aid to Interpret the Results of a Performance Measurement," *Proc. of CPMME,* 1976, pp. 179-186.
15. Cantrell, H. N., and A. L. Ellison, "Multiprogramming System Performance Measurement and Analysis," *AFIPS Conf. Proc.*, Vol. 22, 1968, pp. 213-221.
16. Reeves, T. E., and U. W. Pooch, "A Multiple Subsystem Simulation of Processor Scheduling," *Symposium on the Simulation of Computer Systems III,* 1975, pp. 129-135.
17. Norland, K. E., and W. C. Bulgren, "A Simulation Model of GECOS III," *Proc. of ACM*, 1971, pp. 596-612.
18. Lum, V. Y., Ling, H., and Senko, M. E., "Analysis of a Complex Data Management Access Method by Simulation Modeling," *FJCC,* 1970, pp. 211-222.
19. Lindsay, D. S., "A Hardware Monitor Study of a CDC KRONOS System," Proc. of CPMME, 1976, pp. 179-186.
20. Hall, W. A., "A Simulation Model to Aid in the Design and Tuning of Hierarchical Databases," *Winter Simulation Conference,* 1974, pp. 277-284.
21. Buzen, J. P., "Fundamental Laws of Computer Performance," in *Proc. of Int'l. Symp. on Computer Performance Modeling, Measurement and Evaluation* (CPMME), 1976, pp. 200-210.
22. Charnes, A., and W. W. Cooper, *Management Models and Industrial Applications of Linear Programming,* New York, John Wiley & Sons, Inc., 1961.
23. Ijiri, Y., *Management Goals and Accounting for Control,* Chicago, Rand McNally, 1965.
24. Charnes, A., et al., "A Goal Programming Model for Media Planning," *Management Science,* Vol. 14, No. 8, April 1968, pp. 423-430.
25. Lee, S. H., *Goal Programming for Decision Analysis,* Philadelphia, Auerbach, 1972.
26. MacCrimmon, K. R., "An Overview of Multiple Objective Decision

Making," *Multiple Criteria Decision Making,* eds., J. L. Cochrane and M. Zeleny, Columbia, S. C., 1973, pp. 18-44.

27. Chandler, J. S., and T. G. DeLutis, "A Methodology for the Performance Evaluation of Information Systems Under Multiple Criteria," *Proc. of Computer Measurement Group*, 1976, pp. 221-230.
28. Madnick, S., and J. Donovan, *Operating Systems,* N. Y., McGraw-Hill, 1974.
29. Zurcher, F. W., and B. Randall, "Iterative Multi-Level Modelling: A Methodology for Computer System Design," *IFIP 68,* pp. 867-871.
30. DeLutis, T. G., "The Information Processing System Simulator (IPSS): Language Syntax and Semantics," unpublished research report (Grant No. G-36622).
31. Sutherland, J. W., *Systems: Analysis, Administration, and Architecture,* Van Nostrand Reinhold, New York, 1975.

Automated control of concurrency in multi-user hierarchical information systems

by ALAN F. SWEET and ARTHUR E. OLDEHOEFT
Iowa State University
Ames, Iowa

ABSTRACT

This paper presents a systematic approach to providing a high degree of concurrent access to information in hierarchically structured systems. An algorithm is presented which is designed to operate on the procedures and tree-structured information of two adjacent levels. The algorithm analyzes the procedure and structure refinements and generates the appropriate monitor calls to increase the degree of concurrent access. Assuming the initial level is correct, the refined system remains deadlock free, the integrity of the information is preserved, and individual procedures are checked for determinacy. Graph structured models are used to illustrate examples and definitions.

INTRODUCTION

In this paper, we present a systematic approach to providing a high degree of concurrent access to information structures in a hierarchical information system. The technique may be applied to existing systems which operate on tree-structured information and which appears as levels of functionally decomposed procedures. It may also be applied to the successive levels of procedures and information tree refinements during the top-down design process of information systems.

The basic approach is to analyze concurrent processes, at a given level, for interference and to automatically place requests for access capabilities to structures at those points where the structures are first referenced. Releases are automatically placed after those points where the structures are last referenced. As a new level of procedures are specified and the information trees are refined, the analysis is repeated resulting in the introduction of new controls and possible movement of old ones. Since the information is tree-structured, the access to a substructure may, in certain cases, result in the release of the access capabilities to the predecessor node in the tree, thereby clearing the way for concurrent access to substructures at the same level in different branches of the tree.

Numerous systems have been proposed and implemented using the concept of a hierarchical structure.[1,5,8,9] Our work, however, is restricted to subsystems (file systems, data base systems, etc.) dedicated to a single language which might typically run as a module in a host operating system. These systems manage their own resources and control the flow of information to the users.

The complicating factor, and the major focus of this paper, is the assumption of a multi-user environment in which each user interacts with the system sharing, and possibly modifying, information. It is now critically important that our design process satisfy two additional constraints;

(1) the introduction of potential concurrency wherever possible, and
(2) the guarantee that the system will be correct with respect to three problems of concurrency
  (a) preservation of information integrity among interfering independent processes
  (b) deadlock avoidance, and
  (c) determinacy within a single process

These two requirements place significant analytic burdens on the designer of such a system. Later, we will present an algorithm which can relieve the designer of these burdens. We first, however, more rigorously define the properties of the information system.

Process decomposition is based on a "uses" concept similar to that of Parnas.[9] We, however, define the level of a procedure in a top-down manner as follows:

(1) Level 0 is the outer-most level of the system,
(2) Level i is the set of all procedures which, if they do *use* any procedures, *use* only procedures at level i+1, and
(3) Level k, the inner-most level, is a set of procedures which *use* no other procedures.

Structures organized by this definition have three desirable properties. They are easy to test, since the interfaces between adjacent levels are the only procedure interactions that need be tested. All communication paths to system resources must use common procedures and, as a consequence, scheduling criteria can be more easily enforced

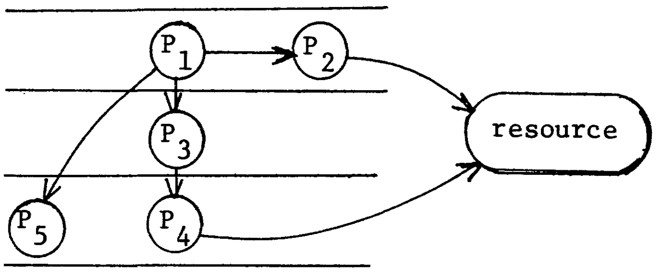


Figure 1—Original level structure and communication paths

with respect to the resources. Finally, we can guarantee a property (in the overall system) by ensuring that the property is maintained in each newly defined level.

Since this definition restricts the communication paths and does not allow intralevel communication between procedures, the structure may contain identity procedures (procedures which only perform a reference to a single procedure at the next level). This definition, for example, would require that a system whose communication and access path are as shown in Figure 1 be restructured to appear like that in Figure 2 with I_1 and I_2 as identity procedures. Identity procedures can increase the amount of overhead time spent as a result of procedure communication. Bernstein and Siegel[2] have recently proposed a solution to this problem with some simple hardware mechanisms which can decrease this overhead.

The algorithm we will present for placing access controls is applied to hierarchical systems like those we have just described. The algorithm assumes the existence of a monitor which can be used to control access capabilities to the structures. The algorithm will analyze and modify the procedures of two adjacent levels, placing monitor uses in the procedures to introduce additional concurrency and still guarantee the stated properties of correctness. In order to perform the modifications, the algorithm will require a parser of the procedural language and additional information about the operands of the language constructs. The monitor will have the unique property in that a process which has access capabilities to a structure will be allowed to request access capabilities to any substructures. This will allow the procedures to release the major structure while maintaining access to substructures. The result is an additional increase in potential concurrency.

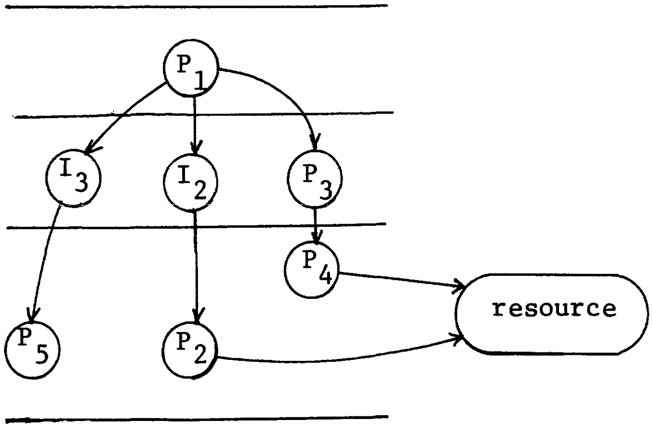


Figure 2—Restructuring of system with identity programs

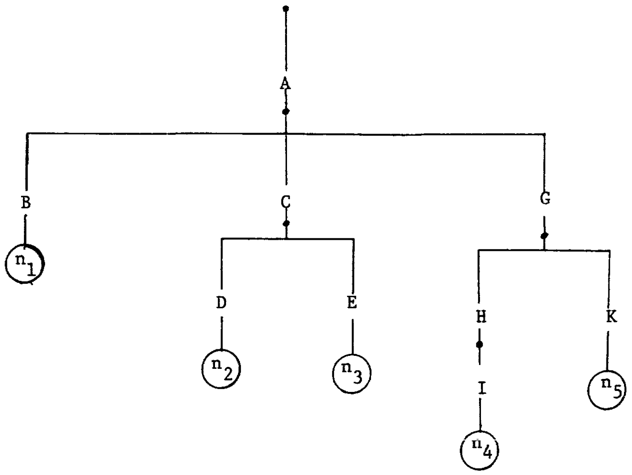


Figure 3—An example information structure

CORRECTNESS CONSIDERATIONS

Figure 3 shows an example of an information structure. Nodes of these structures are accessed by path names. For example, the node n_1 in Figure 3 is referenced by the path name A.B, and the substructure containing nodes n_2 and n_3 is referenced by the pathname A.C.

Let $P=u_1 \ldots u_n$ and $P'=v_1 \ldots v_m$ be path names. Then P is said to contain P', if $n \leq m$ and $u_i=v_i$ for all $i=1, \ldots,$ n. In Figure 3, the path name A.G contains A.G.H. For two sets of path names X and Y, we define:

$$X \mid Y=\{x_i \mid x_i \in X \text{ and } x_i \text{ contains some } y_j \in Y\}$$

The *path intersection* of X and Y is represented by X~Y and defined by the set union of X | Y and Y | X, i.e. (X | Y)∪(Y | X). For example, if X={A.B,A.C.D,A.G} and Y={A.C,A.C.H,A.G.K}, then X~Y={A.C,A.G}.

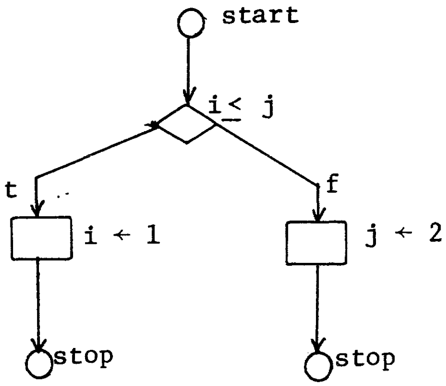


Figure 4—A example process graph

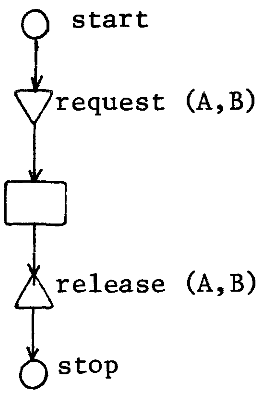


Figure 5—Example of request and release nodes

We use the concept of path intersection to extend Bernstein's[3] definition of noninterference. Let H_1 and H_2 be procedures with the domains of pathnames, D_1 and D_2, and the ranges of pathnames, R_1 and R_2, respectively. The H_1 and H_2 are defined to be mutually noninterfering, if either

(1) H_1 is a successor or predecessor of H_2, or
(2) $R_1 \sim R_2 = D_1 \sim R_2 = R_1 \sim D_2 = \emptyset$

This generalized definition, as it applies to tree-structures, forms the basis for analyzing interference and maintaining the integrity of the information in our system as well as checking for determinacy within a single process. For a procedure H, we use a notational convenience $Y=((R),(D))$ to represent the domain D and range of H. For H_1 and H_2 we define $Y_1 \sim Y_2 = \emptyset$ if $R_1 \sim R_2 = R_1 \sim D_2 = D_1 \sim R_2 = \emptyset$

The procedures in the system will be represented by flow graphs in which nodes represent operations and the arcs between nodes represents the sequential flow of control. Figure 4 illustrates a procedure graph and Figure 5 illustrates the special nodes which represent the monitor calls to *request* and to *release* access capabilities. Figure 6 illustrates the operations *cobegin* and *coend* for parallel execution within a procedure.

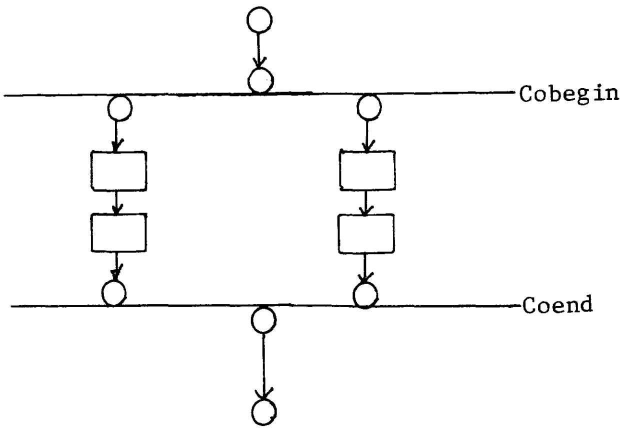


Figure 6—Example of two parallel paths

When independent procedures operate concurrently on shared structures and free access is allowed to the structures, the result may be unpredictable. The term critical region[4,6] has been used to describe that portion of a procedure which operates on a shared item. If a procedure H operates on the shared item A, then H'(A) is used to denote the critical region of H with respect to A. In terms of the procedure graph, H'(A) is the subgraph of H which may access A. This idea is illustrated in Figure 7.

Any execution sequence of the critical region H'(A) is represented by C(A). For the example in Figure 7, C(A) represents either of two sequences of operations, n_2n_3 or n_2n_4.

Let H_1 and H_2 be two procedures at the same level with critical regions $H_1'(A)$ and $H_2'(A)$, respectively. The procedures H_1 and H_2 are defined to be *mutually exclusive* with respect to A, if for all concurrent realizations of H_1 and H_2 either 1) $H_1'(A)$ or $H_2'(A)$ is empty, or 2) $C_1(A)$ executes prior to $C_2(A)$ or 3) $C_2(A)$ executes prior to $C_1(A)$. In Figure 8, the monitor is used to ensure mutual exclusion between two procedures H_1 and H_2.

Let a procedure H operate on a set of shared structures $A=\{A_1, \ldots, A_n\}$ with $H'(A)=\{H'(A_1), \ldots, H'(An)\}$ as corresponding critical regions for A. Let $C(A)=\{C(A_1), \ldots, C(A_n)\}$ represent the execution of the critical regions in H. For example in Figure 9, $A=\{A_1,A_2\}$ and $C(A)=\{C(A_1),C(A_2)\}$ where $C(A_1)=\{n_2n_4,n_2\}$ and $C(A_2)=\{n_3,n_4\}$. Suppose we have two procedures H_1 and H_2 at the same level which share such a set A of structures and let $S_0(A)$ denote the initial state of A. Let $S_1(A)$ denote the final state of A, if H_1 executes prior to H_2 and let $S_2(A)$ denote the final state of A if H_2 executes prior H_1. The procedures H_1 and H_2 are said to preserve the *information integrity* of A, if every concurrent execution of H_1 and H_2 either produces $S_1(A)$ or $S_2(S)$ as the final state. In Figure 10, for example, procedures H_1 and H_2 may not preserve

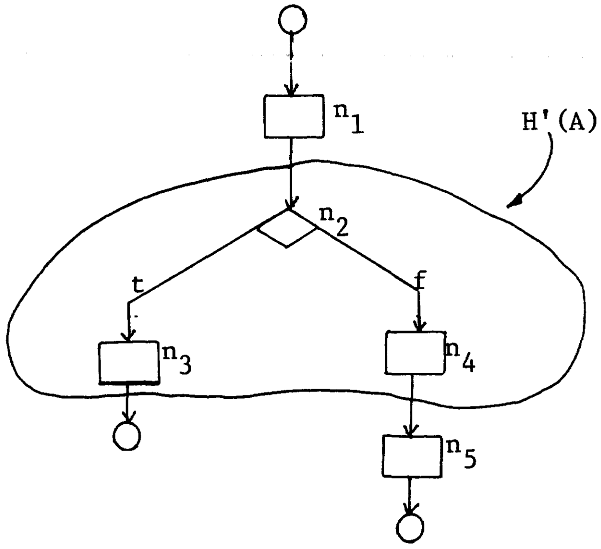


Figure 7—The process H and critical region H'(A)

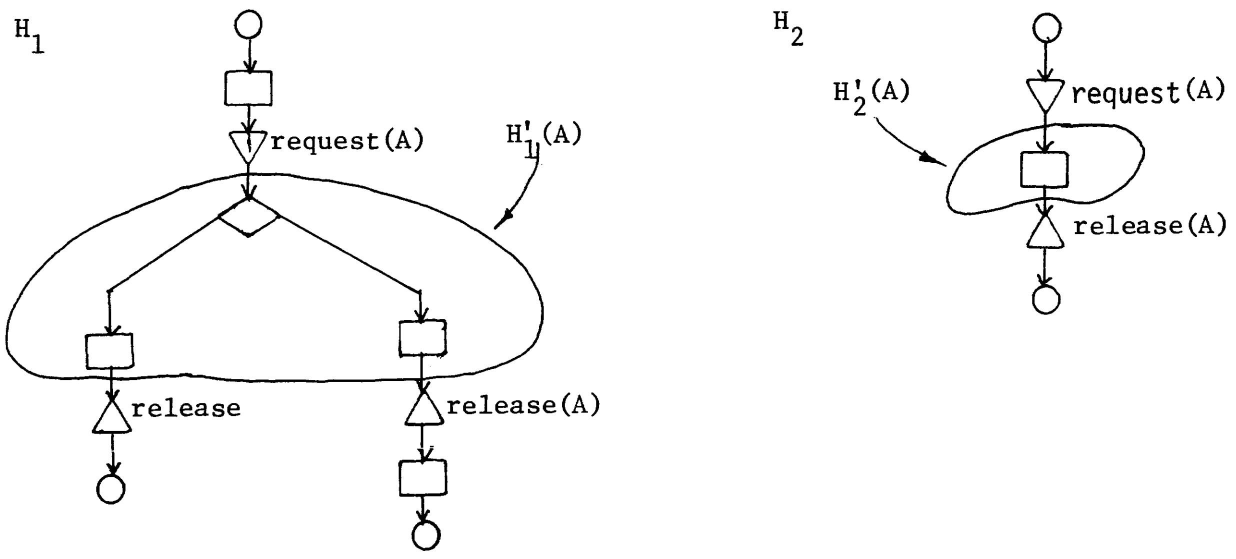


Figure 8—Monitor uses to ensure mutual exclusion

information integrity of $A=\{A_1,A_2\}$. In Figure 11, the monitor is used to ensure that H_1 and H_2 preserve information integrity in that the final state of A will always be {1,2} or {2,3}. In some cases, the requests can be rearranged to increase concurrency while maintaining integrity. Incorrect requests and releases, of course, introduces the potential of deadlock. These ideas are illustrated in Figures 12 and 13.

DESCRIPTION OF UNDERLYING MONITOR

The algorithm presented will assume the existence of a monitor similar to the type described by Hoare[7] and Brinch Hansen.[4] The monitor will have two usages, *request* (X) and *release* (X) where $X=\{x_1, \ldots, x_n\}$ is a list of path names. The request will be allowed only if logical access can be granted for all path names in the list. Otherwise, the procedure will wait until the entire request can be granted. Release returns the logical access capabilities to the path names in the list. The monitor operates on the following data structures:

$P(x_i)$ —the procedure with access capability to x_i
$W(x_i)$ —the set of procedures waiting for access capability to x_i, and
A —the set of path names to which access has been granted.

The monitor operations, as invoked by a procedure H, can be described as follows.

```
request (X):   if for every $x_i$ in X, $x_i \sim A = \emptyset$ or
                   ($x_i \sim A = z_i$ implies $P(z_i) = H$)
               then
                   $P(x_i) = H$ for all $x_i$ in X
                   $A = A \cup X$
               else
                   $W(x_i) = W(x_i) \cup H$ for all $x_i$ in X
                   place H in wait state
release (X):   $P(x_i) = \emptyset$ for every $x_i$ in X
               $A = A - \{A \cap X\}$
               If there exists some procedure H' and
                   some $x_i$ in X such that H' is in $W(x_i)$
                   and H' is waiting for the request (Y)
               then
                   if for every $y_i$ in Y, $y_i \sim A = \emptyset$ or
                   ($y_i \sim A = z_i$ implies $P(z_i) = H$)
```

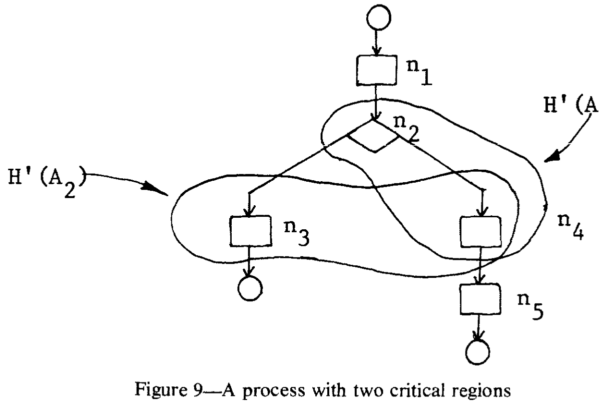


Figure 9—A process with two critical regions

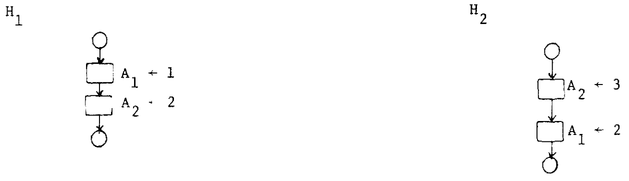


Figure 10—Processes operating on shared structures A_1 and A_2

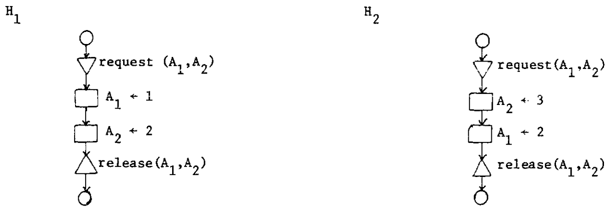


Figure 11—Monitor uses to ensure information integrity

```
then
  remove H' from wait state
  P(y_i)=H' for all y_i in Y
  W(y_i)=W(y_i)-H' for all y_i in Y
  A=A∪Y
```

DESCRIPTION OF THE ALGORITHM

The algorithm is designed to operate on the procedures and the information structures of two adjacent levels. We illustrate the result of applying the algorithm in Figures 14 through 19. The critical region for the procedure in Figure 14 is shown prior to the refinement of the structure A and the specification of the lower level procedures. Let $F(A)=(A.A_1,A.A_2)$ be the refinement of A and suppose the lower level procedure L_1 operates an $A.A_1$ while L_2 and L_3 operate on $A.A_2$. O_1 is assumed to be an operation on the major structure A.

The critical regions for A and F(A) are expressed as in Figure 15. The information integrity of A is preserved by the high level monitor uses in Figure 16. The potential concurrency may be increased, deadlock avoided, and information integrity preserved for A, if the procedures are modified with monitor uses to appropriately request substructures and release the major structure. Figure 17 shows how our algorithm will perform such placements and rearrangements of request and releases.

Suppose L_1 and L_3 of the previous example have the refinements as shown in Figure 18 with the critical regions $L_1'(A.A_1)$ and $L_3'(A.A_2)$ respectively. The potential concurrency is increased still further, if the releases for $A.A_1$ and $A.A_2$ are removed from the high level procedure H and placed in the low level procedures L_1 and L_3. Figure 19 shows how our algorithm will relocate these releases into the lower level procedures.

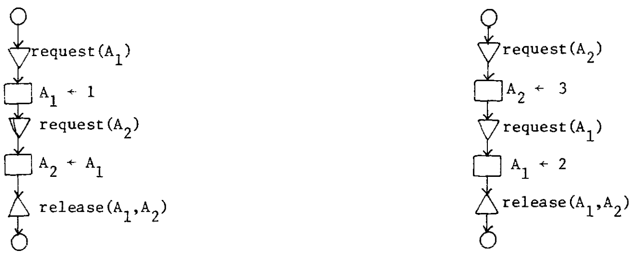


Figure 13—Monitor uses with potential deadlock

If every use of L is followed by a release of the structure A_i of A, and there exists no operations on A, other than releases, between the terminations of L and the corresponding releases, of A_i, then these releases of A_i are said to be movable with respect to L. This concept is exploited by our algorithm.

The algorithm operates over procedures and requires information about the domain and range operands of the language constructs. This information may be determined through a combination of precompilation of the procedures and direct specification by the designer in the case of procedure and function statements. The operands of each language construct are determined by the union of the operands of its parts. This is demonstrated in Figure 20 which shows a procedure and the operand sets Y, Y_1, Y_2, and Y_3.

The algorithm also uses a set of transformations defined over the procedural language which perform the actual placement of the requests and releases. For purposes of discussion, the algorithm is applied to an example language. There are four sets of transformations needed by the algorithm to place the controls in the procedures. These transformations along with the formal definition of the example language are given in the Appendix. These example transformations are for illustrative purposes and may be non-optimal. The algorithm initiates the transformations through the following four starting procedures:

(1) B1(H,X) places the initial requests for the set X of structures in the outer-most procedure H.

(2) C1(H,X) places the releases for the structure X in H.

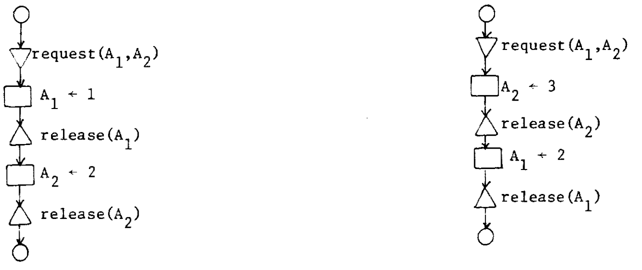


Figure 12—Monitor uses which may increase potential concurrency

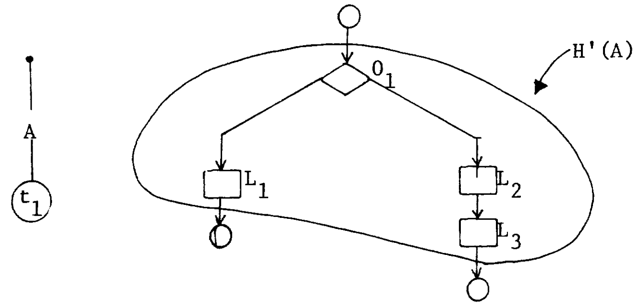


Figure 14—Critical region prior to structure refinement

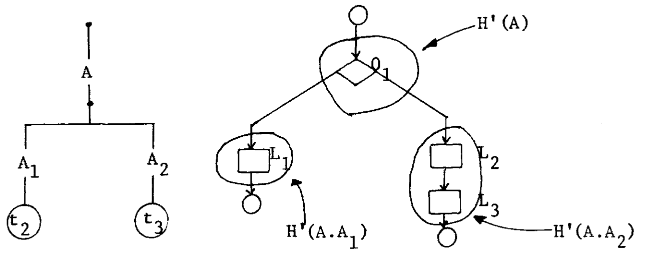


Figure 15—Critical regions after structure refinement

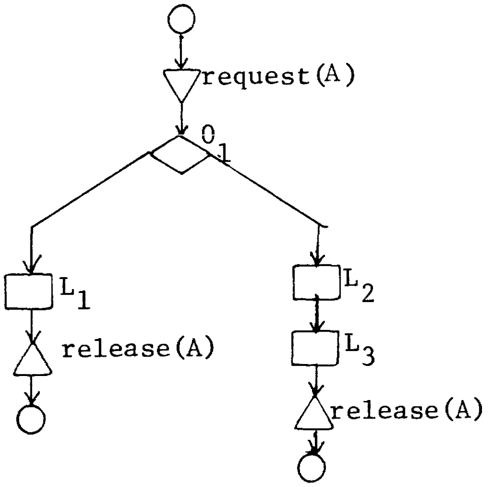


Figure 16—Procedure H with monitor uses for A

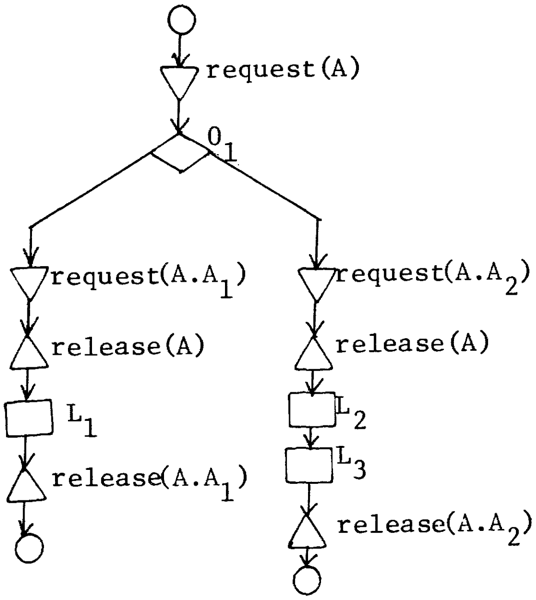


Figure 17—Procedure H with monitor uses for A, A.A.$_1$ and A.A.$_2$

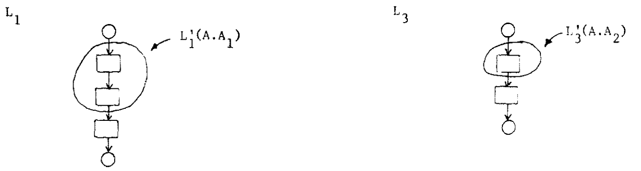


Figure 18—Low level procedures L_1 and L_3 with critical regions L_1' (A.A.$_1$) and L_3' (A.A.$_2$)

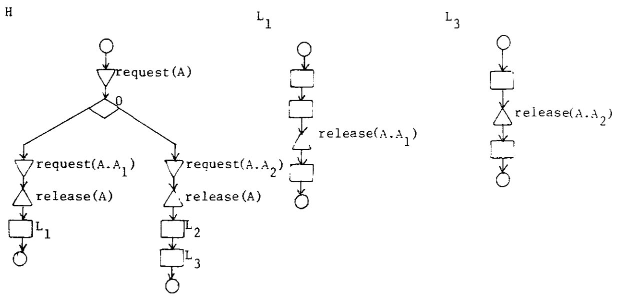


Figure 19—Procedures H, L_1, and L_3 with monitor uses for A, A.A.$_1$, and A.A.$_2$

(3) D1(H,X,F(X)) determines which of the refinements F(X) of X have critical regions outside the critical region H'(X) for the pathname X (see Figure 15 for example).

(4) E1(H,X',X) places the requests for the refinement X' of X in H.

As an example, suppose we wish to execute C1(H,X). This leads to an application of C4. Figure 21 shows the transformation C4 which operates over the "statement" construct to place a release for X. If the "statement" will parse as an "assignment statement" or "procedure statement," the release for X is placed immediately following the "statement." Otherwise, the transformation C5 is applied to the "statement," since it must parse as a "structured statement."

The algorithm uses one other set of functions to analyze the procedures for the sufficient conditions for determinacy. This set is initiated by A1(H) and returns the value *true*, if the procedure H satisfies these conditions. Otherwise, the value *false* is returned.

The entire algorithm is presented in Figure 22. Several procedures are assumed to support the algorithm.

(1) *input-outer-level-procedures (H,X∅)*
This is an initialization procedure which accepts as input the source code for the outer-level procedures H and the set of path names X∅ of their operand structures. In Figure 14, X∅={A}.

(2) *input-next-level-procedure (L,X2)*
This is the first step in the iterative process. The source code of the next level procedures L and their operand structures X2 are input. In Figure 15, we

```
procedure  L1(A,A1);                          Y=((A,A.A1),(A))=Y1∪Y2∪Y3
  begin
    if has2(A ,A1)                            Y1=((),(A))
    then begin
           erase2(A.A1)                       Y2=((A.A1),())
         end
    else begin
         create2(A,A1)                        Y3=((A),())
         end
  end
```

Figure 20—Procedure L_1 with operand sets Y, Y_1, Y_2 and Y_3.

```
C4(<statement>Y)
       ⎧ < assignment statement > release(x)
     = ⎨ < procedure statement > release(x)
       ⎩ C5(< structured statement > Y)
```

Figure 21—The transformation C4 for placement of releases

have X2={A.A_1} for L1, X2={A.A_2} for L_2, and X2={A.A_2} for L_3.

(3) *restate-high-level-operands-with-refinements (H,X1, XØ,L,X2)*

The operands for the constructs of the procedure H (as defined by the grammar) are restated in terms of the original structures XØ and the refined structure operands X2 accessed by the low level L. These are represented by X1. For example, in Figure 15, we have X1={A,A.A_1,A.A_2}.

(4) *remove-all-releases (H)*

All the releases are removed from the high level procedure H.

The algorithm will relocate and generate requests and releases as new levels are defined. An example of a high level procedure with requests and releases is given in Figure 23. This is prior to the input of the next level procedures. Figure 24 gives the next level procedures which operate on the refined structures. Figures 25 and 26 give the placement of the requests and releases after both levels have been processed by the Algorithm in Figure 22.

```
begin
  input-outer-level-procedures (H,XØ)
  if for any H' in H,A1(H')=false then stop
  for every H' in H with operands XØ' do B1(H',XØ')
     for every X  in XØ' do Ç1(H',X) end
  end
  while another level exists do
     begin
       input-next-level-procedures(L,X2)
       if for any L' in L, A1(L')=false then stop
       restate-high-level-operands-with-refinements(H,X1,XØ,L,X2)
       for every H' in H with operands X1' and XØ' do
          remove-all-releases(H')
          for every X in XØ' do
             U=D1(H,X,F(X))
             C1(H,X)
             for every Y in U do
                E1(H,Y,X)
                C1(H,Y)
             end
          end
       end
       for every L' in L with operands X2' do
          for every X in X2' do
             if releases for X are movable with respect to L'
             then do
                "remove release(X) with respect to L' from every
                   high level process"
                C1(L,X)
                end
          end
       end
       H ← L
       XØ ← X2
  end
end
```

Figure 22—The algorithm for analyzing levels and placing monitor uses

```
procedure update (Employee,input)
   begin
     request (Employee-records)
     if hasl (Employee-records, Employee)
     then
       begin
         modifyl(Employee-records.Employee,input)
         release (Employee-records)
       end
     else
       begin
         release (Employee-records)
         write ('Employee not in files')
       end
   end
```

Figure 23—The high level procedure 'update' with monitor uses

```
procedure hasl(records, selector):  returns boolean;
begin
  if selector in records
  then
    begin
      hasl:=true
    end
  else
    begin
      hasl:=false
    end
end

procedure modifyl(record, input)
  begin
    if input.selector in record
    then
      begin
        erase2(record.'input.selector')
        assign2(record.'input.selector',input.val)
      end
    else
      begin
        create2(record,'input.selector')
        assign2(record.'input.selector',input.val)
      end
end
```

Figure 24—The low level procedures 'modifyl' and 'hasl'

```
procedure update(Employee,input)
   begin
     request(Employee-records)
     if hasl(Employee-records,Employee)
     then
       begin
         request(Employee-records, Employee)
         release(Employee-records)
         modifyl(Employee-records.Employee,input)
       end
     else
       begin
         release(Employee-records)
         write('Employee not in files')
       end
   end
```

Figure 25—The high level procedure 'update' with refined monitor uses

```
procedure hasl(records,selector):  returns boolean;
   begin
     if selector in records
     then
       begin
         hasl:=true
       end
     else
       begin
         hasl:=false
       end
   end

procedure modifyl(record,input)
   begin
     if input.selector in record
     then
       begin
         erase2(record.'input.selector')
         assign2(record.'input.selector',input.val)
         release(record)
       end
     else
       begin
         create2(record,'input.selector')
         assign2(record.'input.selector',input.val)
         release(record)
       end
   end
```

Figure 26—The low level procedure with refined monitor uses

Note that in its refinement, the input type has two components, selector and val.

CONCLUSION

The method described in this paper has been extended to apply to systems whose procedures are expressed in a Pascal-like language, allowing for complex control structures. It has been applied to successive levels in the design of a personnel records information system.[10] The algorithm and its associated transformations, when used in the context described in this paper, illustrate a technique for automating the analysis needed to increase concurrency and still maintain correctness in hierarchically structured information system. This means that each level maintains the originally implied information integrity, deadlock will not be introduced, and any indeterminacy introduced by the designer into any individual procedures will be detected. This paper assumed only mutually exclusive access to structures. It is possible, however, to extend this technique by allowing concurrent readers and exclusive writers. This requires somewhat more sophisticated transformations and monitor support, but it will further increase the degree of potential concurrency.

REFERENCES

1. Burner, H. B., "An Application of Automata Theory to the Multiple Level Top-Down Design of Digital Computer Operating Systems," Ph.D. thesis, Washington State University, 1973.

2. Bernstein, A. J. and P. Siegel, "A Computer Architecture for Level Structured Systems," *IEEE Transactions on Computers,* C-24, No. 8, August 1975, pp. 785-793.
3. Bernstein, A. J., "Analysis of Programs for Parallel Processing," *IEEE Transactions on Computers,* EC-15, No. 5, October 1966, pp. 757-763.
4. Brinch Hansen, P., *Operating Systems Principles,* Englewood Cliffs, New Jersey, Prentice-Hall, 1973.
5. Dijkstra, E. W., "The Structure of the T.H.E.-Multiprogramming System," *CACM* 11, No. 5, May 1968, pp. 341-346.
6. Dijkstra, E. W., "Cooperating Sequential Processes," in *Programming Languages,* (F. Genuys, ed.), Academic Press, 1968, pp. 43-112.
7. Hoare, C. A. R., "Monitors: An Operating System Structuring Concept," *CACM* 17, No. 10, October 1974, pp. 549-557.
8. Liskov, B. H., "The Design of the Venus Operating System," *CACM* 15, No. 3, March 1972, pp. 144-149.
9. Parnas, D. L., "Some Hypotheses about the 'Uses' Hierarchy for Operating Systems," Technical report B2-2-76/1. Darmstadt, West Germany, Fachbereich Informatick, March 1976.
10. Sweet, A. F., "Correctness in Multi-User Hierarchically Structured Information Systems," Ph.D. Thesis, Computer Science Department, Iowa State University, Ames, Iowa, 1977.

Appendix

The BNF form of the Example Procedural Language

Rule No.	
G1	< procedure > = < procedure heading > < block >
G2	< block > = begin < statement list > end
G3	< statement list > = < statement >
G4	< statement list > = < statement > < statement list >
G5	< statement > = < assignment >
G6	< statement > = < procedure statement >
G7	< statement > = < structured statement >
G8	< statement > = < access control statement >
G9	< structured statement > = < block >
G10	< structured statement > = < conditional >
G11	< structured statement > = < parallel block >
G12	< conditional > = if < expression > then < block > else < block >
G13	< parallel block > = cobegin < parallel statement list > coend
G14	< parallel statement list > = < block >
G15	< parallel statement list > = < block > < parallel statement list >
G16	< access control statement > = request(< id list >)
G17	< access control statement > = release(< id list >)

In the following paragraphs, Y,Y',Y", and Y'" specify the domain-range pair associated with the syntactix unit that immediately precedes their use.

Functions which Analyze a Procedure for Noninterference

The returned result is true or false.

```
A1( < procedure >Y)  =  A2(< block > Y)                      if rule G1
A2( < block > Y)  =  A3(< statement list >  Y)               if G2
A3( < statement list > Y)
          ┌ A4( < statement > Y)                             if G3
     =   {
          └ A4(<statement>Y') ∧ A3(< statement list > Y")    if G4
```

```
A4(< statement > Y) = {true                                   if G5 or G6 or G8
                      {A5(< structured statement > Y)         if G7
                                    {A2(< block > Y)          if G9
A5(< structured statement > Y)  =  {A6(< conditional > Y)    if G10
                                    {A7(< parallel block >Y ) if G11
A6(< conditional > Y)
          ≡ A6(if< expression > then < block > 1 else< block > 2 Y)
          = A2(< block >1Y") ∧ A2(< block > 2 1Y"')
A7(< parallel block > Y)
          ≡ A7(cobegin < parallel statement list > coend Y)
          = A8(< parallel statement list > Y)
A8(< parallel statement list > Y)
          = {A2(< block > Y)                                  if G14
            {A9(< block > Y' < parallel statement list > Y")
                                                              if G15
A9(< block > Y'  < parallel statement list > Y")
            {A2(< block > Y')A8(< parallel statement list > Y")
          = {                                                 if Y'∿Y"=∅
            {false                                            if Y'∿Y"≠∅
```

Transformations for Placing the Initial Requests for the Set X of Path Names.

The result is a procedure with request(s) for X.

```
B1( < procedure >Y,X)
     = < procedure heading > B2( < block > Y,X)            if rule G1
B2( < block > Y,X) = < block > Y                            if X ∩ Y=∅
B2( < block > Y,X)
     = begin B3( < statement list > Y,X) end                if X∩Y≠∅ and G2
B3( < statement list > Y,X)
       {B4( < statement > Y,X)                              if G3
     = {B4( < statement > Y',X) < statement list > Y"       if X∩Y'≠∅ and G4
       { < statement > Y' B3( < statement list > Y",X)      if X∩Y'=∅ and G4
B4( < statement > Y,X)
       {request(X) < assignment > Y                         if G5
     = {request(X) < procedure statement > Y                if G6
       {B5( < structured statement > Y,X)                   if G7
B5( < structured statement > Y,X)
       {B2( <block > Y,X)                                   if G9
     = {B6( <conditional > Y,X)                             if G10
       {request(X) <parallel block > Y                      if G11
B6( < conditional > Y,X)
     ≡ B6(if <expression > Y' then <block >1 Y" else < block >2Y"'X)
       {request(X) if <expression > Y' then <block > 1Y"
       {           else < block >2Y"')                      if X ∩Y'≠∅
    =  {if <expression > Y' then B7( <block >1 Y",X)
       {else B7(< block > 2Y"',X)                           if X ∩Y'=∅
B7( <block > Y,X)
     = {B2( <block > Y,X)                                   if X ∩Y≠∅
       {begin <statement list> request(X) end               if X ∩Y=∅
```

Transformations for Placing the Release(s) of a Path Name X.

The result is a procedure with release(s) for X.

```
C1(< procedure >Y,X)
     = < procedure heading > C2(< block > Y,X)                  if G1
C2(< block > Y,X)
     = { < block > Y                                            if X ∉ Y
       { begin C3(< statement list > Y,X) end                   if G2 and X ε Y
C3(< statement list > Y,X)
       { C4(< statement > Y,X)                                  if G3
     = { C4(< statement > Y',X) < statement list > Y"           if X ∉ Y" and G4
       { < statement > Y' C3(< statement list > Y",X)           if X ε Y" and G4
C4(< statement > Y,X)
       { < assignment > Y release(X)                            if G5
     = { < procedure statement > Y release(X)                   if G6
       { C5(< structured statement > Y,X)                       if G7
C5(< structured statement > Y,X)
       { C2(< block > Y,X)                                      if G9
     = { C6(< conditional> Y,X)                                 if G10
       { < parallel block > Y release(X)                        if G11
C6(< conditional > Y,X)
     ≡ C6(if< expression > then < block > else < block > Y,X)
     = if < expression > Y'
       then C7(< block > Y',X) else C7(< block > Y"',X)
C7(< block > Y,X)
     = { C2(< block > Y,X)                                      if G9 and X ε Y
       { begin release(X) < statement list > Y end              if G9 and X ∉ Y
```

Functions for Determining Refinements of X not Accessed in Critical Region H'(X).

The returned result is a subset of the total refinement F(X) of X.

```
D1(< procedure > Y,X,F(X))
     = D2(< block > Y,X,F(X))
D2(< block > Y,X,F(X))
     = { F(X)∩Y                                                 if X∉Y
       { D3(< statement list > Y,X,F(X))                        if XεY and G2
D3(< statement list > Y,X,F(X))
     = { D4(< statement > Y,X,F(X))                             if G3
       { D5(< statement > Y' < statement list > Y",X,F(X))      if G4

D4(< statement > Y,X,F(X))
     = { ∅                                                      if G5 or G6 or G8
       { D6(< structured statement > Y,X,F(X))                  if G7
D5(< statement > Y' < statement list > Y",X,F(X))
     = { (F(X)∩ Y") ∪ D4(< statement > Y',X,F(X))               if X∉Y"
       { D3(< statement list > Y", X,F(X))                      if XεY"
```

```
D6(< structured statement > Y,X,F(X))
      ⎧D2(< block > Y,X,F(X))                              if G9
    = ⎨D7(< conditional > Y,X,F(X))                        if G10
      ⎩∅                                                   if G11
D7(< conditional > Y,X,F(X))
    ≡ D7(if < expression > Y' then < block >_1 Y''
                              else < block >_2 Y''',X,F(X))
    = D8(< block >_1 Y'',X,F(X)) ∪ D8(< block >_2 Y''',X,F(X))
D8(< block > Y,X,F(X))
      ⎧D2(< block > Y,X,F(X))                              if X∈Y
    = ⎩F(X) ∩ Y                                            if X∉Y
```

Transformations for Placing Request(s) for the Refinement X' of X.

The result is a procedure for request(s) X'.

```
E1(< procedure >Y,X',X)
    = < procedure heading > E2(< block > Y,X',X)            if G1
E2(< block > Y,X',X)
      ⎧< block >                                            if X ∉ Y
    = ⎩begin E3(< statement list > Y,X',X ) end             if X ∈ Y and G2
E3(< statement list > Y,X',X)
      ⎧E4(< statement > Y,X',X)                             if G3
      ⎪E4(< statement > Y',X',X) < statement list > Y''     if (X'∉ Y'' or X ∉ Y'')
    = ⎨                                                        and G4
      ⎪< statement > Y' E3(< statement list > Y'',X',X)     if X' ∈ Y'' and X ∈ Y''
      ⎩                                                        and G4
E4(< statement > Y,X',X)
      ⎧request(X') < assignment > Y                         if G5
    = ⎨request(X') < procedure statement > Y                if G6
      ⎪request(X') < access control statement > Y           if G7
      ⎩E5(< structured statement > Y,X',X)                  if G8
E5(< structured statement > Y,X',X)
      ⎧E2(< block > Y,X',X)                                 if G9
    = ⎨E6(< conditional > Y,X',X)                           if G10
      ⎩request(X') < parallel block > Y                     if G11
E6(< conditional > Y,X',X)
    ≡ E6(if < expression > Y' then < block > Y'' else < block > Y''')
      ⎧request(X') if < expression > Y '
    = ⎨             then <block > Y'' else <block > Y'''    if X ∉ Y'' ∩ Y'''
      ⎪if < expression > Y'then E2(< block > Y'',X',X)
      ⎩                   else E2(< block > Y''',X',X)      if X ∈ Y'' ∩ Y'''
```

Techniques for requirements-oriented design

by KENNETH J. THURBER
Sperry Univac Defense Systems Division
St. Paul, Minnesota

and

University of Minnesota
Minneapolis, Minnesota

ABSTRACT

The purpose of this paper is to discuss how requirement studies can be performed for computer systems. There are many different types of requirement studies that can be used to define a system and a number of these are described.

INTRODUCTION

The architectural process and its relationship to requirement studies

This paper is about the determination of requirements for the architecture of computing systems. Webster defines architecture as follows:

> architecture: the art or science of building;
> specif: the art or practice of designing and building structures.

As we proceed. we will find the connotation of architecture as art to be particularly appropriate in the context of computing systems and their requirements. While there certainly is a large and rapidly growing body of knowledge concerning the engineering aspects of computing system design, there has been little real emphasis on techniques for determining the requirements to be used as the basis for the design of computing systems.

In the current context, architecture includes additional activities to those delineated in the Webster definition. In particular, the design activity must include requirements analysis and specification as an integral part. In fact, in many computing system applications, initial marketing activity involves creation of customer demand as a first and often overlooked step of the complete architectural process. Of course no one of the integral activities of architecture stands alone. The requirements analysis is closely tied to the design which is iteratively associated with logic design, and software design and implementation. All these activities are associated with evaluation which must take place within each activity and between activities. Consider, for example, the restricted architectural problem of exactly duplicating a competitor's computing system. Presumably a marketing analysis entered into the decision to duplicate and the requirements specification is largely provided by the complete specification of the extant system. Other requirements include projected costs and additional features. The design activity here could concentrate on efficient implementation, and some evaluation activity could be required to determine how accurately implementation costs matched those specified as the target during the requirements analysis. Implementation of the copy system (which could proceed from the design and detailed evaluation) could be required to determine that the implemented system met the specified requirements (both technical and financial). Typical architectural activities include: requirements synthesis and analysis, requirements specification, configuration and subsystem design, detailed design, implementation, and evaluation. Trade-offs cross hardware, software, and firmware boundaries at each level and the design process will proceed iteratively. Some decisions that are made are simply a matter of style. If the designer does not have a total feel for the complete architectural problem there may be no way to show that a given proposed feature is worth the cost of implementation. Based upon the designer's experience the resultant system style will be developed.

Systems

In a paper about systems it is important to discuss what constitutes a system and how the view of a system can be different depending upon whether we are the user or designer of the system. There are no real, precise definitions of what constitutes a system. Companies that sell memories speak of memory systems; mainframe manufacturers speak of computer systems; software manufacturers speak of software systems; etc. Notably, the word "system" seems to mean the end product produced by the manufacturer.

We will define a system to be a hierarchical, dynamic collection of hardware and software entities. A system is composed of an application-defined environment together with a set of software and hardware that hosts the application. The application environment delimits and specifies the system.

The software and hardware which supports the system has hierarchical, dynamic relationships, which together form the basis to support the application. The hardware and software are hierarchical because of the identifiable levels; i.e., microcode, register level, CPU level, etc. They are dynamic in that in processing the application, the various levels interact to support the application.

There are many different views that can be taken of a system. We will briefly discuss several major views of a system. First, there is the view we have of a multi-end user system. An example of such a system is a computer utility. In this case, we see general-purpose hardware and general-purpose software. Another system type is the single-end user system. Examples of such systems are integrated command and control systems, and single-owner computers centers. Such systems have general-purpose hardware and specific software tailored to the system's function. A third view of a system is that of a single-owner system. In such systems, we see specific hardware and software. The last view of a system is that of the designer. In the past, since designers were primarily hardware designers only, the designer saw specific hardware and either specific, general, or no software. The usual case was no software. In the future, it is important that computer architects not be solely hardware designers, but computing system architects capable of making realistic, relevant and user-responsive global system design trade-offs. Only then will systems be more responsive to the user and customer.

SYSTEM REQUIREMENTS

The system design process

In designing a system, we must satisfy a need. This need must reflect the environment as well as the objectives for the system. For example, a system designed for a deep space environment would include consideration of very stringent environmental and reliability constraints on the entire system. The design process starts with determination of requirements; that is, the user's needs. From these needs, the design is specified. It is the extension of this need that makes a market for a product. The user's needs may be broken into two categories. These categories are discussed further under the headings of Requirements and Attributes.

Requirements* are the constraints which the system must satisfy. The requirements specify what the system *must* do. That is, any system concept which meets the requirement is a candidate solution to the customer's problem.

Attributes,** on the other hand, specify either options or evaluation criteria for qualitative comparisons of competing systems that meet the system requirements. There may be many concepts which satisfy the architectural requirements. Attributes may be used to evaluate the competing architectures to obtain a feel for the "goodness" of the architecture in solving the customer's problem or as a set of factors used to optimize system designs. Attributes may also specify options which the user desires but does not necessarily demand.

The problem statement

In designing a system, the customer's needs must first be determined. This involves a problem statement. Usually we will not be given a specific set of requirements. Rather, we will be given a statement of the user's problem in general terms. This problem statement describes the user's need. It is from this basic need and the concurrent design constraints based upon industrial affiliation that the specified system will be derived.

Requirements

Requirements are constraints placed upon a system concept. Requirements are the "musts" that any candidate system shall satisfy in order to be a viable, potential solution to the user's problem. However, depending on the level of definition of the requirements, there may be many systems which meet the requirements but are not acceptable. For example, if the sole requirement were to meet a certain throughput rate, there would be many architectures which could meet the required throughput. However, due to connectivity of processing elements, basic structure, or machine repertoire, a large number of proposed concepts may not actually be applicable. It is extremely important that the requirements be very carefully defined.

The source of requirements is a problem definition. It is from the translation of the customer's functional problem definition that the requirements are derived. There are typically two types of requirement studies that are performed. These are requirements analysis and requirements synthesis. Requirements analysis typically is involved with special purpose systems or market analysis after the fact. The requirements of a system or an application are typically analyzed in terms of what is known about the problem a priori from previous solutions to describe the problem environment and obtain a feel for how new technology could better be applied to the problem. Requirement synthesis, on the other hand, is involved with trying to project what types of systems will be useful in new application environments; i.e., to synthesize or create problems that

* Bell[1] calls requirements "wants."

** Bell[1] calls attributes "objectives" and describes their evaluation as a relative maximization or minimization process.

could be solved with new technologies, or define areas in which new technologies will allow a more cost-effective solution to the created problems. As an example of the requirements analysis, the AMNCS (Advanced Multiplatform Naval Computer Study)[2] is a classic example of a functional requirements analysis of current Navy problems. The AMNCS analysis tried to determine what Navy tactical requirements were established in the past and how new technology may be applied to those functions. The Hewlett-Packard hand-held calculators. are a classic example of requirement synthesis. In the Hewlett-Packard example the designers asked, "If we were able to build such a device, would a market develop?" Obviously, the market did develop since large numbers of people now own pocket calculators.

There are numerous varieties of requirements. Some of these may be generally categorized as marketing requirements, economic requirements, technical requirements, and political requirements.

Examples of marketing requirements may be that the system being designed must be capable of solving a certain list of designated problems. Other marketing requirements may include price goals, goals specifying the production cycle, product life goals, and logistics goals.

Typical economic requirements deal with the financial constraints imposed on the system developer. Examples of economic requirements may be that the non-recurring development costs do not exceed a certain dollar figure; that the spares cost for repair does not exceed a certain figure; and that the manufacturer has been in business a specified number of years.

Political requirements are viewed mainly by the designer of systems that require the use of off-the-shelf equipment and who may be trying to use piece parts that are available through his own company's manufacture versus piece parts that are available from other vendors. In this case, a typical requirement may be that we use our own microprocessor if we're a semiconductor vendor.

At best, the political, economic, and marketing requirements may be extremely vague. In a real sense, however, technical requirements may be made very precise. The types of items that may be considered for technical requirements include system organization, word/byte/bit organization, hardware and software expansion capability, data path widths, word size, memory hierarchy and memory sizes, availability of software, availability and capability of the operating system, assembly and compilation speeds, throughput speeds on specified benchmarks, type and availability of peripherals, interrupt structure and support features, instruction repertoires, and utility packages. Further technical requirements may include alternative throughput capabilities, I/O capabilities, percentage utilization of resources, and cost performance ratio. One real requirement that is typically not considered but is very important is the type of performance evaluation and monitoring features available. This equipment must be available to allow the user to judge how well the system not only initially meets his procurement requirement, but continues to meet his job requirements in the future.

Attributes

Since system requirements cannot be totally comprehensive and precise unless they spell out a unique system architecture concept, attributes are introduced. Generally, it is not in the customer's best interest to define a precise system for if he does, he runs the risk of purchasing a system which is not cost-effective. If you buy a system in a non-competitive environment, you run the risk of paying more for that system than if there are two systems which can satisfy the requirements and, consequently, two manufacturers bid on the requirements. It is therefore to the user's advantage to make the requirements as definitive as possible, but allow for the introduction of various manufacturers' equipment to obtain the best price possible.

There are times when the architecture of the system will not be totally precise and there are features which are not specifically required, but are desired depending on their cost. Attributes are the wants (options) and evaluation criteria used to determine which characteristics make one system more desirable than another system even though they may both meet the requirements. Attributes usually deal only with the detailed technical aspects of a system. Attributes in the form of desired options deal with specific system features. For example, as an option to a processor which requires an interrupt structure, you may ask that the structure not only include enable/disable by class, but that the interrupt structure is desired to have, but not require, the ability to arm, disarm, enable and disable interrupts by level, and that a number of levels (such as 32) should be furnished. However, the requirement for such features may have actually been stated in the following manner: That the computer at least furnish four classes of interrupt with enable/disable characteristics. There are many more machines which satisfy the requirement than would satisfy the attribute. Depending on how systems rank on the option list and on other general attributes, the final selection of the design will proceed. Attributes are the intangibles of design.

The following list of criteria can be used to evaluate candidate architectures. These, then, are the alternatives that must be ordered to provide a list of the attributes and their relative importance in system design. These ranked attributes may then be used to trade off alternative machine architectures: flexibility, expandability, bus complexity, executive complexity, availability, adaptability, partitioning, modularity, reliability, maintainability, manufacturability, production cost, development cost, technical risk, logistics, programmability, support software cost, software adaptability and transferability, compatibility, and service. Having the ability to precisely quantify these attributes so that they may be measured against each other is difficult but necessary to insure a good system selection.

Options which are detailed technical desires may also be rank ordered and included in the attribute evaluation analysis. Typical options may be inclusion of a maintenance processor, ability to upgrade to a virtual memory system, or disk operating system availability. Obviously, detailed technical options are easier to measure than general attributes, but they are also more restrictive measures of desired but not required capabilities.

Requirements-oriented design

There are a number of steps involved in designing a system to a set of requirements. However, there are a specific set of steps involving the use of the requirements and attributes. These steps will be summarized and discussed herein. The actual design step will be simply described as "system design" such that at this point we will be able to see how the requirements and attributes actually lead to the detailed system design step.

The first step in requirements-oriented design is the problem analysis step. This step consists of determining, from the customer, what are the user's functional requirements. This may involve studying the applications as they are currently implemented, trying to project what the application is, or will be in the future, or trying to determine, based on technology issues, what needs could be generated in the marketplace for certain types of products. After the problem analysis step, we will have a detailed statement of the exact problem and its functional characteristics.

The second step is the determination of the requirements and attributes. This step involves taking the problem analysis functional description and translating it into a detailed set of requirements and a detailed set of attributes; the requirements specifying what the product system must do, the attributes specifying what detailed technical options are desired for the system, and a set of attribute priorities ranked for use as architectural trade-off parameters. In the case that more than one system satisfied the requirements, then, the attributes will be used in optimizing system selection. The attributes should be ranked in priority order at this point to insure objective trade-off and comparison of competing architectures.

The third design step is the determination and description of the requirements and attributes in a specification. Two documents should be generated, one which describes the detailed translation of the problem statement into the requirements and how the requirements were derived from the problem statement. This document then ends with a detailed specification of all machine requirements. Analogously, an attributes specification is generated. The attributes document should rank general attributes for trade-off usage and enumerate all detailed technical options.

The system design process (step) consists of designing delimited architectural choices. It is very difficult to translate a set of requirements onto a set of architectures, or to use a set of requirements to select a set of architectures. To achieve an accurate evaluation of the concepts under consideration they all may have to be designed to a level of detail that enables comparison to the requirements specification. Therefore, the design process really consists of using the requirements document and attributes document to narrow the scope of allowable choices and to narrow the number of concepts that satisfy the requirements. After all systems have been screened and those that do not satisfy the requirements eliminated, architectural descriptions are generated and furnished to the attribute evaluation step for all remaining architectures.

Using the attribute ranking, an attribute evaluation step is further used to list the architectures in descending order of their satisfaction of the attributes. Then, a number of systems which all satisfy the requirements and which are ranked in order of priority of satisfaction of the attributes may be selected and bids solicited from manufacturers. Alternatively, if we are designing a system from scratch, a composite concept which satisfies all the requirements and is one of the systems which rank high on the attributes should be selected for detail design. This process is probably iterative and best performed in a trial and error fashion. The main difficulty is that attributes cannot be precisely measured and thus the "optimal" design is always illusionary.

Requirements and attributes make the design process manageable. They cut down the number of choices we must make by clearly delimiting our choices. Obviously, the process must be iterative. Changes in technology, cost considerations, etc., may cause changes in the requirements or attributes specifications. Further, the use of requirements and attributes splits the design goals clearly into "musts" and "options". The effect of the process is to continually narrow the choice so that the designers can quickly focus on the problem. Thus, the requirement and attribute, and problem analysis continues to refine the general spectrum of architectures down to a single architecture best suited for the application. This is the architecture which is then designed.

Goals, policies, and product specifications

To help with project management, after the requirements and attributes are determined, a set of system goals which describe objectives and policies which point the direction for the achievement of the goals (how the goals can be achieved) can be determined. This information is useful from the management viewpoint, but does not get into the detailed system requirements as described in the product specification document which is a result of either system synthesis or system selection.

TYPES OF REQUIREMENT STUDIES—ANALYSIS VERSUS SYNTHESIS

Requirements seem to emanate from two important contexts: (1) Analysis of existing system concepts or (2) synthesis of new system concepts. The basic premises behind analysis efforts are the ideas of either (1) building a better copy of a competitor's product, (2) upgrading a current product, or (3) integrating the best of many concepts into one new design. In these cases, extensive analysis of both the market and technology can be performed with the resulting analysis used to drive the design effort.

Analysis is also quite useful for the selection of a system (end-user perspective).

On the other hand, requirements synthesis involves the conception of a new product or new market. This environment may be intuitive and thus the issues may not be as clear as in applications in which only analysis is performed. In some circumstances, synthesis may only involve the projection of current analysis results to account for technology improvements or enhancements.

Requirements can fall along a spectrum. The design of a system from a requirement thus also can fall on a spectrum. At one end of the spectrum we have market-only requirements. In this case, the issue is to build the cheapest possible product. At the other end of the spectrum is the technology-driven requirements. In this case, the problem is to build the highest performing product given a particular technology. In between there are many variations of the two extremes, most of which can be generically categorized as an attempt to design a product which optimizes the cost/performance ratio. Figure 1 summarizes the spectrum of possible requirement studies.

USE OF ANALYSIS

As previously indicated, analysis can be projected and used either by designers of products or by buyers of products. In this fashion, analysis can become a form of product concept synthesis.

IMPORTANT EXAMPLES

Requirement studies for a number of important systems or application groups have been documented in the literature and are briefly listed below. Some of these studies are important due to their actual resulting product; whereas, others are important not because they resulted in products, but because they illustrate specific design techniques. The studies listed below are illustrative of the types of efforts that have been performed and documented. This list of references does not attempt to be a complete bibliography.

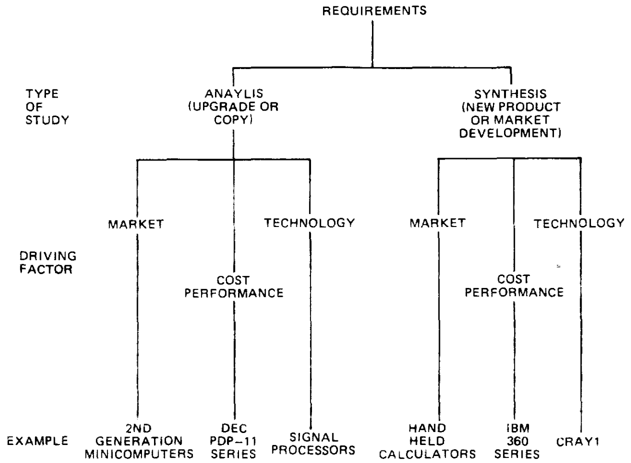


Figure 1—Requirement study genealogy

Bell

C. Gorden Bell[1] provides some interesting insight into the design process in this book.

IBM Series 360

Amdahl et al.[3,4] in these papers presents the design strategy and requirements behind the 360 as well as insight as to how the system was impacted by the requirements.

DEC PDP-11

Bell[5] provides the PDP-11 requirements and their design impact in this paper.

Military systems

The AMNCS (Advanced Multiplatform Naval Computer System) study[2] provides significant insight into the requirements analysis of a large user group. In a related study,[6] Punj provides a survey of the requirements and capabilities of a large number of Naval tactical operating systems. Further, related work includes: the E-2B requirements study[7] of a specific aircraft, the MCF (Military Computer Family) design goals,[8] and analysis of specific Army system requirements for a large number of systems.[9] The Air Force has also performed significant requirements work and technology projections, as noted in References 10, 11, and 12. Further, interesting military studies include References 13 and 14.

Distributed processing

Kilpatrick[15] presents a unique approach to the analysis of specific application for use with a distributed processor.[16]

Software systems

A detailed analysis of many operating systems can be found in Reference 17. Analysis to support specific operating systems with hardware primitives is available in Reference 18.

DISCUSSION

In the following, four examples will be considered. Examples one and two will discuss general purpose computer systems. Example three will discuss the functional computation requirements for a large user application base. The last example will discuss the detailed I/O, computation and memory requirements for a specific system.

Comments on general-purpose computers

For general-purpose, data-processing systems, the functional requirements usually cannot be well defined. The competitive business nature has kept the analyses that have been done in a proprietary vein. Currently, the most important commercial requirement seems to be software compatibility.

Although few general-purpose computing system requirement studies have been discussed in the literature, there are two classic examples of such requirement studies—studies were conducted on the IBM 360 Series,[3] and the DEC PDP-11 Computer.[5] Both articles contain a general description of the design objectives, the major architectural decisions, and some of the reasons for those architectural decisions. Since the machine to be designed is general purpose, the data from the detailed requirement studies has been filtered and the results indicate a series of design goals for the new computer system.

IBM system 360 requirement study

There were four major innovations in the IBM System/360: (1) a flexible storage concept which provided variable capacity; a hierarchy of different speed memories; storage protection and program relocation, (2) an I/O system which provided concurrent operation; large amounts of channel capacity; an integrated design between the hardware and software and CPU interaction; and a standard channel interface, (3) a general-purpose machine organization with very powerful operating system; logical processing operations; and many different instruction and data formats, and (4) machine-level language compatibility over a series of models with a performance range of over 50.

In performing the architecture development of the 360 System, several important systems concepts and trends were noted by Amdahl, et al.[3]: (1) the adaption of business data processing to scientific data processing equipment, (2) the total system concept including I/O, (3) the use of program translators, (4) the development of large, secondary storage mechanisms such as tapes, drums and discs with many order-of-magnitude larger storage capabilities than seen in previous media, and (5) real-time and time-sharing system development.

Based on these general technical trends, a number of different concepts were provided for in the 360 System. The major requirements for the system could be grouped into five major areas: (1) provide for advanced system concepts, (2) provide an open-ended design, (3) ensure a general-purpose functional capability, (4) provide a cost-effective performance range, and (5) produce complete intermodule software compatibility. In each of these five requirements there was a number of subrequirements.

In the advanced concept area it was recognized that a major break would have to be made with existing products even though this would result in some software incompatibility. The break would establish the new family of machines. Therefore, the following subrequirements of the advanced concept requirement were considered: (1) that the computer provide for a family capability to provide growth, and to allow for a succession of product lines, (2) that a high-performance, general I/O technique be developed which would allow I/O devices tailored for applications to be used with any machine even though the I/O devices differed in rate, access times, or functionality (also, that the input/output channel and input/output control program had to be designed to be compatible with each other), (3) to utilize the throughput of a machine not to obtain high-speed processing, but to obtain high-speed problem solution by making a complex machine and programming system that are easy for the user to manipulate, (4) to increase CPU utilization for computing by providing for addition of compilation, I/O management, etc., (5) to provide a comprehensive operating system which includes extensive interrupt facilities, and good storage protection, (6) to provide a failsafe/failsoft capability in systems with more than one CPU, (7) to provide a large storage capability rather than the 32,000 words normally required and furnished at that time, (8) to provide for large word lengths to accommodate large fixed- and floating-point words, and (9) to provide detailed hardware maintenance and diagnostic aids to reduce system downtimes and make identification of individual malfunctions easier.

The open-ended design requirement was an attempt to ensure customers that when they made the break with the previous software concepts, they would have a long-term, viable computer system which would continue to use the same architecture but be upgraded for speed and performance over a long time. This then enabled IBM to satisfy their customers so that when they made the switch to the new machine, they would not immediately, in three or four years, be required to make another switch. In this area, a number of subrequirements were identified: (1) that the new design must provide customer programming capability for over a decade; thus, the machines would have to remain with the same architecture for at least a decade, (2) that the design permit asynchronous operation of major subsystems so that subsystems may be updated technologically without impacting the total system configuration, (3) that many decisions be made to ensure that the functions of the machine are general; that is, that spare bits, etc., be carefully placed in the words to ensure that new techniques or new functions that came along did not obsolete the new product line, (4) that hardware and software control be embodied in the machine such that it could directly sense control and respond to other equipment modules via techniques which are outside the "normal techniques." This would provide for the construction of "super systems" that could be dynamically managed from the basic system. It would also provide for the construction of special systems designed for specific applications and would allow for the construction of systems where some shortsightedness of the original design had been encountered.

In order to meet varying requirements such as those encountered in commercial, scientific, time-sharing, data reduction, communications, and other types of processing, the 360 CPU would have to be capable of hosting these different applications. Thus, different types of facilities may have to be offered as options, but must appear as integral features from the viewpoint of the system's logical structure. In particular, the general-purpose objective dictated: (1) that manipulation of words or bits be such that the operation depends upon the general representation rather than on any specific selection of bits, (2) that operations be code independent, i.e., all bit combinations are acceptable as data and no data can exert any control function on the machine, (3) that bits be addressable, (4) that the addressing structure be able to address directly the unit used for character representation, i.e., addressability to the byte.

In the performance area, the main consideration is that the various products in the product line have a consistent cost-performance ratio that decreases or remains stable as the system performance increases. However, due to the compatibility constraint, there is a large problem in this area.

The last 360 requirement was for intermodel compatibility. At least six models were anticipated with a performance range of 50. Intermodel-compatible really meant program-compatible. Program-compatible meant that any valid program whose logic did not depend implicitly upon time of execution, or other side effect programming which would run on Configuration A, would also run on Configuration B if B contained at least the required storage, I/O devices, and optional features. A hedge clause was placed in this description such that any invalid program which violated the programmer's manual was not constrained by the manufacturer to yield the same results and thus was not strictly program-compatible. Therefore, if the user adhered to the programmer's manual, since the architecture of all the machines was identical, he could run on any 360 structure regardless of the speed differences between models. However, the program can run at different rates.

The article by Amdahl, et al.[3] continues to describe how some of the decisions were made for the machine design which resulted in the 360, based on the previously summarized requirements. However, this paper is really only interested in the requirements and thus will not delve into the 360 architecture. But, we will make one point; that is, in doing a requirement study, regardless of the type, whether a general purpose study such as that for the IBM 360 or a detailed special-purpose requirement study, the design step must begin somewhere. It is difficult to break out of the mode of determining the requirements and starting the design. Therefore, it is irrelevant as to where we begin the design, except that we must break the requirements open from some position and say, "If we make this selection, how does this impact the other requirements?" In the 360, for example, this was accomplished by considering the basic addressing structure and first determining what the data format should be. The first decision was to go with an 8-bit byte. Once this decision is made, the design could proceed and the architect could lay out the formats, work on the field specifications, the instruction decisions, and the system architects could go to work synthesizing the various configurations. The architects will thus know how large the machine will tend to be; what kind of addressing modes are envisioned; how the memories have to be addressed, etc.; the point being, that we can get into a circuitous mode during the requirement study. A decision must be made! After making that first decision, we can then assess the impact upon the requirements and then change the decision, if necessary. But most importantly, we can begin the design process and cut down the amount of information we are required to deal with in general terms. The specifics we decide upon can be traded off against each other so that the system design may progress.

The DEC PDP-11 requirement study

The PDP-11 requirement study is slightly different than the IBM 360 requirement study. Whereas IBM decided to make a major break from their architectures, the PDP-11 was designed without regard as to whether or not it would be a major break. Rather, it was designed in order to solve certain technical problems that had been encountered by the customers of the DEC Corporation. Their customers were using four different machines at the point that the PDP-11 was conceived, a PDP-5, LINC, a PDP-4, and a PDP-8. Furthermore, these models were being used in communication control environments, instrumentation environments, preprocessors and communication processors for large systems, data acquisitions, etc. The PDP-11 was designed to overcome weaknesses that had been encountered in the current DEC mini-computers based on the customer's application experience. The weaknesses that the PDP-11 was to overcome include: (1) limited addressing space, (2) too few registers, (3) lack of hardware stack capability, (4) slow context switching among multiple processes, (5) lack of byte string manipulation capability, (6) lack of read-only memory storage facilities, (7) elementary I/O concepts, (8) lack of ability to upgrade users to a higher performance model, and (9) high programming costs due to lack of high-level languages and their associated software support.

The new machine family was to take advantage of new integrated circuit technologies that were becoming available, contain enough machine models to span a range of functions and performance, update the DEC product lines into what is considered classical, third-generation machines, work equally well in the addressing mode mechanizations 0, 1 or 2 address machine, and present the user with a very sophisticated connection system which later became known as the Unibus.

Notably, the PDP-11 requirements tended to be much simpler than the 360 requirements. There are a number of reasons for this including the size of machine, the size of the corporation and its market base, and the context in which the machines were used. However, one will note that the requirements which were used to define the PDP-11 are very distinct and direct to the points that describe what changes must be made to be successful in the minicomputer business.

Functional requirements analysis for a large user group

In this discussion we will first consider the problem definition taken from the Advanced Multiplatform Navy Computer Systems Project (AMNCS).[2] The problem statement or objective of the AMNCS Project was to provide information and guidance on requirements for the Advanced Multiplatform Naval Computer System. The specific project objectives were: (1) to identify a set of common functions for tactical data systems, (2) identify major, common subfunctions for the functions identified, and (3) identify computational functions for the subfunctions. Additional objectives of the AMNCS study dealt with computer architecture and technology projection, however, they are not pertinent to the objectives of this paper.

ASIS	TIPI	DTAS	TWAES	MTACCS	JIFDATS	MTDS	LDMX	TFCC	HAROY	PROJ 749	FCSC	STICS/AID	ANEW	IOIS/IOIC	AGEIS	NIPS	OSIS	ASWCCCS	ASWSCCS	MPDS	IFDS/FFDS	ATDS	NTDS		
				X		X				X	X	X	X		X		X		X			X	X		Tracking
				X	X	X		X	X	X	X	X	X	X	X				X			X	X		Sensor Data Process.
	X									X	X	X			X		X	X	X			X	X		Multisensor Corr.
					X					X			X		X							X	X		Sensor Equ. Control
									X	X			X		X				X			X	X		Navigation
													X		X							X			Guidance
				X															X			X	X		Air Traffic Control
				X		X									X							X	X		Air Intercept Control
				X						X			X										X		Terminal Control
				X		X				X					X							X	X		AAW
				X		X				X	X				X							X	X		Strike Operations
										X			X									X			AEW
	X	X		X	X	X	X	X	X	X	X		X	X	X	X	X	X	X			X	X		Data Link Comm.
	X	X	X		X		X	X	X	X	X					X	X	X		X	X				Message Process.
	X	X	X				X	X	X	X	X						X	X		X	X				Message Dist.
	X	X	X	X			X	X	X	X			X		X		X	X	X	X		X	X		Multicomm Control
	X	X		X	X	X	X	X	X	X	X		X		X		X	X		X		X	X		Secure Comm.
	X	X		X	X		X	X	X	X	X		X	X	X	X	X	X		X	X				Secure Processing
			X	X		X					X		X		X				X			X	X		Weapons Assignment
	X		X	X	X						X	X		X	X		X	X				X	X		Threat Evaluation
			X										X		X										Weapons/Sys/Veh. Sim
	X								X		X	X	X	X	X							X	X		EW
									X				X		X							X	X		ECM/ECCM
	X				X			X		X		X		X											ELINT
											X	X	X				X	X	X				X		ASW
									X				X									X			Auto Preflight Sys.
X	X	X	X	X			X	X				X	X	X		X	X	X		X	X		X		ISR
X	X	X	X	X		X	X	X		X	X	X	X	X	X	X	X	X	X	X	X	X	X		Console/CRT Display
				X		X		X			X		X		X		X	X			X	X	X		Geo Display
	X	X	X	X		X	X	X			X	X	X		X		X	X	X			X	X		Monitoring
	X	X	X	X		X	X	X		X	X		X	X	X		X	X	X	X		X	X		Alert Generation
	X	X	X	X		X	X	X		X	X		X	X	X		X	X	X	X	X	X	X		Hardware Fault Det.
	X									X	X		X		X		X								Software Fault Det.
	X		X	X		X		X		X	X		X		X		X	X					X		Degraded Operations
	X	X	X	X		X	X	X		X			X	X	X		X	X	X	X		X	X		Recovery
	X	X		X			X	X		X			X	X	X		X	X	X			X	X		Diagnostics
	X	X					X		X		X		X		X							X			Built-in Test
X											X		X					X			X				Logistics
X		X		X		X							X		X			X	X		X	X	X		Resource Management
X			X	X	X																				Amphibious Warfare
X	X	X	X	X			X	X		X	X	X	X	X		X	X	X		X	X				Data Base Management

Figure 2—System vs. major functions

In identifying a set of common functions, two specific groups of functions were identified. These were functions which all tactical data systems tended to perform, and specific mission capabilities required of tactical data systems. Typical functions performed by all tactical data systems are: (1) data collection, (2) data measurement, (3) data processing, (4) data correlation, (5) data display, and (6) system executive control. Typical mission application functions that are contained in most tactical data systems are: (1) track management, (2) air interception computations, (3) air traffic control computations, (4) strike control computations, (5) electronic warfare, and (6) weapons allocation and fire control. Therefore, when analyzing a particular tactical data system, we may examine or specify a particular system in terms of its general common functions which exist in all tactical data systems and its specific mission-oriented functions. The main difference, then, between tactical data systems in this limited kind of environment will be in the functions included in the concept, rates of computation of the various kinds of functions, complexity associated with the computations; namely, decision requirements, number of variable requirements, etc., along with the distribution of the functions in the computing system. The system concepts of the tactical data system can cause these changes in requirements on a specific function basis due to factors such as degree of computation accuracy, amount of input data, type of computers to be employed in the system, volume and type of communications, physical dynamics of the physical systems, and the types of display information that must be generated. Looking at the list of common and mission type functions, Shen next generated a list of major functions for tactical data systems. Shen provides a list of representative United States Navy tactical data systems and a list of major functions that appear in these tactical data systems. Figure 2 indicates which major functions appear for each tactical data system. This figure is representative of the type of tabulation performed in a requirements analysis study. From Figure 2 we can also determine all the functions that must be computed for any given system. However, each of the systems have potentially different throughput requirements, input data rates, etc. Based on these data rates and the break out in terms of major functions, we are able to obtain an estimate of the computational complexity of any given system. That is, with rate information we now have the global description of any given system in terms of its system functions. Using the system functions, we can continue to refine each of the functions until we are able to determine the exact computation rates to be used in the design of the computer system. Therefore, the next step would be to take each of the functions, such as tracking, and break it down in terms of subfunctions. In doing this for the AMNCS Study, Shen broke the major functions down into a set of 72 specific computational requirements and tabulated these 72 computational requirements against the 27 major functions. Typical computational modules (requirements) used by Shen included: (1) matrix operations, (2) sensor calibration, (3) triangulation and trilateration, (4) R,θ xy conversion—range and bearing to rectangular conversion (and inverse), (5) data encrypting, (6) track correlation, (7) vector operations, and (8) trigonometric functions.

With the list of subfunctions, Shen was able to generate a chart that compared the subfunctions to the major functions. This allowed a more detailed definition of typical rates for the functions, interaction between functions, parameters that must be passed between each other, difficult computations, etc. This level of detail provides a basis for the type of computation rates that must be performed in developing a particular architecture. At this point, we have not received any information, other than rate information, that would tend to give us a feel for the type of detailed computations to be performed. The subfunctions, we have determined, tend to be very small, such as coordinate conversion type functions. At this level of complexity we are able to break down these functions even further in terms of the system model. Having the subfunctions, we are able to break out and construct a model of any given particular system. We could start in a tactical data system and construct a figure which consisted of, on the left side, a set of all the sensors available, a set of all the functions included in the system, and a set of the outputs in the system. Using such diagrams, we can determine the parameters to be passed between subsystems. Each function or subfunction may now be described in terms of its given inputs into the function, computations to be performed, outputs to be returned, and special comments. For example, coordinate conversion, as a function, is given a range and a bearing. It computes the x,y coordinates based on the equations $x=r\mathrm{COS}\theta$, $y=r\mathrm{SIN}\theta$ and returns as outputs x and y. We can now develop a good description of all functions that must be actually computed in the system. A phase diagram indicating all the parameters and I/O data that has to be passed between functions of a particular system and the timing of all computations could be generated for estimation purposes.

A specific application requirement analysis

Kilpatrick[15] considered detailed requirements for a number of limited systems. Most of the systems considered would compare to one of the smallest systems considered by Shen. In particular, we will consider Kilpatrick's air-to-ground attack system concept. A summary of this concept is shown in Figure 3. It consists of electro/optical functions for target acquisition, threat warning function, fire control function, three variations of navigation functions, an air data function, flight control function, digital data link function, and display function. These functions are all sensor processing functions that connect to a system via some types of I/O links. In this case, the system concept depicted in Figure 3 was a distributive processor memory system and is therefore listed as DPM (Distributed Processor/Memory) Processor. DPM does the appropriate computations on the data and performs the appropriate functions associated with the functional concepts. A detailed system description is then generated. It breaks out not only the detailed subfunctions, but additionally all the signals between the I/O system and the computer system.

Let us review each of the primary subfunctions provided by the system. The primary functions are:

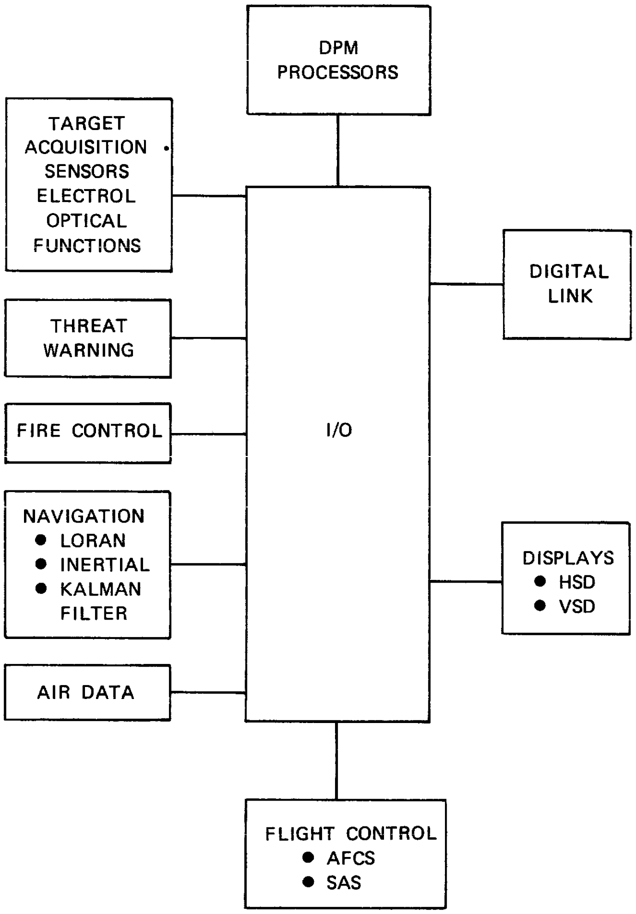


Figure 3—Air-to-ground attack concept block diagram

- Target Acquisition and Electro/Optic—a means for target recognition.
- EW/ECM—the Electronic Warfare, Electronic Counter Measures function. It is used for threat warning location and neutralization of enemy vehicles.
- FLR—a Forward Looking Radar function that is used for target acquisition, the navigation function, and weapon delivery.
- LORAN/Inertial—a part of the navigation function.
- Flight Control—the Stability Augmentation Function (SAF) and the Attitude Flight Control System (AFCS) function.
- Digital Link—provides two-way communication between the aircraft and the ground.
- Vertical Situation Display—provides a pilot with flight direction, sensor imagery, and weapon delivery information.
- Horizontal Situation Display—provides navigation information, threat and target location along with sensory image functions to the pilot.
- Control Unit/Data Entry—provides for subsystem mode of operation and data insertion into the computer from the pilot.

From the subfunction definitions and detailed equations, and the block diagram which shows the interaction of all the functions, the requirements analysis is begun. Kilpat-

FUNCTION	MEMORY CYCLES (THOUSANDS/SECOND)
LORAN	150
INERTIAL (SD/PLAT)	400/50
AIR DATA	30
FLIGHT CONTROL/CHANNEL	350
FIRE CONTROL	350
KALMAN FILTER	100
DISPLAY	200
TARGET ACQUISITION*	300
TOTAL:	1600/1250

*TARGET ACQUISITION AND FIRE-CONTROL FUNCTIONS ARE ASSUMED NOT TO BE PERFORMED AT THE SAME TIME.

Figure 4—Processor speed requirements summary

rick first generated the I/O data transfer requirements of which there are some important points to be noted. The system block diagram provided a listing of all primary information that must be transferred within the machine and processes within the system. Each of the subsystems has a number of other associated functions. These are the preflight type of functions of on/off power test and various kinds of status signals that do not impact the actual operational system.

With assumptions and derived information, the total I/O now can be specified for the system. Furthermore, taking a more detailed look at each of the functions associated with each of the subsystems, we can determine the data parameters that must be passed between each of the functions. These data parameters are tabulated as input data requirements versus computational functions.

With the I/O information and now the information detailing the interaction of data between the various functions, the requirements definition process can proceed by taking each of the separate functions and examining their algorithms. Thus, a set of total processing-time, memory and I/O requirements, by function, can be determined for each particular function.

Figure 4 lists the functions and memory cycles in thousands per second for the complete system shown in Figure 3. Since Kilpatrick made the assumption of a single-address machine with two memory cycles required for the execution of simple instructions such as Load or Add, a total computer memory cycle or instruction rate can be determined. The memory requirements for this concept are given in Figure 5 and the I/O requirements are summarized in Figure 6. With the I/O requirements, the memory requirements, and the system cycle speeds established, along with the detailed block diagrams and the detailed functional block diagrams, the requirements analysis is complete for this system. We could now proceed with the synthesis of the actual computing system.

FUNCTION	MEMORY WORDS	
	INSTRUCTION	DATA
LORAN	3900	500
INERTIAL	2200	350
AIR DATA	1300	70
FLIGHT CONTROL	3250	400
KALMAN FILTER	800	2000
FIRE CONTROL	4800	500
DISPLAY	2500	500
TARGET ACQUISITION	1500	2500
TOTAL SYSTEM	20,250	6800

Figure 5—Memory requirements summary

SYSTEM	DATA WORDS/SECOND	
	INPUTS	OUTPUTS
LORAN	50	75
INERTIAL (SD/PLAT)	600/150	30
AIR DATA	50	----
FLIGHT CONTROL	600	200
FIRE CONTROL	----	----
KALMAN FILTER	----	----
RADAR ALTIMETER	20	----
RADAR	20	40
E/O	40	40
DISPLAY	----	1000/4000
LINK	10	10
TARGET ACQUISITION	1000	----
TOTAL	2400/2000	1400/4400

Figure 6—I/O requirements summary

SUMMARY

This paper has presented a description of the use and generation of computer system requirements and a description of the relationship of the design process to system requirements. Two types of criteria (musts-requirements and wants-attributes) were introduced. Four important example requirement studies were discussed and a brief bibliography of important requirement studies was given.

REFERENCES

1. Bell, C. G., *Designing Computers and Digital Systems using PDP-16 Register Transfer Modules,* Digital Press, 1972.
2. Shen, J. P., "Advanced Multiplatform Navy Computer Systems (AMNCS): Initial Architecture Study in Technology Organization Projection for 1980-1990," Naval Electronics Laboratory Center, 26 September 1972, NELC/TR1847.
3. Amdahl, G. M. et al., "Architecture of the IBM System/360," *IBM Journal of Research and Development,* April 1964.
4. IBM, "The Structure of System/360," *IBM Systems Journal,* No. 2 and No. 3, Vol. 3, 1964.
5. Bell, C. G., et al., "A New Architecture for Mini Computers—The DEC PDP-11," *AFIPS Conference Proceedings,* Vol. 36, 1970.

6. Punj, D., et al., "A Survey of Navy Tactical Computer Applications and Executives," October 1975, Contract N00039-75-C-0312.
7. Smith, W. R., "Simulation of AADC System Operation with an E-2B Program Workload," NRL Report 7259, April 1971.
8. Coleman, A., "Army/Navy Military Computer Family," *COMPCON 76 Fall,* September 1976.
9. System Development Corporation, "Embedded Computer System Data Processing Requirements for U.S. Army Weapon/Data Systems," March 1976, Contract DAAB07-76-C-0334.
10. Rand Corporation, "Information Processing/Data Automation Implications of Air Force Command and Control Requirements in the 1980's (CCIP-85)," 1973.
11. Brodnax, C. T., "A Conceptual Study for Digital Avionics Information System," 1974, Technical Report AFAL-TR-73-427.
12. Turn, R., "Computer Systems Technology Forecast," 1975, AD-A010 944/7ST.
13. General Dynamics, "Space Tug Avionics Definition Study," 1974, Contract No. NAS8-31010.
14. Honeywell, "Airborne Computer Study," 1968, AD787033/OSL.
15. Kilpatrick, P. S., et al., "All Semiconductor Distributed Processor/Memory Study, Volume 1: Avionics Processing Requirements," August 1973, Technical Report AFAL-TER-72-226.
16. Johnson, M. D. et al., "All Semiconductor Distributed Processor/Memory Study, Volume 2: Data Processing," August 1973, Technical Report AFAL-TER-72-226.
17. Abernathy, D. H. et al., "Survey of Design Goals for Operating Systems," *SIGOPS Newsletter,* April 1973 (Part I), July 1973 (Part II) and October 1973 (Part III).
18. Sockut, G. H., "Firmware/Hardware Support for Operating Systems: Principles and Selected History," *SIGMICRO Newsletter,* December 1975.

A multi-microprocessor approach to a high-speed and low-cost continuous-system simulation

by RYOICHI YOSHIKAWA, TATSUO KIMURA, YASUHIRO NARA, and HIDEO AISO
Keio University
Yokohama, Japan

ABSTRACT

A high-speed continuous-system simulator of the multi-microprocessor configuration will be presented. This system is composed of simple microprocessor units and carries out highly parallel operations on a small task level. The processor unit is controlled by the microprogram and only performs basic operations such as integration, addition, and multiplication. The results of the performance evaluation show that the simulator can generate a sine-wave at 3.5 KHz with an accuracy of 0.1 percent. The simulator has a greater processing capability than a large digital computer system employing a high-level simulation language. The system presented here is also much less expensive than conventional digital simulators.

INTRODUCTION

The aim of continuous system simulation is to obtain a time-series solution of a set of differential equations, which describes a model of the continuous system. Analog computers and advanced hybrid computers have previously been utilized for simulation, and recently, digital computers also have been utilized for simulation.

In analog computers, high-speed operation units are available and when used with the analog computer's native parallel operation feature, response times become fast enough to be used for most real time simulations. On the other hand, analog computers have problems of scaling, accuracy of solution, realization of non-linear components, and preservation of programs. In order to use a hybrid computer, A/D and D/A conversion techniques, as well as analog and digital techniques must be mastered. On the other hand, a digital computer not only greatly alleviates these undesirable problems, but also permits the user to communicate with the computer using a high-level language. These features have made digital simulation popular, but digital simulations are very expensive and too slow to be used for most real-time applications.

Recently, however, a high-speed and less expensive digital simulation system has been suggested by Korn.[2] It is the multi-processor system composed of standard mass-produced minicomputers. In this system, parallelism, contained in a continuous system simulation, is utilized to guarantee fast and low-cost simulations. From the consideration of parallel processing, it seems that a very low-cost yet high-speed digital simulation system could be realized by using recently developed microprocessors and by making full use of the parallelism contained in simulation processing. With this in mind, the authors have designed a continuous system simulator (KCSS: Keio Continuous System Simulator) which carries out highly parallel executions on a small task level, and have now implemented a prototype system called KCSS-1.

SIMULATION PROCEDURE AND MICROPROCESSORS

The digital simulation of a continuous system is an iterative operation of an integration step, i.e., derivative evaluation and integral calculations to obtain a time-series solution of the system variables. For numerical integration, methods such as the Euler method and the Runge-Kutta method are generally used. The following are noticeable features of simulation processing:

(1) Simulation processing generally contains a high degree of parallelism.
(2) Processing consists of recursive operations of predefined and fixed calculations, that are never altered within the simulation run.
(3) The continuous system of most engineering models is often partitioned into blocks (subsystems) requiring relatively little intercommunication.

The first feature means that in a continuous system simulation, a well tailored parallel processing system has the great advantage of being fast as well as low in cost. The second and the last features imply that a sophisticated task scheduling mechanism and a shared memory are not necessary, and even a simple bus configuration of the multi-processor system will produce few conflicts in data transfer. It should be emphasized that even a multi-processor system of simple structure can offer a great deal of processing capability, and that when the above three features are incorporated with good design one can be ensured of an effective continuous-system simulator.

A microprocessor, made possible by the advancement of the LSI technology, is appropriate for simple applications and has the features of being low in cost and yet good in performance. This implies that a low-cost microprocessor will give the possibility of bulk use in realizing a continuous system simulator of good performance using highly parallel solutions. In order to realize cost-effective parallel processing systems, it is very important to fully utilize the parallelism in the procedure and also to take advantage of the low-cost characteristic of the microprocessors. Therefore, a processor unit, which is the operational element of the system, should be designed to execute one of the basic operations in the simulation, and its hardware should consist of an LSI microprocessor and additional ICs without sophisticated external circuits in order not to sacrifice the low-cost characteristics.

The above discussions are reflected in the design policies of the KCSS as follows:

(1) The Processor Unit (PU) of the KCSS only executes one of the basic operations required for simulation at a time, such as integration, multiplication, or addition.
(2) The PUs are interconnected to each other physically or logically before the simulation run to build up data transfer paths, which are determined by the processing procedure to be executed. In the basic KCSS, described later, the PUs are physically interconnected using dedicated data buses and a logical interconnection mechanism is also discussed.
(3) A simple mechanism, instead of the interruption, is employed for the transfer of data between PUs, i.e., each PU decides, by itself, whether or not it can transfer data using I/O test instructions. A control processor for the management of data transfer between PUs is not necessary in KCSS.

THEORETICAL ASPECT OF PARALLEL PROCESSING

The details of parallel processing employed by KCSS were given in a previous paper.[3] The unique parallel processing on a small task level is outlined again in this chapter. The simple Euler method is adopted in the following discussion.

An example of the PU connection diagram is shown in Figure 1. This is very similar to a set-up diagram for analog computers, in that the operation task corresponding to an analog unit is assigned to a PU of the KCSS. Each PU, which is supposed to have a large input data buffer, starts its operation when it gets the operands, and then sends the result to its successor PUs connected to the output port. The simulation processing proceeds in a manner such that data travel along the loops organized by connecting PUs. At the beginning of the operation, these data are originally sent out from each integrator PU with their initial values. In Figure 1, one datum travels along LOOP 1, and two data along LOOP 2. The integrator PU gets its derivative value as the operand, and calculates the value for the next integration step.

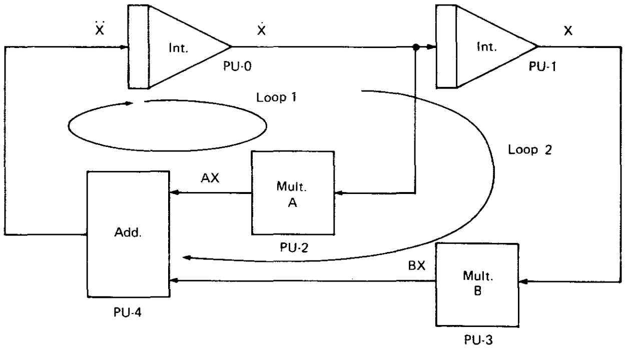


Figure 1—Example of the PU connection diagram: Ẍ=AX+BX

The processing time of one integration step for each loop is obtained as follows:

$$T_{loop1}=T(Int.)+T(Mult.)+T(Add.)$$

$$T_{loop2}=T(Int.)+\frac{1}{2}(T(Mult.)+T(Add.))$$

In general,

$$T_{loopi}=\frac{\text{The time required to circulate a datum around the loop}}{\text{The number of integrators in the loop}}$$

$$=T(Int.)+\frac{1}{n}\sum_{j=1}^{m}T(f_j)$$

where, n is the number of integrators in the loop and $T(f_j)$ is the processing time of task f_j. In the case, however, that $loop_i$ contains a time-consuming task requiring more processing time than T_{loopi}, it causes injury to a smooth data flow and the integration-step time is restricted by its processing time. Therefore,

$$T_{loopi}=Max\{T(Int.)+\frac{1}{n}\sum_{j=1}^{m}T(f_j),\ Max(T(f_j)\,|j=1,\ldots,m)\}$$

Since all loops of a simulation model are interrelated with each other, the processing speed of the simulation is determined by the most time consuming loop. Consequently, the effective processing time of one integration step is,

$$Te=Max(T_{loop}i\,|i \text{ for all loops})$$

The procedure for one integration step using a 4th-order Runge-Kutta method is divided into four sub-procedures, each of which is almost like the procedure for one integration step using the Euler method. Therefore, the effective processing time of one integration step using a 4th-order Runge-Kutta method is almost four times the processing time using the Euler method.

STRUCTURE OF CONTINUOUS SYSTEM SIMULATOR

The architecture of the basic KCSS mentioned here is based on the prototype system already developed by the authors with a few modifications having been made. This architecture still leaves some room for further improvements.

System configuration of the basic KCSS

The basic KCSS illustrated in Figure 2 is a continuous system simulator with a multi-microprocessor configuration, which carries out parallel processing on a small task level. The PU is a data-driven processor controlled by a microprogram which starts its operation when the input operands have arrived, and transfers the result to the successor PUs when they have used the former results. The PUs are interconnected with each other using Dedicated Data Buses according to the simulation model or the simulation procedure. The Control Unit (CU) is an interface between the Host Minicomputer and the PUs. The Host Minicomputer executes the simulation support programs and I/O management programs. Before the simulation run, the minicomputer sends a Function Code and some parameters to each PU through the Control Data Bus. Each PU starts its operation when a command "RUN" is sent from the minicomputer, and then desired simulation results are transferred through the CU. When the required integration steps are completed, a command "HOLD" is sent to the PUs.

Structure of the processor unit

The PU of the basic KCSS, as shown in Figure 3, is composed of 4 bit-slice microprocessor chips (MMI 6701×4), several data registers, a ROM as the microprogram storage, and simple logic circuits to synchronize the data transfers among PUs. The information needed for each PU to perform its operation is loaded into the Parameter Registers before the simulation run. This information includes a Function Code (starting address of the microprogram routine to be executed), an integration step size for integration, a constant term for coefficient multiplication, and so forth. A Status Register is provided to inform the minicomputer of the execution status of the PU, but it is also used as a buffer register to transfer the contents of the internal registers when a command "EXAM" is sent. The Instruction Register is provided for branching to the specified microroutine, and is also used for microsubroutine calls. Input/Output Registers are buffer registers for data communication between PUs. The PU recognizes such commands as RUN, STOP, and EXAM from the minicomputer by testing command signals using its test instructions.

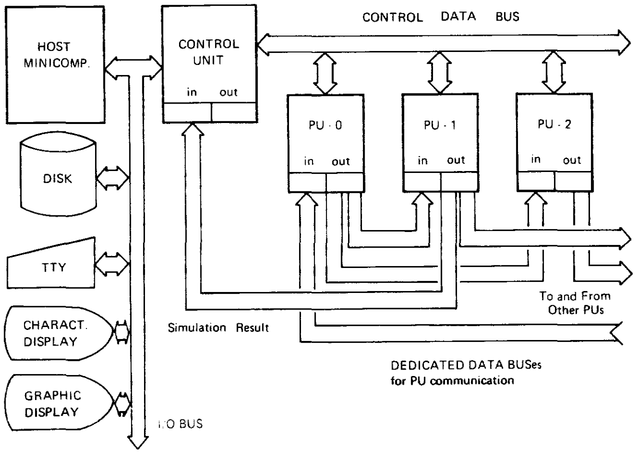


Figure 2—System configuration of the basic KCSS

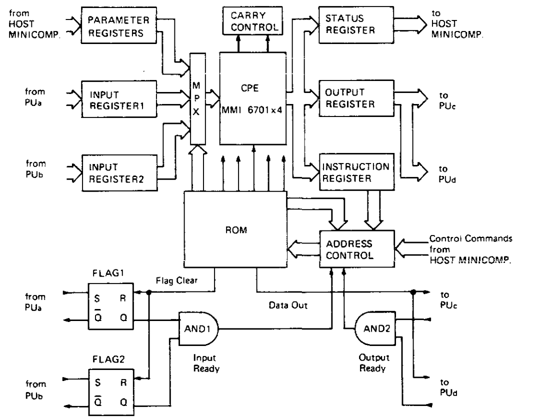


Figure 3—Processor unit of the basic KCSS

The logic circuits for the task synchronization consist of flip-flops (Input Flags), each corresponding to an Input Register, and a few logic gates. The Input Flag is kept to "1" while the corresponding Input Register holds a new operand. The term "Output Ready" is defined as the state such that all of the data stored in the Input Registers of its successor PUs have been used. Each PU recognizes the Output Ready state by testing the Output Ready signal, and sends out the Data Out signal to transfer the output data. The term "Input Ready" is defined as the state such that all of the Input Registers hold a new operand. Each PU recognizes the Input Ready state and accesses the input operands, and then clears the Input Flags to "0."

Microinstruction format

The PU is controlled by 40 bit horizontal-type microinstructions. The size and the function of each field are listed in Table I.

Microprogram and the simulation control

The microprogram stored in the ROM consists of an initialization routine, an input routine, an output routine,

TABLE I—Microinstruction Format

field	bit width	function description
A	5	ALU Instruction Code
B	3	Load and Shift Control
C	4	Source Reg. Number or External Input Reg. Number or Upper Emit Data (if E field designates "EMIT")
D	4	Destination Reg. Number
E	2	Field Control: if "EMIT", 8-bit literal data is obtained if "TEST", 2-way conditional jump is done
F	4	Upper Next Address or Test Condition (if "TEST") or Lower Emit Data (if "EMIT")
G	4	Lower Next Address
H	1	Load IR to ROM Address Reg.
I	2	Carry Flag Control
J	2	Input Carry Control (Arithmetic)
K	2	Input Carry Control (Shift)
L	2	External Output Reg. Number
M	1	Clear Input Flags
N	1	Data Out
-	3	Unused

and some arithmetic routines. Integration, addition, multiplication, coefficient multiplication, and so forth are provided as arithmetic routines. The Function Code designates the starting address of one of the arithmetic routines, and also specifies the through/negate operation for each input operand. The initialization routine is provided to intialize the contents of the registers and the Input Flags before the simulation run and also to recover from operation errors. When the PU has recognized a RUN command, it executes the specified arithmetic operation in the following manner:

(1) Input Routine—The PU waits until it detects the Input Ready state, and then branches to the arithmetic routine designated by the Function Code.
(2) Arithmetic Routine—The PU gets the operands into internal registers and clears the Input Flags to "0" to enable the new operands to be loaded. After the completion of the arithmetic operation, the PU stores the result in the Output Register, and goes to the output routine.
(3) Output Routine—The PU waits until it detects the Output Ready state, and then outputs the data, and goes to the input routine.

In the case of the integration operation, the PU outputs the initial value before going to the above operation sequence. The HOLD command, accepted in the input or the output routines, causes the program sequence to branch to the initialization routine.

Current development and discussions of KCSS

(1) Prototype system: KCSS-1—A prototype system, named KCSS-1, has been developed as a straightforward implementation to prove the effectiveness of the parallel processing on a small task level. The PU of the KCSS-1 is composed of two microprocessor chips as the CPE (Central Processing Element) and 59 other ICs, and has a parallel 8 bit operation capability. A 40 bit×256 word ROM is used to store several double-precision arithmetic routines for integration, addition, and multiplication. The PU is already in operation with a 350 ns clock. As the Host Minicomputer for simulation control, a NOVA model/01 is connected to the PUs, and some support software programs under the control of the NOVA RDOS (Realtime Disk Operating System), such as an interactive microprogram assembler, a microprogram simulator, a simulation control command interpreter, and so forth are provided. The CU, an interface between PUs and the minicomputer, has been implemented with 88 ICs. One of the PUs has a WCS (Writable Control Storage) instead of a ROM. The KCSS-1 has been able to generate a sine wave at 10 Hz with an accuracy of approximately 1 percent employing the Euler method.

(2) Automatic interconnection mechanism for the PUs—A flexible interconnection mechanism for the PUs seems to be required so that the PU connection can be altered by the host minicomputer according to the model to be simulated. An interconnection mechanism for the automatic interconnection of 8 PUs has been designed. This mechanism is composed of 62 ICs and one data-transfer sequence is completed in 200 ns. According to the data-transfer simulation, this mechanism causes no serious overhead to data transfer. The interconnection mechanism and 8 PUs organize a PU block, and such elementary PU blocks are planned to be connected to a higher-level interconnection mechanism.

(3) Substitution of queueing register for input register—It is assumed in the theoretical study of parallel processing that each PU has a large input data buffer. The PU of the basic KCSS has only one register at each data input port. This may cause interference with the smooth flow of data or with effective parallel processing in the case where two or more data loops of greatly different processing times have common data paths. This interference is eliminated by substituting for the input register an input queueing register (FIFO register) of, in general, a few words. This requires no modifications to the data input/output interface or the microroutines.

(4) Simulation software—The translator for a simulation-oriented language and simulation control software is a necessity for a useful simulation system. In KCSS, each PU has a one-to-one correspondence with the operational elements of analog computers. Therefore, the model description of continuous systems in the form of a block-oriented language is very convenient for KCSS because each expression simply shows the function of the PU and its connection. A translator of an equation-oriented language has been developed by the authors where differential equations are given in the form of FORTRAN-like expressions. This translator also accepts the model description of the block-oriented language. A command interpreter for simulation control has also been developed. This allows the user to control the simulation interactively with easy operations.

(5) Compound function of the PU—PU utilization of the basic KCSS is low, because in general problems the data loop is often organized with a larger number of PUs than the number of data in the loop. For example, loop 1 of one datum in Figure 1 is organized with 3 PUs, and loop 2 of two data is organized of 4 PUs. Therefore, it seems that an efficient approach to solving this problem would be to

define common operation sequences of those operations which frequently appear together in simulation processing, as compound functions of the PU, for example, addition and integration, coefficient multiplication and addition, addition and coefficient multiplication, etc. The compound functions would increase PU utilization and also decrease the data communications among PUs so that the overall system performance would become greater.

PERFORMANCE EVALUATION

In order to clarify the processing performance of the KCSS, the execution speed and accuracy of the basic KCSS has been evaluated for some cases. The evaluation has been made for each operation method as shown in Table II, and the 4th-order Runge-Kutta method has been employed. For fixed-point arithmetic, the result of integration has been kept in double precision in order to reduce the accumulation of round-off errors. In each case, it has been assumed that the PU has a FIFO register at the input port. In the particular case where the integration step size is given in the form of $3\times2^{|n}$, special microroutines for the integration have been written and used in the evaluation. In this case, better results, in speed and accuracy, have been obtained because a shift operation can be used instead of multiplication.

Execution time of each basic operation

Microroutines for each basic operation have been coded with primary interest in execution time. The actual execution microsteps of these routines including data input/output operations have been determined as shown in Table III. The floating-point microroutines have been coded with the assumption that the PU has a 7-bit shift and a few additional bit-test functions, which can be incorporated with only a fractional cost. The execution cycle time of the microinstruction has been assumed to be 200 ns, which can be realized with appropriate hardware design.

Benchmark problems and simulation results

(1) Sine-wave generation problem—The first problem used in the performance evaluation is the solution of the differential equation, $\ddot{X}+X=0$, with initial values $X=0$ and $X=1$, from $t=0$ to 25. Thus the time, T_R, required for the real time solution is 25 sec. In this problem, the time required for one integration step, T_e, is $4\times T(\text{Int.})$. Then the time, T_S, required to produce a real-time 25 second solution is;

TABLE II—Operation Methods Employed for Performance Evaluation

	Fixed Point Single Precision	Fixed Point Double Precision	Floating Point
Operands and Results	16 bits	32 bits	Exponent:16 bits Mantissa:32 bits
Internal Results of the Integration	32 bits	48 bits	ditto

TABLE III—Execution Microsteps and Times of Each Operation

	Fixed Point Single Precision	Fixed Point Double Precision	Floating Point
Addition	6 (1.2 μs)	9 (1.8 μs)	46 (9.2 μs)
Multiplication	25* (5.0 μs)	146* (29.2 μs)	171* (34.2 μs)
Integration** with Mult.	40* (8.0 μs)	175* (35.0 μs)	281* (56.2 μs)
Integration** with Shift ($h=3\times2^{-n}$)	18+n*	25+3n*	63* (12.6 μs)

* Theoretically derived average of execution steps for random input operands.

** Average steps per stage in the four-stage operation of the 4th-order Runge-Kutta method.

$$T_S=\frac{T_R}{h}\cdot T_e$$

Therefore,

$$\frac{T_R}{T_S}=\frac{h}{T_e}$$

This means that KCSS is h/T_e times faster than the real-time. And the frequency of the sine-wave generated is;

$$F_S=\frac{h}{2\pi T_e}$$

The simulation results appear in Table IV.

(2) Van der Pol's equation—The second problem is the solution to Van der Pol's equation with initial values, $\dot{X}=0$ and $X=1$ from $t=0$ to 25. This is a typical nonlinear problem. In this problem, $T_R=25$ and $T_e=4T(\text{Int.})+6T(\text{Mult.})+4T(\text{Add.})$. The representative simulation results are shown in Table V. More accurate solutions have been obtained with reduced integration-step sizes; in the case of $h=3\times2^{-8}$, $T_R/T_S=52$ and maximum error=0.021; and in the case of $h=3\times2^{-9}$, $T_R/T_S=26$ and maximum error=0.0021.

The same problem has been solved on the DARE III B simulation system,[1] a CSSL-type simulation language on a

TABLE IV—Simulation Results of Sine-wave Generation Problem (integration step size: $h=2^{-3}$, truncation error of RK-4:0.085%/cycle)

	Te(μsec)	TR/Ts	Fs(Hz)	Accuracy*(%)
Fixed Point Single Pricision Int. with Mult.	32.0	11,700	1,870	0.13
Fixed Point Single Precision Int. with Shift	16.8	22,300	3,550	0.085
Fixed Point Double Precision Int. with Mult.	140	2,680	426	0.085
Fixed Point Double Precision Int. with Shift	27.2	13,800	2,190	0.085
Floating Point Int. with Mult.	225	1,670	265	0.085
Floating Point Int. with Shift	50.4	7,440	1,180	0.085

* Maximum absolute error per cycle:

$$(1+\text{Max}\,\varepsilon)^{\frac{t}{2\pi}} \geqq \frac{|(\text{KCSS Result}:\dot{X})-(\text{Exact Value}:\dot{X})|}{\text{Maximum Value of }\dot{X}}$$

TABLE V—Simulation Results of Van der Pol's Equation (integration step size: $h=2^{-7}$, truncation error of RK-4:0.311%)

	Te(μsec)	TR/Ts	Accuracy*(%)
Fixed Point Double Precision Int. with Shift	225**	104	0.311
Floating Point Int. with Shift	292	80.2	0.311

* Maximum absolute error from t=0 to 25:

$$\varepsilon = \frac{\text{Max}\left|(\text{KCSS Result}:\dot{x}) - (\text{Exact Value}:\dot{x})\right|}{2}$$

** This Te containes the execution time of some additional operations for appropriate scaling.

CDC 6400, and it required 1.7 seconds for the simulation run. The accuracy of its solution is not explicitly noted, but it is estimated to be worse than 2 percent according to high-accuracy numerical analysis. Thus the KCSS can solve this equation with floating point arithmetic operations at least five times faster than DARE III B.

Simulation summary

Each of the simulation results shows that the KCSS has a great processing capability. The processing speed of the KCSS is not so inferior to most high-accuracy analog computers. Furthermore, it is noticeable that low-cost microprocessors offer a higher performance than large digital computer systems. As a remarkable feature of the KCSS, the processing speed does not depend on the problem size. From the consideration of the parallel processing concept, the KCSS clearly offers a great deal of processing capability for simulation models represented by higher-order simultaneous differential equations such as transmission-line problems. The integration with the shift operation is faster and more accurate than the one with multiplication as shown in Table IV.

CONCLUSION

The architecture of the basic KCSS, a continuous system simulator of the multi-microprocessor configuration, was proposed. The performance evaluation based on computer simulations shows that the basic KCSS generates a sine-wave at 3.5 kHz with an accuracy of 0.1 percent. The processor unit constituting the KCSS can be implemented at the IC cost of approximately Y100,000 ($330). It has become clear that a digital continuous-system simulator, with multimicroprocessor architecture like KCSS, offers a high-speed simulation capability and better cost-performance when compared with conventional large-scale digital computer systems employing high-level simulation languages. The Processor Unit of the KCSS is controlled by the microprogram so that many types of operation methods convenient for various simulation problems can be optimally incorporated into it. This flexibility is very attractive in realizing a useful digital simulation system. Also, the processing speed of an optimally microprogrammed PU can be faster than that of a conventional microprocessor. It is believed that in the near future conventional analog computer systems can be replaced by fast multi-microprocessor systems.

ACKNOWLEDGMENT

The authors would like to gratefully acknowledge the appropriate directions, support, and consultation of Mr. Eiichi Yura, the former chief of the KCSS group at Keio University, who is now working for SONY Corporation. They also wish to thank Dr. Mario Tokoro and their colleagues in Keio University for their enthusiastic discussions, suggestions, and assistance. Also, their thanks go to Ms. Anne M. Marso of the University of Maryland for her kind assistance in checking this manuscript.

REFERENCES

1. Trevor, Alexander B. and John V. Wait, DARE III B—a CSSL-Type Batch-Mode Simulation Language for CDC6000-Series Computers, *SIMULATION*, Vol. 8, No. 6, June 1972, pp. 215-226.
2. Korn, Granino A., "Back to Parallel Computation: Proposal for a Completely New On-Line Simulation System using Standard Minicomputers for Low-Cost Multiprocessing," *SIMULATION*, Vol. 19, No. 2, August 1972, pp. 37-45.
3. Yura, Eiichi, et al., "An Approach to Parallel Processing for Continuous Dynamic System Simulation with Microprocessors," *Proceedings of Second USA-JAPAN Computer Conference*, 1975, pp. 172-177.
4. Yura, Eiichi, et al., "The KCSS: Keio Continuous System Simulator," Internal Report, Department of Electrical Engineering, Keio University, December 1975.

Instrumented architectural level emulation technology

by HARRISON R. BURRIS
TRW Defense and Space Systems Group
Redondo Beach, California

ABSTRACT

The advent of general purpose emulators as tools for computer architecture research and system development is briefly traced. The concepts of an architectural level emulation and of instrumenting an emulation are introduced. An operational emulation-based computer system development facility is described. The results of a 1976 Independent Research and Development program aimed at improving emulation development time, emulation execution time, and instrumentation flexibility are discussed.

INTRODUCTION

There are, at present, a number of operational or planned computer architecture and system development facilities that are based upon general purpose emulation technology and have requirements to evaluate many different computer architectures and software packages. These emulation oriented facilities have found application across all segments of the computer industry including: TRW's Computer System Development Facility,[1] Stanford's Emme,[2] the Argonne Microprocessor,[3] Army's Teleprocessing Design Center,[4,5] and Air Force's Space Data Systems Facility,[6] and Reconfigurable Computer Facility.[7] In spite of the successful implementations, further development of emulation technology is still required if system designers (hardware and software) are to be provided a reliable, cost effective system design tool. During the past year, TRW Defense and Space Systems Group conducted an Independent Research and Development program (for which the author was principal investigator) that had the objectives of (1) implementing an emulation based architectural research capability and (2) exploring the feasibility of several proposed extensions to that technology[8] that would either increase the range of problems to which emulation could be applied or would increase the flexibility of emulation technology from a user standpoint. This paper is, in part, a report on the result of this IR&D effort.

Emulation center evolution

In 1951, Wilkes suggested microprogramming as a more efficient means of designing a computer sequence controller.[9] Microprogramming also facilitated engineering changes to the architecture. By the mid 1960's, another benefit of microprogramming had been identified; by emulating earlier machines, microprogrammed computers could execute the software of the machines they replaced as well as software written in their own "native mode" instruction sets.[10] This software compatibility prevented the loss of significant investments in software. The advent of the user microprogrammable computer (writable control store) in 1970 made possible a third application of microprogramming, the emulation of many computer architectures by means of a single host machine. The existence of general purpose emulators made possible an emulation-based system development facility. The next steps in the evolution path probably occurred in many locations at about the same time. The development of the Army Teleprocessing Design Center provides a clear history of the next steps.

In December 1970, the U.S. Army Computer System Command and elements of the Department of the Army Staff began exploring the application of emulation to a software development facility. The rationale was that a software developer supporting many different computers could replace separate systems at his development facility with a single general purpose emulator. This machine could then be microprogrammed to emulate whichever of the machines was required for developing a piece of software. Further consideration of the potential for emulation within the Army led to the proposal[8] of additional benefits: (1) Software development tools (e.g., traces, variable range checks and activity counters) can be embedded in the emulation permitting software testing without destroying the integrity of the software under test (e.g., address realignment due to breakpoint code), and (2) The hardware performance (e.g., signal timing traces) can also be modeled by emulation, thereby permitting evaluation of hardware and hardware/software design trade-offs. In 1973, the Army Teleprocessing Design Center[4,5,11,12] was implemented under the direction of the Office of the Project Manager for Army Tactical Data Systems with the mission of providing both hardware and software evaluations.

INSTRUMENTED ARCHITECTURAL LEVEL EMULATION

Architectural level

Bell and Newell, in their book on computer architecture,[13] defined a hierarchy of levels at which computer structure may be described, two of which (programming level and logic design level) are of daily interest to the system designer. For the purposes of this paper, the architectural level model of a computer structure is considered to include both the programming level and logic design level since it permits hybrid models, portions of which are implemented at each of these levels of representation.

Emulation

Computer system simulations in which the processing functions performed by one machine (the target) are replicated on another computer (the host) have been termed functional simulations.[14] While both interpreters and emulators are functional simulations of a target machine, they differ in the method of implementation. The interpreter is a software program that is written in a language executable by the host computer and which accepts as input data and then executes computer programs written for the target computer. An emulator is a set of equivilances (microprograms) between operation codes of a target machine language and the processing logic of the host computer. When the microprograms for a target language are loaded into the sequence controller of the host computer, programs written in the target machine language are decoded and executed by the hardware of the host machine. Traditionally, the suitability of an interpreter or emulator for a particular project was determined by the trade-off: interpreters are fast (relatively low cost) to implement and modify and are more general (more space available for code), but have slow execution speeds: Emulators are slow (relatively expensive) to implement and modify, and restricted as to complexity modeled (limited control store), but execute considerably faster than interpreters for the corresponding target and host machines.[1,15] In the immediate future, this trade-off will probably remain valid even for the high level language (Direct Executing Language) machines, since it would seem that the execution speed advantage remains with directly emulating the target machine instruction set instead of emulating a high level language and using an interpreter program written in the HLL to simulate the processing of the target machine.

A current paper[16] succinctly illustrates that microprogramming (emulation) is becoming more user-oriented in a manner similar to the evolution experienced with "conventional" software. Recent developments in high level microprogramming languages,[17–19] the availability of larger (and writable) control stores and other techniques being reported in this paper are beginning to eliminate the development cost and generality advantages of interpreters, and it seems entirely consistent to predict that within three to five years, emulation will be the implementation method of choice whenever the functions of one computer are to be replicated by another. However, until some of these newer developments have matured and are generally available, the above trade-off will remain an important consideration in deciding to emulate or interpret a target machine architecture.

Instrumentation

Computer instrumentation is the control and performance of (1) measuring, (2) screening, (3) recording, (4) processing and (5) displaying data describing the operation of the target computer. This ordering of activities does not necessarily reflect the operational relationship in an instrumentation system. While measurement is first and display last, the intermediate steps vary widely. For example, all measurements may be recorded and uninteresting data eliminated (screened) as part of the processing. Also, the instrumentation process may be interrupted by recording data and then processing and displaying it sometime later (off-line). Conventionally, instrumentation is performed by means of hardware monitors (probes and logic analyzers) attached to the physical hardware of the target system, or by software monitors (programs executed by the system being measured). A discussion of these techniques including identification of their difficiencies and proposed solutions is found in Reference 20, and a detailed treatment of computer instrumentation that is perhaps as thorough as is possible for such a new and rapidly evolving topic appears in a recent book by Svobodova.[21]

The particular measurements to be performed during an instrumentation exercise will depend upon the question being investigated. Typical classes of measurements are: (1) interval timing (2) event counting, such as utilization figures for hardware resources, and software characteristics such as page fault frequencies and branching, (3) discrete values, such as those resulting from computations and replacements, and (4) extents such as the amount of primary or secondary storage consumed by a process. It is often very difficult or even impossible to obtain some of these measures on particular target computers because there is no way to attach a probe to the signal of interest since it is deep within the physical hardware.[22–24] Also, there is nothing to instrument if the particular configuration of equipment for which the measurements are desired is either not in existence (under development) or not available. If an ALE modeling all of the signals and architectural entities of interest to the investigator is constructed, the desired measurements can be obtained by instrumenting the emulation of the target system rather than by instrumenting the actual hardware.

The idea behind instrumenting an emulation of a target system instead of instrumenting the target system itself is this. General purpose hardware and software instrumentation techniques are available. These methods can be applied to the host computer system as well as to the target computer system. The emulation "programs" representing the target system are microcode and, in some cases, special "emulation mode" hardware resources. The instrumentation of the emulated target machine hardware such as the program counter then reduces to instrumenting the memory location or register of the host machine where the emulated target machine program counter resides.

The concept of instrumenting an emulation of a machine is best illustrated with an example. If the contents of a buffer register and the timing sequence of signals on two control lines are of interest, an ALE, including register transfer entities for these three machine features, must be created. Then instead of placing probes on the register and two signal lines, the memory locations with which the emulator represents the three register transfer level entities are examined by the instrumentation. The contents of the buffer register and presence of 1's or 0's on the signal lines will be the same as though the actual hardware had been probed.

If the ALE has already been implemented, instrumentation of the ALE can also be faster and less costly to perform than installing the probes on the physical hardware of the target machine since the only task involved is identifying addresses of memory locations within the host computer corresponding to target machine resources. This same flexibility that makes initial instrumentation of the emulation easier than instrumenting the hardware also makes it easier to reconfigure the probe points. With an appropriate instrumentation control program, dynamic reconfiguration during a measurement experiment is also possible.

COMPUTER SYSTEM DEVELOPMENT FACILITY

The Computer System Development Facility (CSDF) used for the IALE development IR&D is configured from mainframe computer systems maintained in TRW's Minicomputer System Facility. A block diagram of the CSDF is presented in Figure 1. The host machine for the emulation is a Nanodata QM-1.[25] The QM-1 is an extremely flexible two-level (micro and nano) user microprogrammable machine of the type characterized by Flynn as a soft computer architecture.[26] The QM-1 design supports dynamic modifi-

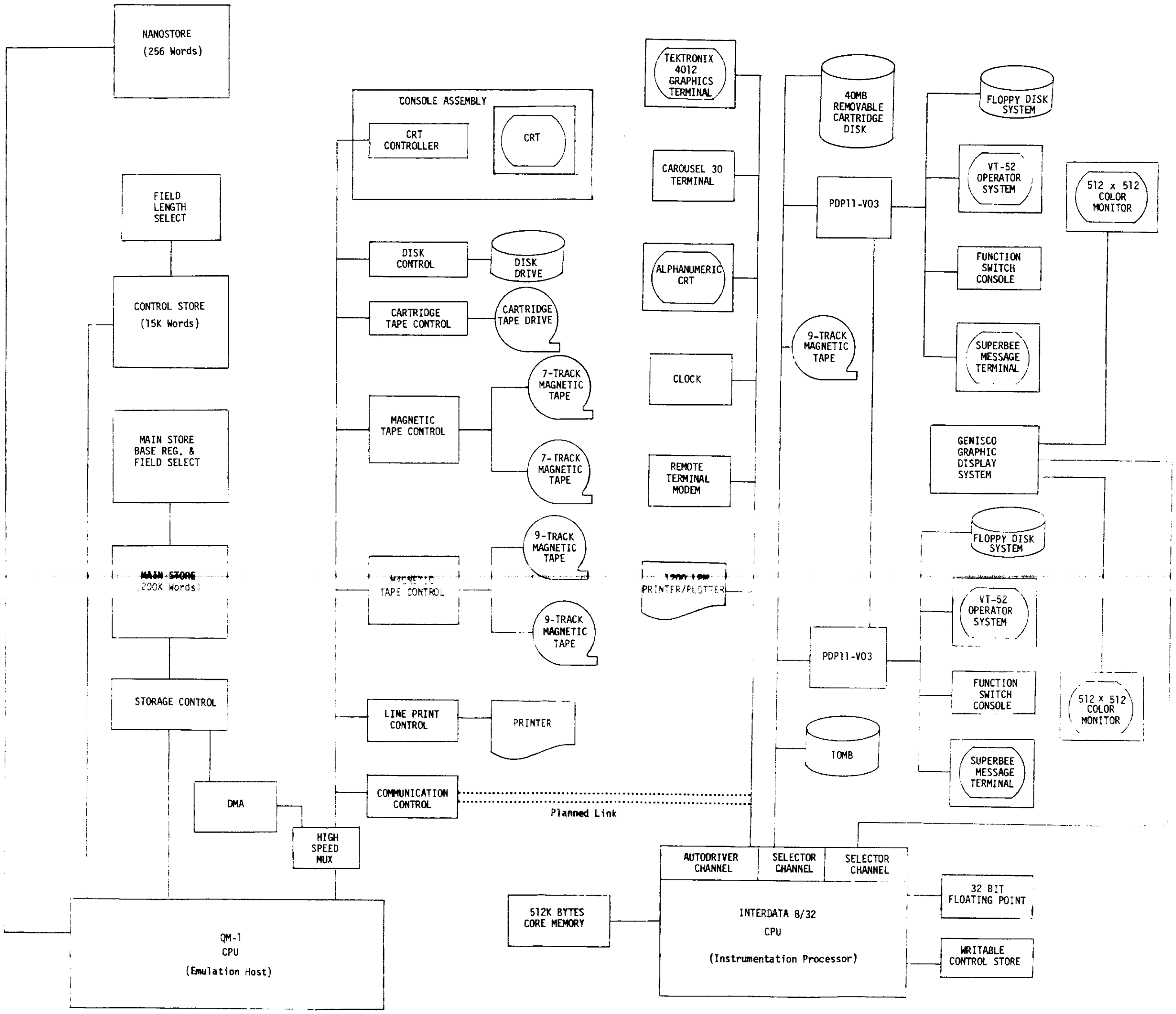


Figure 1—Current CSDF configuration

cation of both the micro and nano level programs. The microinstruction set is not fixed, but instead depends upon the mapping to the nano-level instructions. This is extremely important for the emulation of a large number of radically different target architectures. The processing and display of instrumentation data is performed by an Interdata 8/32[27] configured with a Genisco Graphic Display System.[28] At present, the connection between the measurements recorded by the QM-1 and the processing and display of the Interdata 8/32 is by means of a 9-track tape at 1600 BPS. An 8-bit parallel 9600 CPS bus between the two processors is currently being installed and dynamic interaction between the QM-1 and the 8/32 should be possible by March 1977. As part of the IR&D effort, it was desired to validate the instrumentation results obtained by the IALE (this is discussed in more detail later). For this purpose, an IMSAI 8080[29] microprocessor system with much of the architectural generality of larger computer systems (as opposed to the microprocessor system development kits) was selected as the target architecture. The target architecture is shown in Figure 2. The unit was specially designed to support hardware instrumentation and both a minicomputer (PDP11/20) and a Hewlett-Packard logic analyzer were used to collect measurements.

Concept of operation

The following description of the use of the system development facility is presented to illustrate the capability desired. The system developer constructs the desired target system emulation at the level of detail appropriate to his experiment (hardware alternative selection, software debugging, etc.) by assembling already defined emulation building blocks from a library supplemented as necessary with newly defined emulations (which could be added to the library once they pass the accuracy of emulation test[1]). Next, the desired measurements are specified in terms of the target machine architecture (e.g., busses, registers, status indicators, etc.) to be examined and the conditions (events) to be reported. The processing to be performed (e.g., calculation of channel and CPU active and wait state percentages) on the measured data and the method of display (e.g., tabular, Kiviat diagram,[30,31] etc.) is specified. The emulation and measurement microcode is loaded into the QM-1 micro and nano control stores under control of a "micro operating system", MOS. The target machine software is loaded into upper mainstore (lower mainstore contains the overlay library used by the micro operating system). The remaining instrumentation software is loaded into the instrumentation processor and the emulation experiment is initiated. The results of the instrumentation would be displayed to the user and the measurement, processing, and display could be interactively redefined during the experiment. While emulation microcode could be compiled or assembled under control of a more extensive version of MOS, the more straightforward method of iterating emulations is used. The system user must return to the support software provided by NANODATA to redefine the emulation. This process can still be initiated from the console, however, permitting a fully interactive evaluation session with a series of emulation experiments.

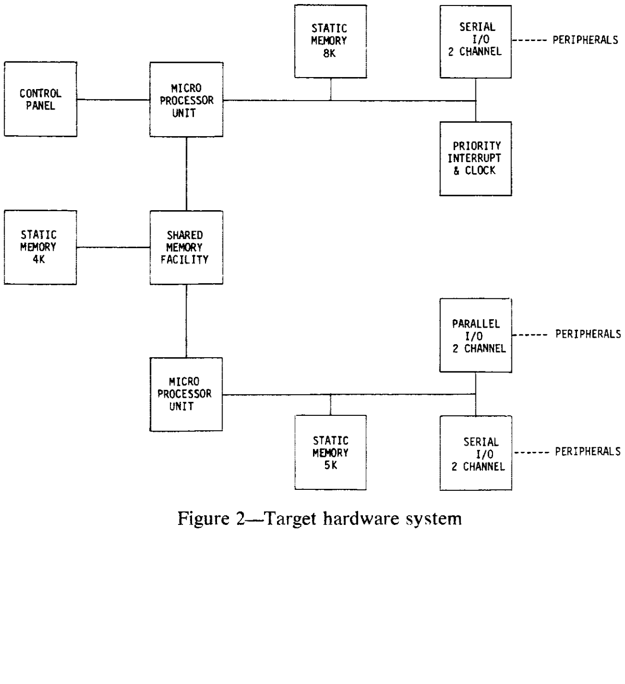


Figure 2—Target hardware system

IALE feasibility demonstration

The concept of the feasibility demonstration was to implement an Instrumented Architectural Level Emulation of a particular target hardware configuration, build the target hardware system, perform instrumented experiments on both implementations, and compare the results. Considerable effort was expended during the project planning[32] to design a set of feasibility demonstration experiments which would meet the twofold objectives of demonstrating the feasibility of IALE and also providing data on a target system application of intrinsic interest. The IMSAI 8080 microprocessor system to be used as the target hardware was assembled by the author using a mother board design intended to provide maximum access to each card and 2-level wirewrap sockets were used on the printed circuit boards instead of soldertail sockets to provide an easy means of connecting instrumentation probes (see Figure 3). The 8080 software selected for the demonstration involved encryption and fault tolerant communications processing. Detailed applications oriented analysis of the results of these experiments have been presented elsewhere.[33,34] Figure 4 shows the process used to conduct each experiment of the feasibility demonstration. Aside from the lower path through the instrumented target system hardware, the process is the same as would be followed in performing any emulation-based experiments with the computer system development facility.

IALE TECHNOLOGY ADVANCES

As part of this project, improvements to the basic IALE concept were identified which had potential for improving

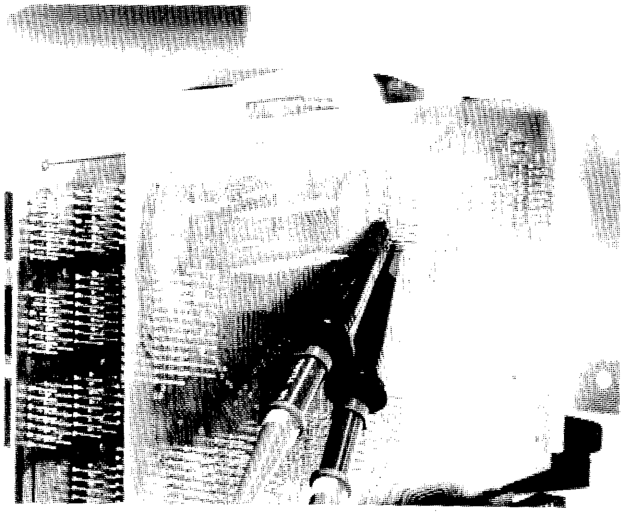


Figure 3—Target hardware and probe points

(1) emulation development time, (2) emulation execution time, and (3) modeling and instrumentation flexibility. The feasibility of each of these potential improvements and any restrictions they imposed upon the emulation process were explored. Several of the improvements under consideration had been previously employed with interpreter based systems, but had unique ramifications when applied to emulation based systems.

Microprogram operating system

Simulation at the circuit sublevel is used for some types of hardware analysis. If these simulations are to be performed by means of emulation, it is often necessary to somehow modularize the microcode since even the soft computer architectures do not provide sufficient control store for a program of this size. Even in cases where an

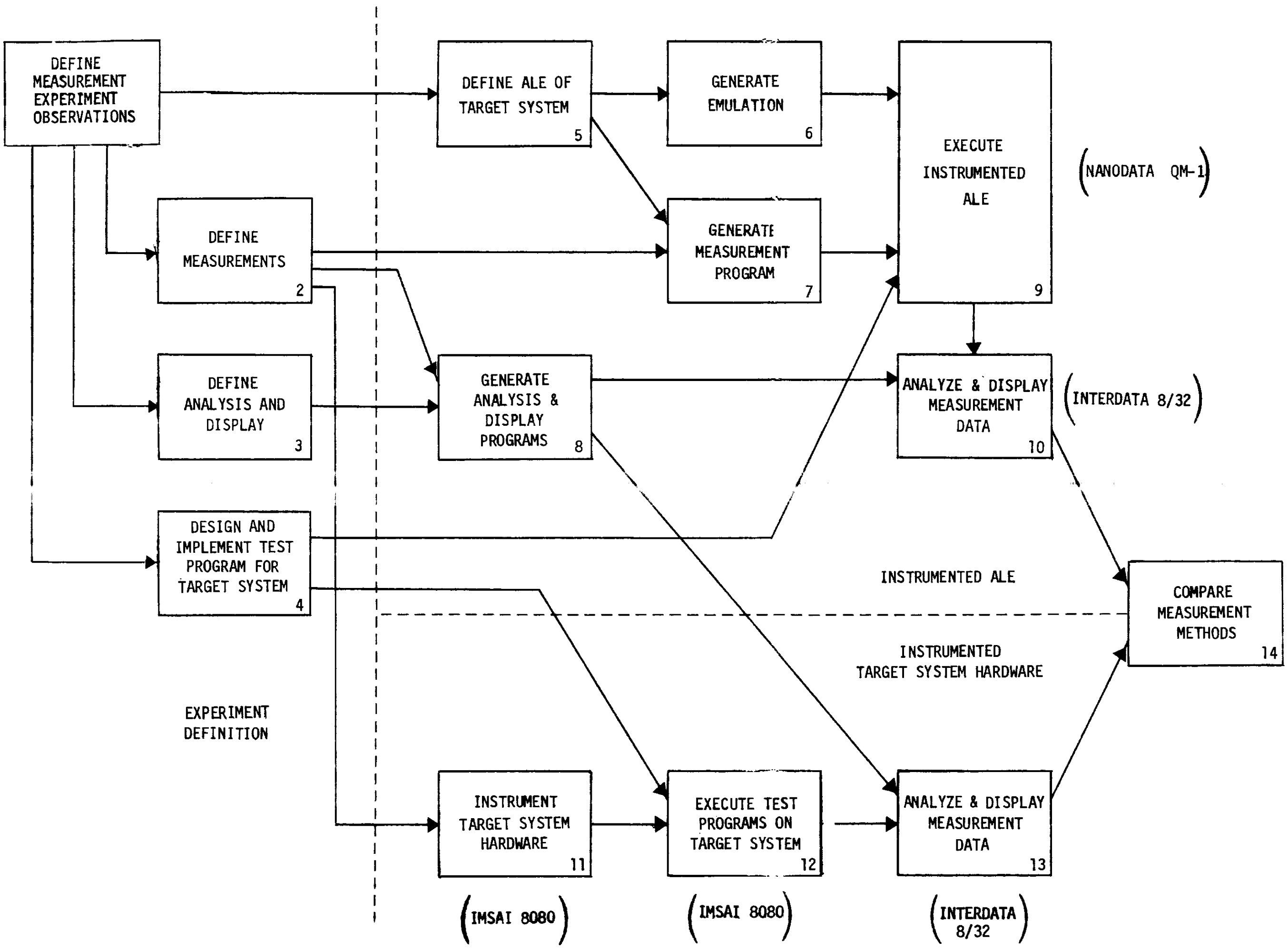

DEFINE MEASUREMENT EXPERIMENT OBSERVATIONS
DEFINE ALE OF TARGET SYSTEM 5
GENERATE EMULATION 6
EXECUTE INSTRUMENTED ALE 9
(NANODATA QM-1)
DEFINE MEASUREMENTS 2
GENERATE MEASUREMENT PROGRAM 7
DEFINE ANALYSIS AND DISPLAY 3
GENERATE ANALYSIS & DISPLAY PROGRAMS 8
ANALYZE & DISPLAY MEASUREMENT DATA 10
(INTERDATA 8/32)
DESIGN AND IMPLEMENT TEST PROGRAM FOR TARGET SYSTEM 4
INSTRUMENTED ALE
COMPARE MEASUREMENT METHODS 14
INSTRUMENTED TARGET SYSTEM HARDWARE
EXPERIMENT DEFINITION
INSTRUMENT TARGET SYSTEM HARDWARE 11
EXECUTE TEST PROGRAMS ON TARGET SYSTEM 12
ANALYZE & DISPLAY MEASUREMENT DATA 13
(IMSAI 8080)
(IMSAI 8080)
(INTERDATA 8/32)

Figure 4—Feasibility demonstration process

entire architecture at the appropriate level of detail can be emulated by wholly resident microcode if many separate architecture alternatives are to be evaluated, the capability to construct the emulation from a library of predefined building blocks that are combined by a linkage editing or parameter passing process considerably reduces the emulation development time. For these two reasons, a microprogram operating system (MOS) was considered critical to the IALE effort.

The MOS is loaded by the QM-1 support system, uses a directory and library of emulation modules contained in lower main store to build the emulation from a predefined root module, and this continues to provide microstore memory management during execution of the emulation. The only interrupt presently recognized is F switch 1 which is used as a run/halt control. Eventually, the MOS will also service interrupts from the Instrumentation Processor. Input/output functions including those of the measurement processes are supported by the individual emulation and measurement modules.

The particular microcode instruction set defined for the feasibility demonstration did not support relocatable microcoding. This necessitated programming the microprogram overlays for fixed overlay points within the micro-storage space, and proved to be a serious restriction upon the flexibility of which programs could be operated with each other. For example, two overlays moving in and out of the same overlay space severely slow down the speed of execution. An experiment examining the 8080 CPU and a circuit level model of a memory control model executed a segment of 8080 machine code 5727.46 times slower than the target machine when the CPU and control module emulation overlays were coded at the same overlay point (and had to be overlain whenever execution passed from one to the other). When the memory control module was recoded so that both emulations could be coresident, time to execute the 8080 program was reduced to 91.89 times that required by the target machine. This result demonstrates the impact that overlay contention could have on a fairly typical computer system experiment, and indirectly indicates the value of a relocatable program approach to microprogramming. Since relocation was not available on the QM-1, a set of fixed overlay spaces were defined with sets of building blocks associated with each. This, of course, restricted the building blocks that could be assembled into emulations. Approaches to implementation of relocatable microcode such as suggested in Jones[35] are considered to be one of the highest potential areas for further development.

The CPU and memory control module were also recoded as a single microcode program with a processing flow identical to the building block model except that the parameter passing support of the MOS was not needed and the emulation could execute as a stand-alone. This simulation was 86.19 times slower than the target machine. Therefore, for the particular segment of 8080 machine code being executed, a 6.61 percent execution time penalty is incurred to obtain the increased flexibility of modular emulations.

Emulation building blocks

One of the most significant trends in digital logic design has been the willingness of logic designers to forgo the speed advantages of discrete component systems for the design ease of MSI and LSI packages. The target system emulation design process can benefit from the use of predefined building blocks of larger functions in much the same way as the target processor hardware design process. A considerable reduction in emulation development time and cost could be achieved if an emulation designer could assemble already programmed building blocks into the desired emulation instead of having to code each emulation from scratch. This is just a further extension of the general argument for doing away with hand-crafted software.

As an example of the process, consider a small special applications device consisting of two identical micro-processor chips, a solid state memory chip, a contention resolving chip and a clock circuit. The logic designer can assemble the system design by joining together the chips without repeating the design of their insides. An emulation designer should be able to assemble the emulation of the system design, by providing code joining already working emulations of each of the chips being used. Furthermore, once this five-chip emulation is assembled, it can become a building block available for use in still bigger emulations.

This idea of a library of emulations imposes some additional design requirements on the emulations. The emulation code should be reentrant so that several copies may be active within a single larger emulation. For instance, the two identical micro-processors in the above example should require only one copy of the emulation code. Also, this library concept is a strong argument for the use of separate instrumentation routines since if the instrumentation is included in the emulation building block, it would have to be sufficient for the most general cases; which would be unacceptably slow and probably would make the code too big for most target systems of interest.

Multilevel emulation

When the host system is of a newer technology than the target machine, (e.g., an IBM 360 emulating a 1401) software often executes significantly faster on the emulation than on the target machine. However, when target and host are of the same technology or the host is being used to evaluate advanced target technologies, speed of execution can become a problem. In almost all cases, it seems likely that when emulations are used to simulate target machines (or subsystems) at the logic level, a speed penalty would be incurred. Thus, in the context of system development and computer architecture research, one of the problems with both interpreters and emulators is that they are slow in execution. The degree of slowness for both methods depends upon (1) the level of target machine detail being simulated and (2) the closeness of the mapping of the target hardware resources into the host hardware resources. Improving execution speed by augmenting the host hardware is rejected since once a host has been selected, little can be done to improve execution speed for general purpose emulation experiments requiring several different target machine architectures. If many emulation experiments are to be performed on a single target machine architecture (e.g., a software development facility), then it may be desirable to improve execution speed by replacing complex firmware procedures with hardware, such as the transform boards often used in CDC 5000 series based emulators.[5,11] However, if this approach were adapted when several alternative architectures were being considered, it might become necessary to construct a different transform board for each. This is nearly brassboarding each alternative—one of the very things IALE is attempting to avoid.

The other factor, level of detail being simulated, provides a general means of increasing execution speed. An approach to the definition of emulations (also applicable to interpretation) which I call multi-level emulation, can considerably reduce the time required to execute a particular emulation experiment by modeling in detail only those operations of the target system that are of interest and emulating at a higher level of abstraction the remainder of the system necessary to provide the input data for the lower level model.

Different types of system performance analyses (i.e., throughput, reliability, utilization, security) require the measurement of different system parameters. The hardware characteristics which must be observable in order to evaluate these parameters also vary. This observation implied that an increase in emulation speed due to reduced emulation complexity is obtainable by restricting an ALE to representations of target system hardware entities that are required for a particular measurement experiment instead of executing a general purpose (complete in every detail at the lowest logic level) ALE of the target system for all experiments. For example, during an experiment concerning the utilization of a memory contention controller and a shared memory, it was possible to model the memory contention controller at the circuit level while the CPU, I/O logic, and memory need only be emulated at a level of detail sufficient to generate the memory reference strings[36–38] needed to drive the memory contention resolver.

Figure 5 illustrates a multi-level ALE which could be instrumented to perform the evaluation suggested above. The CPU's and the remainder of the system, other than the memory control hardware of principal interest, are emulated at the RT level. This emulation, in terms of such constructs as program status words, floating point registers and interrupt status words would execute the object code equivalent of target system programs. Whenever a reference is made to memory (next instruction or operand store/fetch) the address is loaded into the architectural equivalent of a memory address register (MAR). The MAR and the equivalent memory buffer register MBR are crossover paths between the CPU and the model of the memory control hardware. The logic gate (AND, OR, XOR, etc.) representation of the MAR is loaded with the address contained in the RT level MAR and a cycle of emulation through the memory control logic is begun. At the conclusion of that cycle, control returns to the RT level CPU emulation which resumes processing where it was halted while the memory access was emulated.

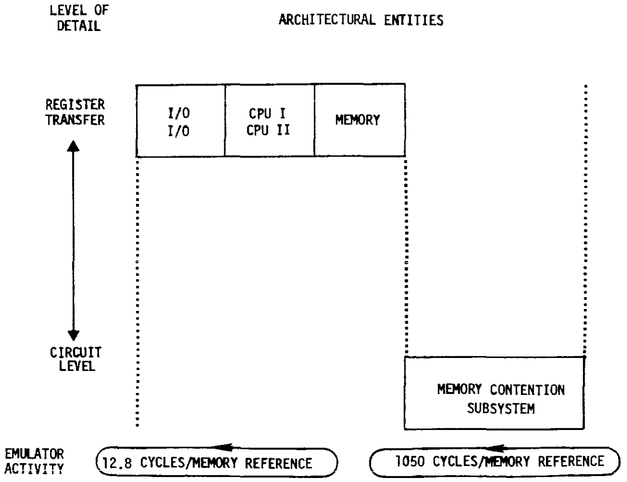


Figure 5—Ale using multiple levels of emulation detail

For one of the alternative contention resolution circuits evaluated a driver program of 8080 software executed 3354 times slower than the target machine when a circuit level approximation* of the CPU was used. When the same 8080 software was executed by an emulation using a register transfer model of the CPU, the execution time was reduced to 91.89 times that of the target machine, an improvement by a factor of 36.5 times. For the above comparison, only one CPU was modeled (no contention effects).

Emulation timing

When a functional simulation is used to determine how much time (or how many machine cycles) a target program ~~will take to execute, it is generally satisfactory to maintain~~ a simple counter or "virtual clock" which is incremented by the number of cycles required for each machine instruction. When the interaction of many signals (events) in a piece of hardware is being simulated, obtaining the correct timing between the signals is more complex. Approaches used include an event queue which synchronizes the modeled signals with a virtual clock that is regularly incremented,[39] or that synchronizes the signals with an external clock that is regularly sampled by the host.[4] An additional method of time-scaling an emulation is to include the signal synchronizing in the code for each step of an emulation.[14] This latter method is particularly useful for interfacing an emulation with physical hardware (e.g., universal interface board) under conditions where the emulation produces

* Some unaccountable aggregation of logic gates probably occurred since the internal design of the chip had to be assumed.

signals at precisely the same timing as the equipment being emulated (time-synchronous emulation[14]).

An emulation providing at least one level of detail at which events (e.g., occurrence of input and output signals) in the target system are related to the occurrence of the corresponding events in the emulation by a constant ratio (the scaling constant) is defined as a time-scaled emulation. When the scaling constant (emulated time÷target time) is equal to one, the emulation is termed time-synchronous.

The register transfer level 8080 CPU emulation was time-scaled to the 8080 instruction set with a scaling factor of 9. This was achieved without an attempt to optimize the emulation speed. However, it seems to indicate that the QM-1 (without hardware augmentation) is not sufficient to perform a time-synchronous emulation of the 8080 chip. Considerations of target/host architecture mapping and relative technologies are again a factor.

As a counterexample, an error detection and correction process used for 1200 BPS communication links in the TACFIRE and TOS[2] Army Tactical Data Systems was emulated by the QM-1 at a 0.87 scaling factor. The QM-1 could easily be time synchronized as a driver to examine the performance of these tactical computer networks.

Figure 6 illustrates the process used to develop a time-scaled emulation. The accuracy of emulation test requires that all input-output transformations of the target system be replicated in the emulation before time-scaling is attempted.

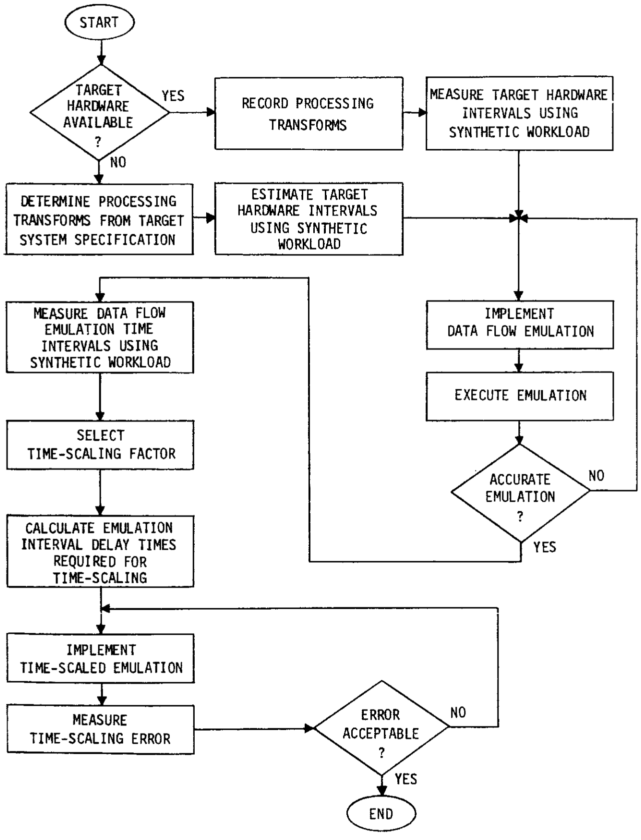

START
TARGET HARDWARE AVAILABLE ?
YES
NO
RECORD PROCESSING TRANSFORMS
MEASURE TARGET HARDWARE INTERVALS USING SYNTHETIC WORKLOAD
DETERMINE PROCESSING TRANSFORMS FROM TARGET SYSTEM SPECIFICATION
ESTIMATE TARGET HARDWARE INTERVALS USING SYNTHETIC WORKLOAD
MEASURE DATA FLOW EMULATION TIME INTERVALS USING SYNTHETIC WORKLOAD
IMPLEMENT DATA FLOW EMULATION
EXECUTE EMULATION
ACCURATE EMULATION ?
NO
YES
SELECT TIME-SCALING FACTOR
CALCULATE EMULATION INTERVAL DELAY TIMES REQUIRED FOR TIME-SCALING
IMPLEMENT TIME-SCALED EMULATION
MEASURE TIME-SCALING ERROR
ERROR ACCEPTABLE ?
NO
YES
END

Figure 6—Time scaled emulation development process

Emulation instrumentation

This section could equally well be titled "Instrumentation Emulation" because what was developed was a way of instrumenting an emulation by means of emulated instrumentation . Svobodova has suggested augmenting computer architectures with hardware to support measurement such as *interval times* and *event counters*.[20] What we propose is to emulate these measurement hardware facilities along with the emulation of the computer hardware they are intended to measure. This emulated measurement hardware can then be used by external measurement equipment and internal software in the manner proposed by Svobodova. A significant advantage to emulation of the measurement hardware is that the amount and composition of the measurement hardware can be varied to suit the desired data acquisition requirements of a particular experiment. This flexibility in part anticipates a new law of computer performance analysis: For any computer architecture having N general purpose interval counters, there will be at least one experiment of critical interest requiring measurement of N+1 intervals.

The choice of using separate measurement programs instead of building the measurement hardware into the target architecture emulation was made to preserve the maximum generality of the system. This choice, as well as a desire to provide for external measurement (i.e., by DMA) decided another feature of the emulation architecture. At the microcode (microstore) level, each storage device of the target architecture is represented by a location in a resource vector of words in microstorage (i.e., a COMPOOL of emulator data sets). Instrumentation code outside of the building block can refer to the target hardware by a displacement into this table. Since the displacement is relative to an address maintained by the MOS for each building block and this address is updated whenever this resource vector is overlain at a new point, the instrumentation is unaffected by microstore memory management actions. This organization of the emulation building blocks into executable code and separate data storage also permits reentrant coding by the creation of a new copy of the resource vector for each new use of the building block (each building block also accesses the resource vector through updatable relative addressing). Storage for all status and scratch areas used in the building block is appended just below each copy of the resource vector. An optimum policy for determining when to overlay a building block while retaining the resource vector in microstore card and when to overlay a building block and copy the resource vector to mainstore has not yet been formulated. For some of the circuit level blocks, it was possible to retain only the active resource vector in microstore because of size limitations. At present, all resource vectors are removed to mainstore when their building block is overlain and are restored when the block is again activated. Some circuit level building blocks (i.e., 8080 CPU) require overlays within the building block because of the length of the code, these overlays do not affect the resource vector. Some typical microstore sizings are shown in Table I.

TABLE I—Microstorage Utilization for 8080 CPU Building Block

	Register transfer level	Circuit level
Microcode	2974 words	57695 words
Resource vector	218 words	1452 words
% of Total Memory Requirement for Resource Vectors		
1 active copy	6.83%	2.45%
2 active copies	12.79%	4.79%

Conditional measurements

Not all the measurements taken during an experiment represent data of interest to the experimenter. For example, when determining which instructions cause the heaviest loads upon the ALU, any opcodes not associated with ALU operations are not of interest and can be deleted from opcode occurrence measurements. As the number of system parameters being measured increases, the difference between the amount of data present and the particular combinations that are of interest rapidly increases. In order to retain efficient host execution speeds, conditional measurements are introduced. When implementing the instrumentation, a choice exists between (1) recording of all of the measurement data and then screening the unnecessary data at the instrumentation processor as part of the analysis process or (2) screening the measurement data as part of the measurement routine processing. The objective is to minimize the amount of instrumentation processing that must be performed by the host and the trade-off is between the amount of time required to output all the measurement data as opposed to the time required to screen the data that is recorded. Bulk recording of measurement data is the most straightforward to implement, but, in general, results in unsatisfactorily slow execution of the emulation. The incorporation of the screening process into the host was, therefore, selected for implementation during the feasibility research.

The standard instrumentation building block accesses the object (host resource representation of a) target machine entity to be measured and then controls the I/O process responsible for outputting the measured data in a specified format on a selected I/O device (disk, tape, line printer, CRT or host front panel). The conditional processing logic is interposed between the measurement and the action (e.g., I/O) performed as a result of the measured data matching the specified conditional phrase. Conditional phrases can be formed recursively by concatenations of conditional phrases and any logical operators. Classes of conditional phrases which were built into instrumentation blocks included: (1) signals contained by any target processor resources (memory, registers, busses, status indicators, etc.), (2) values of clocks (virtual or actual), and (3) control console inputs (and planned instrumentation processor inputs). The interpretations of the target processor signals depend upon the architectural level of interest. Typical software oriented representations could include: (1) references or stores to specified target processor resources, (2) values of the target machine program counter, and (3) values contained by target resources. When measured, data are treated as higher level representations (i.e., signals as values or addresses) a conditional phrase with a value specification can also contain range specifiers (LT, LE, GE, GT). The conditional processing logic was directly microcoded into the building blocks during the feasibility research; when the interactive instrumentation link becomes operational and interpretive capability is planned.

Fault injection

For development and testing of secure and fault tolerant computing applications, it is desirable to be able to inject abnormal conditions (faults) into the processing stream and then be able to determine the response of the hardware and software. Interpreters with fault injection capabilities were used during the development of several fault tolerant computer systems.[40,41] The same fault injection capability is desired for architectural level emulations of the building block type. The fault injection system is very much the complement of the measurement portion of an instrumentation system. The measurement system asynchronously removes data from the emulation system while the fault injection system asynchronously inserts data (faults) into the emulation. As part of the fault injection capability, it was decided to permit instrumentation results (e.g., instruction counts, and time intervals) to be specified as the triggering event for fault injection. No unusual difficulties were encountered in the implementation of fault injection for emulations. The fault injectors operate as independent modules in the same manner as the instrumentation programs. For the dynamic fault injection routines, the injection trigger is a conditional measurement command whose response is the injection procedure.

An 8080 software version of the error detection and correction algorithm used in the Army Tactical Data Systems was tested on a data path between two 8080's each executing a copy of the program and passing a string of characters back and forth. One, two and three bit errors were introduced and the trace of control through the software and variation in hardware resource utilization under the different error conditions was measured. Hardware faults (stuck on one, stuck on zero, and indeterminate[42]) were then injected into one of the CPU's and the response of the software was followed.

SUMMARY

The application of Instrumented Architectural Level Emulation to computer architecture and computer system development facilities was described. Several efficiency-improving methods were described and experimental implementations performed as part of an Independent Research and Development study were discussed. The results of this study indicate that for a host architecture like that of the QM-1, (1) multilevel emulations of different computer architectures can be constructed from a library of module building blocks and (2) that separate conditional measurement and fault injection programs can operate upon these architectural level emulations.

REFERENCES

1. Burris, H. R., "Computer System Development Facility," submitted to *Proc. International Symposium on Computer Performance Modeling, Measurement, and Evaluation,* IFIP, August 1977.
2. Flynn, M. J., C. Neuhauser and R. M. McClure, "EMMY—An Emulation System for User Microprogramming," *Proc. NCC,* 1975, pp. 85-90.
3. Barr, R. G., J. A. Becker, W. P. Lidinsky, and U. V. Tantillo, "A Research Oriented Dynamic Microprocessor," *IEEE Trans. Comp.* 1973, pp. 976-985.
4. Mattson, R., and A. Salisbury, "The Microprogrammable Multi-Processor-(MMP) System for Simultaneous Emulation of Interoperating Computer Systems," *Seventh Annual Workshop on Microprogramming,* 1974. pp. 290-296.
5. Svobodova, L., and R. Mattson, "The Role of Emulation in Performance Measurement and Evaluation," *Proc. International Symposium on Computer Performance Modeling, Measurement and Evaluation,* Hardvard University, March 1976, pp. 126-135.
6. McClean, R. K., *SPACE DATA SYSTEMS FACILITY,* Final Report, Contract No. F04701-75-C-0194, USAF, Hq. Space & Missile Systems Organization, Los Angeles, Calif., October, 1976.
7. Klayton, A., "Concept for a Computer Architecture Research Facility," *Proc. 1976 International Conference on Parallel Processing,* August 1976, pp. 189-190.
8. Burris, H. R., Trip Report—Microprogramming Applications Within D.A., Memorandum for Record, Office of the Project Manager for Army Tactical Data Systems, Ft. Monmouth, New Jersey, August 1971.
9. Wilkes, M. V., "The Best Way to Design an Automatic Calculating Maching," *Manchester University Computer Inaugral Conference,* 16-18, July 1951.
10. Tucker, S. G., "Emulation of Large Systems," *Comm. ACM,* Vol. 8, Dec. 1965, pp. 753-761.
11. Beach, E. J., and J. Mercurio, "Emulation Capabilities of a Microprogrammable Multi-Processor System," *Proc. Seventh Annual Pittsburgh Conference on Modeling and Simulation,* ISA, April 1976.
12. Salisbury, A. B., "MCF: A Military Computer Facility for Computer-Based Systems," *Computer Architecture News* (SIGARCH) Vol. 5:4, October 1976, pp. 17-20.
13. Bell, C. G. and A. Newell, *Computer Structures: Readings and Examples,* New York, McGraw-Hill, 1971.
14. Burris, H. R., "Development and Application of Time-Scaled Emulations," *Proc. Sixth Annual Pittsburgh Conference on Modeling and Simulation,* ISA, April 1976.
15. Burris, H. R., "A Simulation Method for Selecting Computer Processors for Computer Processor Emulation," *Proc. Sixth Annual Pittsburgh Conference on Modeling and Simulation,* ISA, April 1975, pp. 415-421.
16. Fuller, S. H., et al, "The Effects of Emerging Technology and Emulation Requirements on Microprogramming," *IEEE Trans Comput.,* Vol. C-25 October 1976, pp. 1000-1009.
17. McClean, R. K. and B. Press, "The Flexible Analysis, Simulation and Test Facility: Diagnostic Emulation," *TRW Software Series,* TRW-SS-75-03, October 1975.
18. Ramamoorthy, C. V. and M. Tsuchiya, "A High Level Language for Horizontal Microprogramming," *IEEE Trans. Comput.,* Vol. C-23, August 1974, pp. 791-801.
19. Patterson, D. A. "Strum: Structural Microprogram Development System for Correct Firmware," *IEEE Trans. Comput.* Vol. C-25, October 1976, pp. 974-985.
20. Svobodova, L., "Computer System Measurability," *Computer,* Vol. 9, No. 6, June 1976, pp. 9-17.
21. Svobodova, L., "*Computer Performance Measurement and Evaluation Methods: Analysis and Applications,* New York, Elsevier North-Holland, 1976.
22. Browne, J. C., "An Analysis of Measurement Procedures for Computer Systems," *Performance Evaluation Review (ACM Sigmetrics),* January 1975, pp. 29-32.
23. Boehm, B. W. and T. E. Bell, "Issues in Computer Performance Evaluation: Some Consensus, Some Divergence," *Performance Evaluation Review (ACM Sigmetrics),* July 1975, pp. 4-39.
24. Bennetts, R. G. and R. V. Scott, "Recent Developments in the Theory and Practice of Testable Logic Design," *Computer,* Vol. 9, No. 6, June 1976, pp. 47-63.
25. QM-1, Nanodata Corp., Buffalo, New York.
26. Flynn, M. J. and M. D. MacLaren, "Microprogramming Revised," *ACM National Conference Proceedings,* Vol. 22, Thompson Books, Washington, D.C. 1967, pp. 457-464.
27. Interdata 8/32, Interdata Corp., Oceanport, New Jersey.
28. Genisco Graphic Display System, Genisco Technology Corp., Irvine, Calif.
29. IMSAI 8080, IMS Associates Inc, San Leandro, Calif.
30. Merrill, H. W. B., "A Technique for Comparative Analysis of Kiviat Graphs," *Performance Evaluation Review (ACM Sigmetrics),* March 1974.
31. Merrill, H. W. B., "Further Comments on Comparative Evaluation of Kiviat Graphs," *Performance Evaluation Review (ACM Sigmetrics),* January 1975.
32. Burris, H. R., *Instrumented Architectural Level Emulation: 1976 Project Plan,* TRW Defense and Space Systems Group, Doc. No. 99994-6323-TU-00, March 1976.
33. Burris, H. R., "Microcomputer Implemented NBS Encryption Algorithm," *Proc. Microcomputer '77,* April 1977.
34. Burris, H. R., "Time-Scaled Emulations of the 8080 Microprocessor," *Proc. Seventh Annual Pittsburgh Conference on Modeling and Simulation,* ISA, April, 1977.
35. Jones, L. H., "Instrumentation Sequencing In Microprogrammed Computers," *Proc. NCC,* 1975, pp. 91-98.
36. Belady, L. A., "A Study of Replacement Algorithms for Virtual-Storage Computers," *IBM Systems Journal,* 5(2):, 1966, pp. 78-101.
37. Coffman, E. G. and L. C. Varian, "Further Experimental Data on the Behavior of Programs in a Paging Environment," *Communications of the ACM,* 11(7):, July 1968, pp. 471-474.
38. Chu, W. W. and H. Opderbeck, "The Page Fault Frequency Replacement Algorithm," *Proceedings of the AFIPS 1972 Fall Joint Computer Conference* 41(1):, pp. 597-609.
39. Karnes, R. E. and W. A. Carter, "Computer Design Verification via Software Simulation," *Proc. NCC,* 1975, pp. 847-851.
40. Anderson, J. E. and Macri, F. J., "Multiple Redundancy Applications in a Computer," *Proc. 1967 Annual Symposium on Reliability,* Washington, D.C., January 1967, pp. 553-562.
41. Kuehn, R. E., "Computer Redundancy: Design, Performance, and Future," *IEEE Trans on Reliability,* Vol. R-18, No. 1, pp. 3-11, February 1969.
42. Avizienis, A., "Fault-Tolerant Computing: An Overview," *Computer,* Vol. 4, No. 1, January-February 1971, pp. 5-8.

ARES—A memory, capable of associating stored information through relevancy estimation

by TADAO ICHIKAWA
Kokusai Denshin Denwa Co., Ltd.
Tokyo, Japan

and

KEN SAKAMURA and HIDEO AISO
Keio University
Yokohama, Japan

ABSTRACT

In this paper, a novel principle of association in the field of pattern recognition is presented along with its mechanism. Mutual relevancies of information are estimated in terms of the Lee distance, and association is performed without the direct calculation of the Lee distances. Furthermore, for information composed of several blocks, the relevancy checking is extended from blockwise to interblocks. The number of associated information can be adaptively controlled according to the application. The memory is called ARES. Following the overview of the principle behind ARES is a detailed description of the hardware implementation. In the interest of practicality, the design concept incorporates standard modular LSI devices. The possibility of applying ARES to pattern recognition problems involving on-line character recognition is then considered. ARES is a step towards more sophisticated memories specially designed for advanced pattern recognition.

INTRODUCTION

With the rapid progress of semiconductor technology, a processor based on a content addressable memory and a related ensemble of some processing elements has proven extremely useful in many applications including air traffic control, computer graphics, information retrieval, data base management as well as pattern recognition problems. STARAN,[1] which is considered to be the first practical associative processor ever produced, has shown that the content addressable memory has a place in the solution of those problems.

When we direct our interest to pattern recognition applications, however, there arises the necessity to extend the association capability ordinary content addressable memories have. The association observed in human cognitive behavior is the ability to bring a small set of patterns intuitively to mind from a variety of unconsciously acquired information. These patterns are quite similar to the pattern to be recognized, and recognition is accomplished very effectively using a very limited number of associated patterns.

In this paper, a novel principle of association with its mechanism is given in the field of pattern recognition. Mutual relevancies of information are estimated in terms of the Lee distance, and association is performed without the direct calculation of the Lee distances. Furthermore, for information composed of several blocks, relevancy checking is extended from blockwise to interblocks. This is simply realized with the conventional content addressable memory technology by adopting the error correction principle of coding theory.

The memory is called ARES because of its ability to associate stored information through relevancy estimation. Following the overview of the principle behind ARES is a detailed explanation of the hardware implementation of this mechanism. The configuration is first generally described in terms of modularly specified functions for ease of adapting to a variety of possible applications. The claim of practicality is, however, based on the fact that the design concept for ARES allows the use of standard modular LSI devices that are now widely used in the computer industry. The size and the speed required should be clearly specified at the time of designing. The possibility of applying ARES to some pattern recognition problems is considered regarding the on-line character recognition.

In conclusion, increased performance can be obtained. ARES is a step towards more sophisticated memories for a wide variety of information processing problems in general.

PRINCIPLE OF ASSOCIATION

We assume that the essential features of information are expressed by a symbol sequence of a certain fixed length N where each symbol takes either of the q distinct figures; 0, 1, . . . , $q-1$. A slight deviation on features of the information expressed as above causes the appearance of symbols that most probably take adjacent figures to each other, where $q-1$ is assumed to be adjacent to 0. Therefore, the mutual relevancies of information can be measured in terms of the Lee distance on the sequence of symbols related to them.

The Lee distance $d_L(X, A)$ is defined as follows for the information X and A.

$$X=(x_1, x_2, \ldots, x_i, \ldots, x_N),$$

$$x_i \in \{0, 1, \ldots, q-1\} \text{ for } 1 \leq i \leq N. \quad (1)$$

$$A=(a_1, a_2, \ldots, a_i, \ldots, a_N),$$

$$a_i \in \{0, 1, \ldots, q-1\} \text{ for } 1 \leq i \leq N. \quad (2)$$

$$d_L(X, A)= \sum_{i=1}^{N} \|x_i - a_i\|,$$

$$\|x_i - a_i\| = \min\{x_i - a_i (\text{mod } q), a_i - x_i (\text{mod } q)\}. \quad (3)$$

Suppose that A's are the pieces of information stored in a memory, and X the information applied to the memory. We try to associate a set of A's which are quite similar to X in terms of the Lee distance at a time by X without executing direct calculations of the Lee distances. The principle of our association which is based on coding theory is explained as follows with the help of the schematic diagram given in Figure 1.

When X is given, we apply the error correction procedure, and let the error corrected code vector of X be denoted as IX. Suppose the error distance correctable by the code is t, then some pieces of information stored in the memory as A's are related to X when the circle containing these A's coincides with the imaginary circle around X having IA which is equal to IX, where IA is the error corrected code vectors of A's. Thus, the A's which are similar to X in terms of the Lee distance are associatively read out by X without calculating the Lee distances between them. IX and IA are called the indices for the association, namely, the recalling of A's with X.

The code which is available for the association is a perfect code since, for every piece of possible information which might appear, an error corrected code vector should also be placed as its index inside the circle of radius t which contains the original information in it.

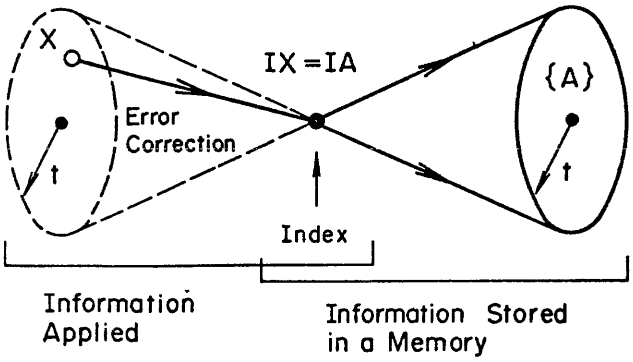


Figure 1—Schematic diagram for explaining association principle

Let n, k and t denote the code length, the number of information digits, and the correctable error distance, respectively.

As the perfect (n, k, t)-codes over q symbol alphabet, the following two are known.

(i) $n=(q^{n-k}-1)/2$ for q odd; single error correctable.
(ii) $q=2t^2+2t+1$, $n=2$, $k=1$; t error correctable.

Hereafter, we will use the class (i) code, because it covers the wide range of n and q which can be properly selected depending on the application.

The error correction of class (i) codes is as follows:[2] A codeword $V=(v_1, v_2, \ldots, v_i, \ldots, v_n)$ with $k=n-1$, and $q=2n+1$ satisfies

$$\sum_{i=1}^{n} i \cdot v_i \equiv 0 \ (\text{mod } q). \quad (4)$$

For a received code vector $V'=(v_1', v_2', \ldots, v_i', \ldots, v_n')$, calculate

$$\sum_{i=1}^{n} i \cdot v_i' = d \ (\text{mod } q) \quad (5)$$

where $-n \leq d \leq n$. If $d=0$, then V' is correctly received with no error superposed. If $d>0$, we change v_d' to $v_d'-1$. If $d<0$, we change $v_{|d|}'$ to $v_{|d|}'+1$.

Received code vectors and error corrected vectors correspond to the original information and their indices, respectively.

ASSOCIATION MECHANISM

As stated before in the preceding section, the length n of the code available for the association is limited, and selected depending on the application in relation to the level q of each symbol.

Suppose $n \leq N$. Then, the information X and A of length N is decomposed into $p(=N/n)$ blocks as expressed in (6) and (7).

$$X=(\bar{X}_1, \bar{X}_2, \ldots, \bar{X}_j, \ldots, \bar{X}_p),$$

$$\bar{X}_j=(x_{n(j-1)+1}, x_{n(j-1)+2}, \ldots, x_{nj}) \text{ for } 1 \leq j \leq p. \quad (6)$$

$$A=(\bar{A}_1, \bar{A}_2, \ldots, \bar{A}_j, \ldots, \bar{A}_p),$$

$$\bar{A}_j=(a_{n(j-1)+1}, a_{n(j-1)+2}, \ldots, a_{nj}) \text{ for } 1 \leq j \leq p. \quad (7)$$

The error correction procedure is applied separately for each block of $\bar{X}_1', \bar{X}_2', \ldots, \bar{X}_k', \ldots, \bar{X}_s'$ which are particularly selected from $\bar{X}_1, \bar{X}_2, \ldots, \bar{X}_p (s \leq p)$. The index IX of X, thus obtained takes the form expressed in (8).

$$IX=(I\bar{X}_1', I\bar{X}_2', \ldots, I\bar{X}_k', \ldots, I\bar{X}_s'), (1 \leq k \leq s). \quad (8)$$

The IA's related to A's are assumed to be already calculated from A's, and stored together with A's in a memory. The adequacy of this assumption will become

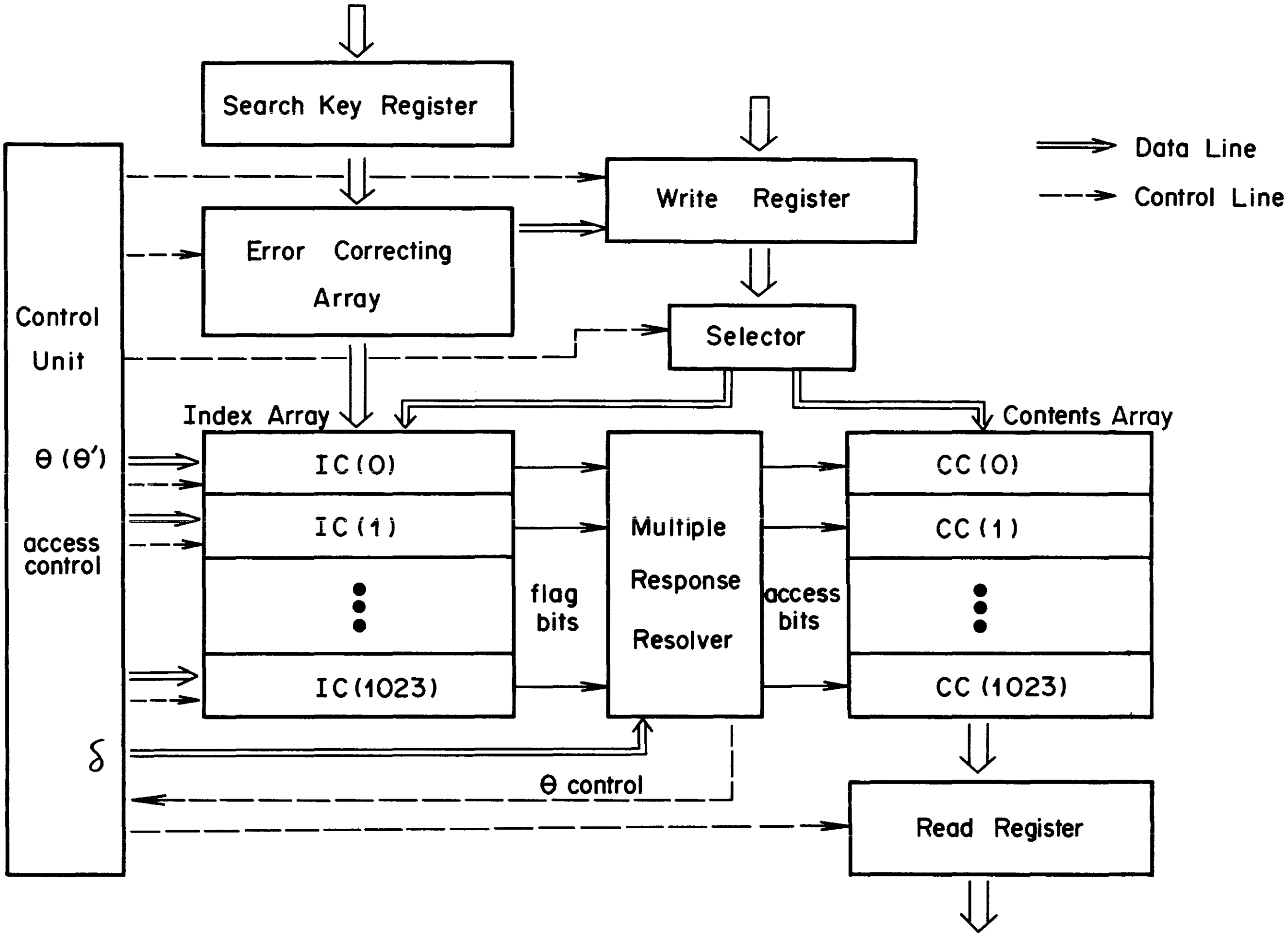


Figure 2—Functional block diagram

clear when we refer to the way of feeding new information into the memory.

When X is applied, the number of index blocks which coincide with each other is checked for all A's in parallel. Let it be denoted as $C(IX, IA)$. The calculation of $C(IX, IA)$ follows to (9), where $d_H(IX, IA)$ denotes the Hamming distance.

$$C(IX, IA)=s-d_H(IX, IA),$$

$$d_H(IX, IA)=\sum_{k=1}^{s} c_k, \qquad \begin{cases} c_k=0 \text{ for } I\bar{X}_k=I\bar{A}_k, \\ c_k=1 \text{ for } I\bar{X}_k\neq I\bar{A}_k. \end{cases} \tag{9}$$

The A's whose $C(IX, IA)$ are equal to or greater than $\theta(0\leq\theta\leq s)$ are associated by X. The θ is called the association parameter. The chance of missing important code vectors at each block association which might be caused by the limited error correction capability ($t=1$) of the class (i)

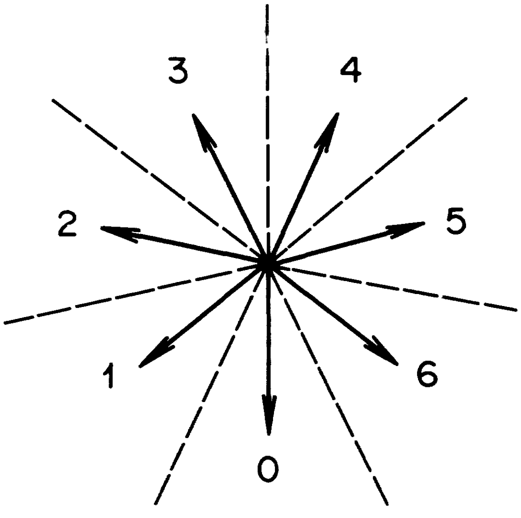


Figure 3—Coding diagram

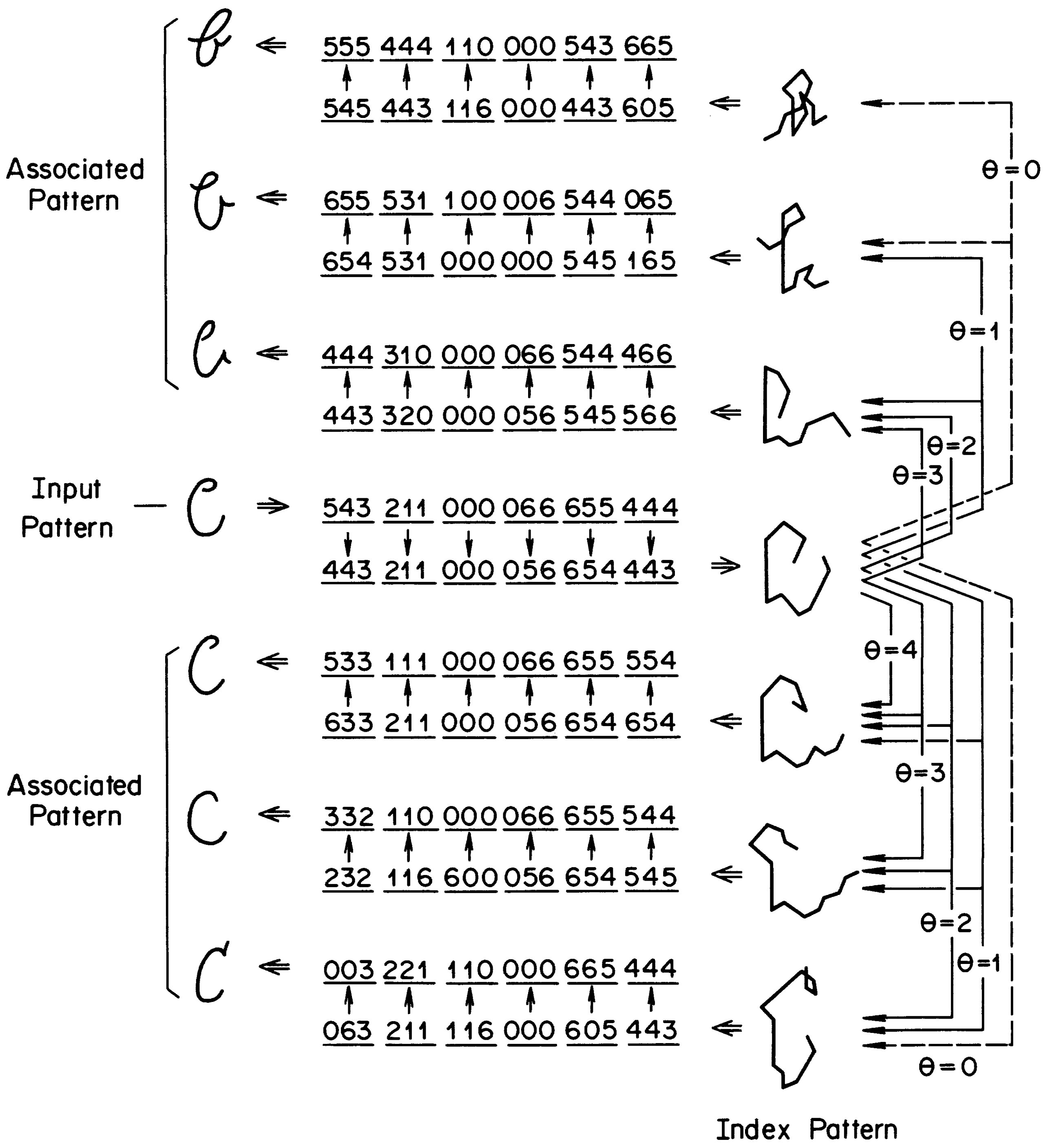


Figure 4—Association control by θ observed at character recognition

code is reduced to some extent by taking θ smaller than s at the index identification over s blocks of information.

The number of the information which are associated with the s and θ selected is specified as $N(s, \theta)$ by (10), where ρ and σ characterize a local distribution of block information around indices and global distribution of block information in the code space, respectively.

$$N(s,\theta)=\binom{s}{\theta}\cdot N(\theta,\theta)-\sum_{\mu=\theta+1}^{s}\binom{\mu-1}{\theta-1}\cdot N(s,\mu),\ (0<\theta<s)$$

$$N(s,s)=(\rho\cdot q^{n-k})^{s}\cdot(\sigma\cdot q^{n})^{p-s}=\rho^{s}\sigma^{p-s}q^{np-ks},\ (0<\theta=s) \qquad (10)$$

The proof is given in Reference 3. The value that $N(s, \theta)$ actually takes differs greatly with application depending on its particular distribution of information in a code space.

The input of newly applied information X is controlled as follows. When A's which satisfy the condition $C(IX, IA) \geq \theta' (0 \leq \theta' \leq s)$ are found, these A's are replaced by X. Otherwise, X is added, unmodified, to the memory. Modification of A's already stored in the memory is also possible by taking the weighted average of X and A, and by feeding it again into the memory. This allows the flexibility of adapting to the continuous use of a recognition machine by particular writers resulting in the increase of recognition rate when the memory is applied, for example, to on-line character recognition systems.

HARDWARE IMPLEMENTATION

In this section, we give an exact description on the hardware implementation of the association mechanism from the architecture technological point of view. Here, (1) each function is modularly structured for ease of adapting to a variety of possible applications, (2) parallelism, and pipeline control schemes are fully adopted to meet the requirement on speed for some possible real-time uses of the memory, and (3) control is distributed so as to save the overcrowding of control lines. This serves to lower the cost of the system. Hereafter, we will refer to this memory as ARES. The functional block diagram is shown in Figure 2.

The control of ARES is explained as follows. For the association of stored information, the input X, which is composed of p blocks of the class (i) code, is fed into the error correcting array through the Search Key Register.

At the Error Correcting Array, the IX is derived from X by applying an error correction procedure to each block of X, separately. Consequently, IX is composed of p blocks of error corrected code vectors, and is called the index of X for the association.

On the other hand, the information A's are stored in the Contents Array, and the indices IA's of A's which, it is assumed, are previously obtained are kept in the index array.

The Index Array is essentially to check the blockwise coincidence of IX with IA's, but, it assumes its most important role at our association, and characterizes ARES. The number of blocks coincided with each other is counted over $s(\leq p)$ blocks unmasked for all IA's stored in the memory, and compared with the θ properly selected. When it is equal to or greater than θ, a flag is set at the output of the index cell which contains the corresponding IA, and the corresponding information in the contents array is selected as a candidate for the information to be associated. But, for most practical applications, there will be a limit δ on the number of associated information, and θ is adaptively selected at every step of the association through trial and error. This requires ARES to have a very high speed processing of index identification with probably a larger number of modules than ordinary content addressable memories have. We will take all bits in parallel comparison at index identification.

The number of flags appearing at the outputs of index cells is counted and compared with δ at the Multiple Response Resolver for the control of θ. If the number of flags is greater than the limit δ on the information to be associated, θ is increased by one. Otherwise, it is lowered by one. Thus, θ is heuristically selected so that a greater number of flags as possible are obtained within the limit δ, and then the contents array is accessed by the flag bits.

The input of newly applied information is controlled by θ' as explained in the preceding section, and is transferred to the index and contents array through the Write Regiser.

ARES FOR CHARACTER RECOGNITION

The configuration, which is generally described in the preceding section, strongly depends on the application. The size and the speed required should be clearly specified at design. Some alternatives are provided by compromising cost and speed. We will show, as an example, ARES applicable to the on-line character recognition.

The essential feature of each character is coded into a symbol sequence by tracing the directions of the stylus movements following the diagram in Figure 3. The length of the coded symbol sequence thus obtained is normalized into $N=18$, and this is divided into $p(=N/n)=6$ blocks so that the perfect (3, 2, 1)-code of $n=3$ and $q=7$ is used for the association. The normalization is governed by local rules which are applied to the symbols in a sequence simultaneously in parallel.

When an input pattern is applied, a set of patterns is associated through index identification with the θ properly selected. Figure 4 explains how the association is controlled

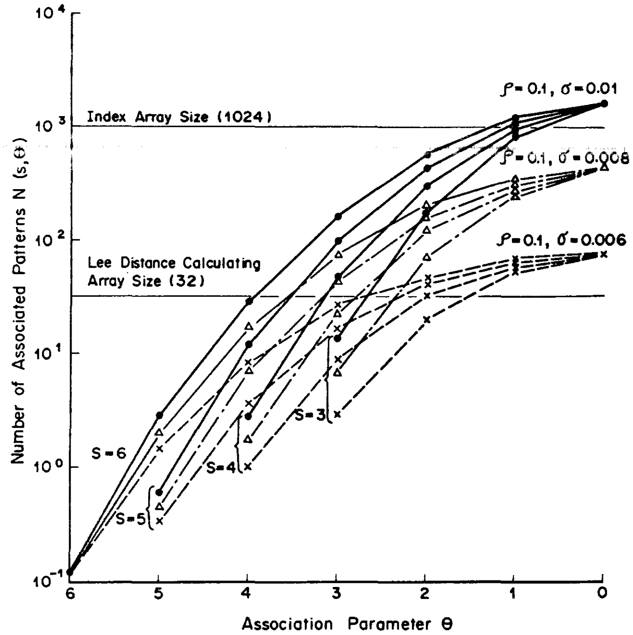


Figure 5—Number of associated patterns $N(s, \theta)$, $s \leq p = 6$

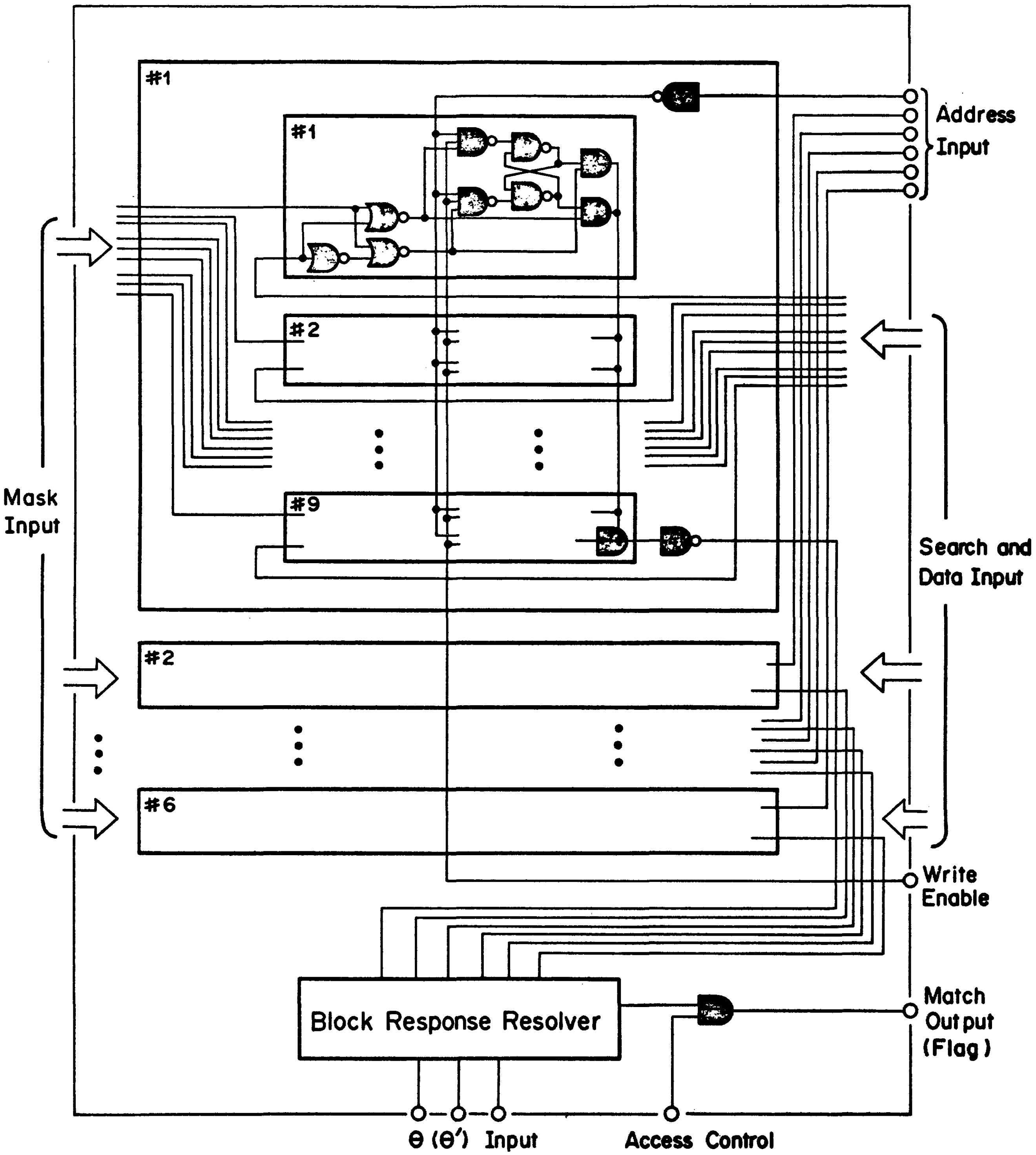


Figure 6—Index cell

by θ in the case of $p=s$. When $\theta=0$, there occurs no association capability. Imaginary visible representations of index patterns are given at the right side of the figure. The associated patterns are necessarily the patterns which are most helpful in identifying the input pattern to be recognized, and detailed checking of the similarity is carried out concerning this limited number of associated patterns. The preliminary experimental results on handprinted character recognition obtained through computer simulation are given in Reference 4.

Figure 5 shows the number of associated patterns $N(s, \theta)$ estimated by (10) for some possible information sources specified by $q=7$ and $N=18$. It is observed that around 30 patterns are associatively selected with $\theta=3\sim5$ from the $100\sim1000$ patterns stored in the memory. Actually, the $N(s, \theta)$ differs greatly for each pattern to be recognized, and ARES for character recognition is designed to select θ adaptively for each so that the logic array provided for the Lee distance calculations of 32 ($=\delta$) patterns at most is filled up with as many associated patterns as possible.

The above observation gives the exact design of the index cell. The index array is composed of index cells of up to 1024. The diagram of an index cell is shown in Figure 6. As the comparator, a conventional content addressable

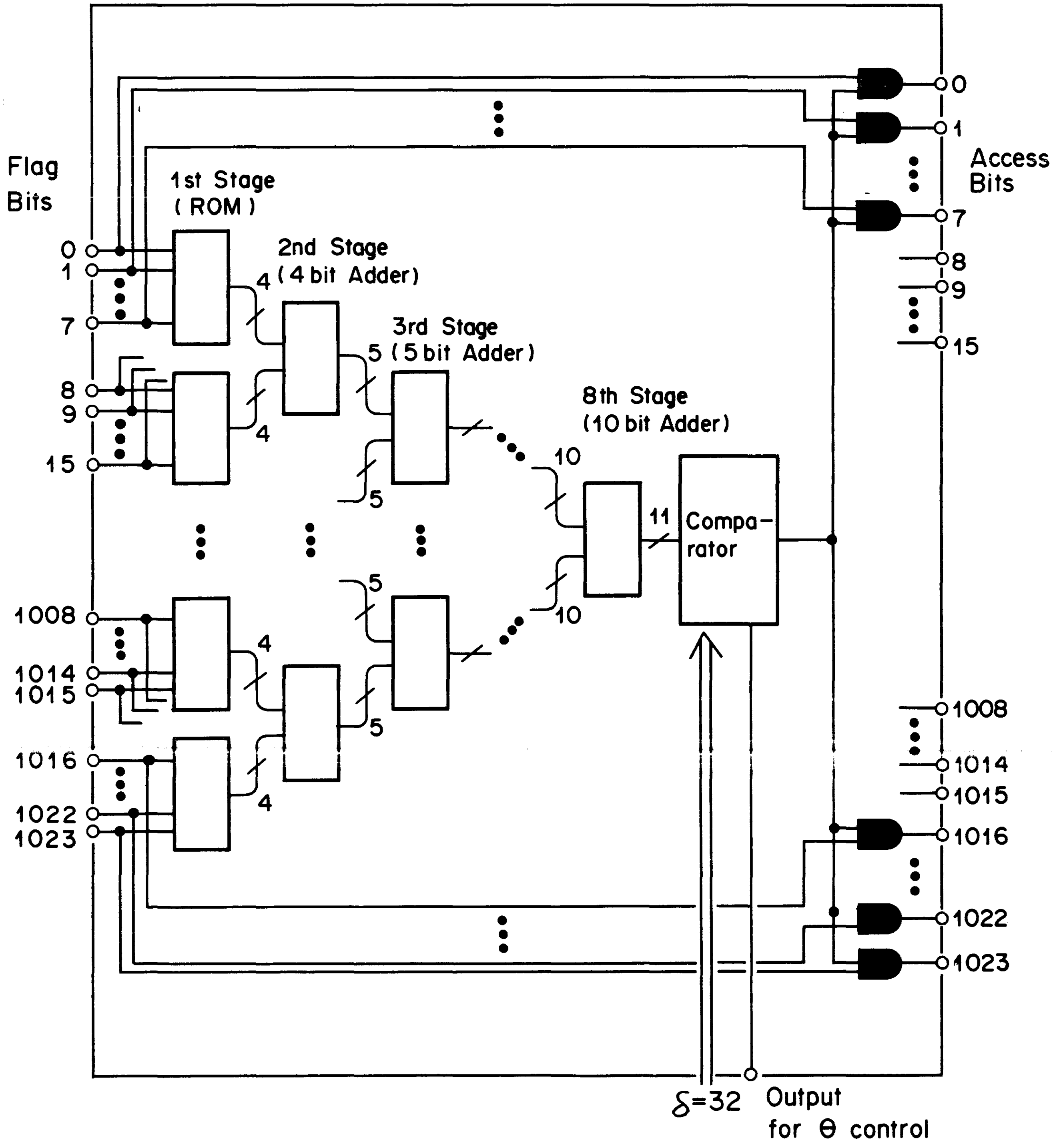


Figure 7—Multiple Response Resolver

memory is applicable. The number of index blocks which are identified by the index blocks of applied information is compared with θ (or θ') at the Block Response Resolver.

There are three alternatives in the design of the Multiple Response Resolver. These are: (1) serial counting by a shift register, (2) table look-up by ROM, and (3) counting by adders. As a compromise between the cost and speed, we apply a mix of (2) and (3) resulting in 50 percent decrease of the required components with a slight increase on speed (300 to 350*ns*) compared with (3). The diagram is given in Figure 7.

The execution time attainable is less than 1 μsec for the association with heuristic searching of θ.

Some possible ways to economize ARES especially for on-line recognition are: (1) adoption of bit serial processing at content addressing, (2) simplification of the index cell structure and common cell utilization on a time-sharing basis, (3) sequential controlling throughout the system, (4) use of cheaper devices, and (5) partial or entire simulation by microprocessors.

CONCLUSION

An association mechanism is presented in the field of pattern recognition; characteristic features are as follows:

- Pieces of information are stored in a memory together with their indices obtained by the application of an error correction procedure.
- Mutual relevancies of information defined in terms of the Lee distance are estimated simply by checking blockwise coincidence between indices without the direct calculation of Lee distances.
- When the number of coinciding index blocks exceeds the threshold θ, the information corresponding to the index is associated, and the number of associated information is well controlled by adapting θ to the externally given condition.

The hardware implementation of the proposed mechanism is described; its main characteristics are as follows:

- ARES is modularly structured, and its control algorithm is realized on microprogrammed logic or PLA (programmed logic arrays). This makes the system extension easier.
- Parallelism and pipeline control schemes are fully adopted throughout the system to meet the requirement for speed on some possible real-time uses of ARES.
- By sacrificing speed, depending on its application, the structure of ARES can be easily simplified.

In conclusion, ARES is a step towards more sophisticated memories specially designed for advanced pattern recognition applications. The effectiveness of the association will become more obvious when a large scale ARES is shared with a number of small sized recognition logic arrays. Moreover, it is expected that ARES will offer significant advantages as an intelligent and adaptive memory for improving the execution performance of high level language machines. Error correction systems can be properly selected depending on the application.

REFERENCES

1. Rudolph, J. A., "A Production Implementation of an Associative Array Processor—STARAN," *Proc. FJCC 1972,* pp. 229-241.
2. Golomb, S. W. and L. R. Welch, "Algebraic Coding and the Lee Metric," in *Error Correcting Codes,* edited by H. B. Mann, John Wiley & Sons, 1968, p. 181.
3. Ichikawa, T., "Studies on the Application of Close-Packed Codes in the Lee Metric," *KDD Research Report,* No. 84, July 1975, pp. 30-31.
4. Ichikawa, T. and J. Yoshida, "On-Line Recognition of Handprinted Characters with Associative Read-Out of Patterns in a Memory," *Proc. of Second Int'l Joint Conference on Pattern Recognition,* August 1974, pp. 206-207.

Cache memory systems for multiprocessor architecture

by O. P. AGRAWAL* and A. V. POHM
Iowa State University
Ames, Iowa

ABSTRACT

The performances of two types of multiprocessor systems with cache memories dedicated to each processor are analyzed. It is demonstrated that by appropriate cache system design, adequate memory system speed can be achieved to keep the processors busy. A write through algorithm is used for each cache to minimize directory searching and several main memory modules are used to provide interleaved write. In large memories a cost performance analysis shows that with an increase in per bit costs of 5 to 20 percent, the memory throughput can be enhanced by a factor of 10 and by a factor of 3 or more over simple interleaving of the modules for random memory requests. Experimental evidence indicates smaller cache memories are required for dedicated processors than for standard processors. All memories and buses can be of modest speed.

INTRODUCTION

In this paper, buffered or cache memory organizations suitable for multiprocessor systems employing a set of identical processors or individualized dedicated processors are examined. The intent of the analysis is to show organizations which have adequate performance, low incremental costs, and simple logical and electrical requirements. Such an analysis is particularly pertinent in terms of the rapid microprocessor development which is occurring.

These recent advances in solid state technology coupled with the consistent decline in the cost of hardware and persistent increase in the cost of software, have provided the impetus to reexamine the traditional hardware/software boundary. Conventional machines and other early stored program computers were constrained to a single instruction stream, a single data stream and to sequential organizations primarily because of economic considerations. Recently, highly hardware oriented computing structures with radical departure in architectural organization such as the SYMBOL-2R computer[1,2] designed by Fairchild and now operating at Iowa State University, Ames, have appeared. In addition, the arrival of cheap, flexible but quite powerful, bipolar microprogrammable bit slice microprocessor chips along with low cost, high speed, compact memory chips are making the organization of large computers potentially more modular. A collection of homogeneous or heterogeneous microprocessors with either common or localized memories can form a cost effective system.

In regard to changing architecture, Winsley[3] states that electronic disks will cause radical changes to occur in computer architectures. Three major impact areas cited by him are:

- Time-sharing computing systems with mini-processor units attached to a shared large central memory (Figure 1);
- Stand alone minicomputers with large memories;
- Systems in which main memory is treated as a small cache with the main data storage being in the electronic disk.

In the analysis conducted in this paper, the properties of the new electronic disks have not been specifically included. However, the analysis can be easily extended to include these memories by choosing appropriate parameters for the main memory.

One of the basic system architectures assumed is similar to that of SYMBOL-2R computer with dedicated processors working out of a common memory. Figure 2. The other organization assumes a collection of general purposes processors working out of a common memory. Figure 1.

Building individual processors from basic bit-slice bus transceivers, micro sequencers and microprocessor chips imparts a highly flexible and modular nature to the basic architecture of such systems. Basic building block features of the microprocessor chips can allow all processors to appear homogeneous as far as their bus interfaces are concerned. Particular heterogeneity features can be imparted by microprogramming. Microprogramming enables one to tailor any microprocessor for any particular dedicated function without changing any hardware. Provision of a small amount of cache memory for each dedicated microprocessor improves tremendously the performance of the system[22] and reduces the contention for global memory.

* Now at Rockwell International, Cedar Rapids, Iowa.

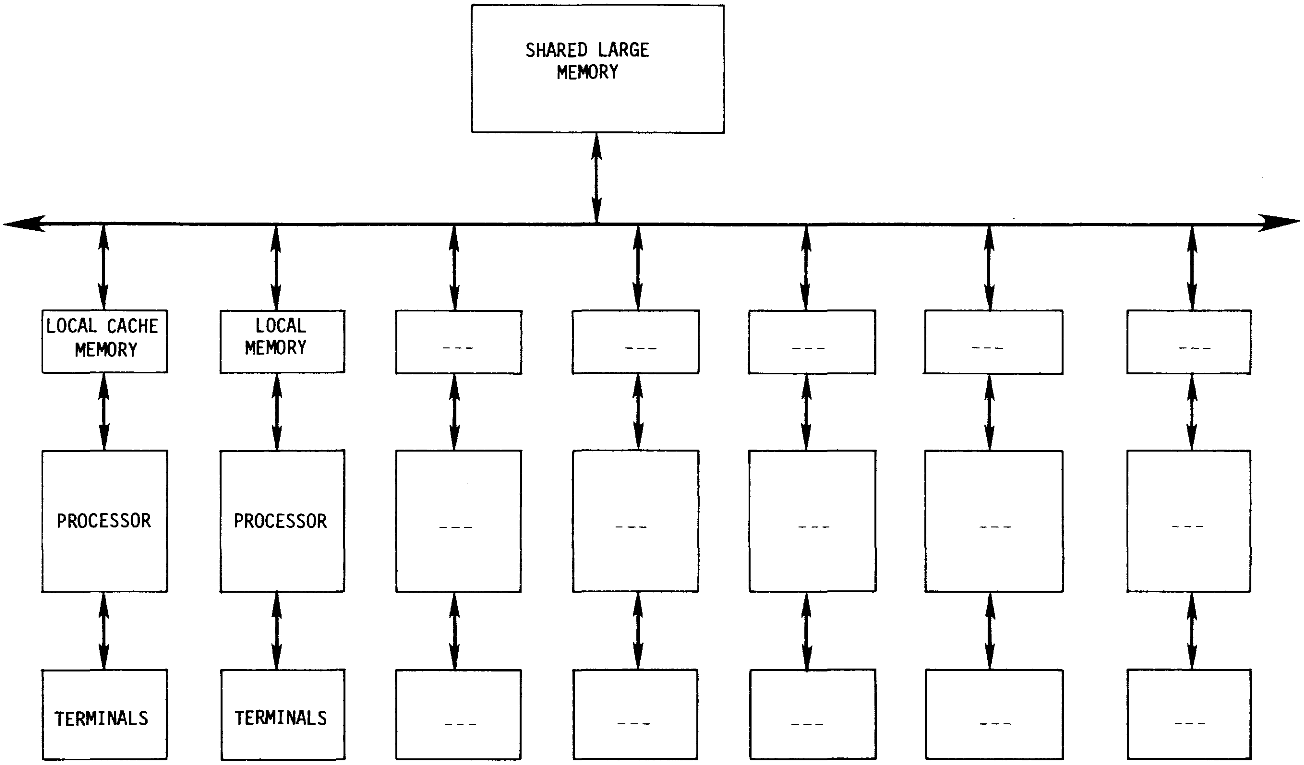


Figure 1—Multiprocessor system using standard processors

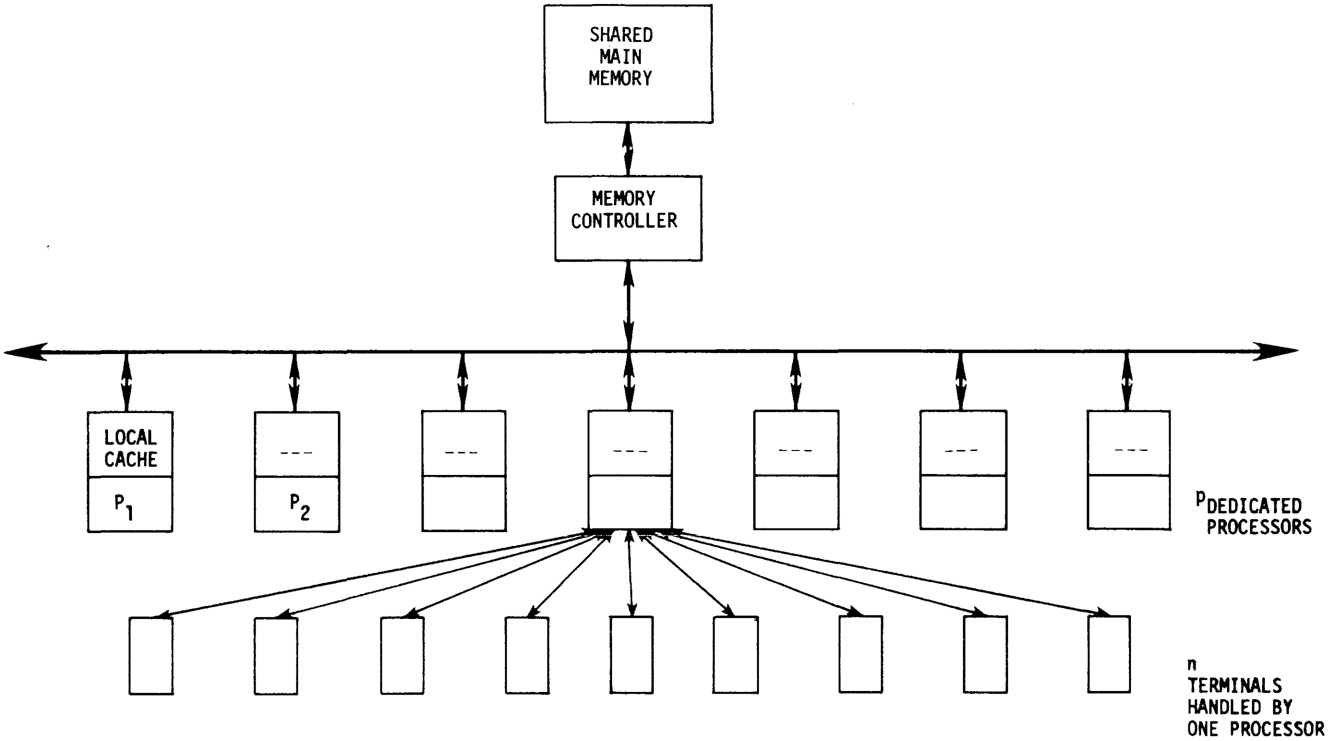


Figure 2—Dedicated processor arrangement

A sample basic architectuer for any particular processor is shown in Figure 3.

In most large systems, memory speed is a limiting throughput factor so that a collection of processors most likely would aggravate the problem unless memory enhancement techniques are employed.

For the systems analyzed, a buffer is assigned to each processor to enhance effective memory performance. To alleviate the need for directory searching in all buffers a write through algorithm is assumed to maintain valid information in the main store. Further, in order to enhance performance, the main memory also is assumed to be multi module. As noted below, numerous buffer organizations are possible.

BUFFERED MEMORIES

The use of buffered memories for conventional and unconventional computing structures has been the subject of numerous investigations over the past decade.[4–21] As early as 1962, Bloom et al.,[4] first proposed the use of a small associatively tagged memory to enhance memory performance and termed it a look-aside memory. The memory as discussed by Lee[5] and Wilkes[6] described a "slave memory" to store the most recently used instructions in a direct mapping mode. Gibson[7] described a buffered memory system in which multiword blocks were transferred and he reported the effect of various replacement strategies on the percentage of words not found in the buffer. Kaplan and Wind son,[14] Meade,[11] Liptay,[8] Bell et al.,[13,15] and Mattson,[11,12] have conducted extensive studies on the effects of buffer size and block size on the miss ration for a variety of computing environments. They have also reported on the relationship between the number of classes and the hit ratio.[11] Pohm et al.,[16–19] have also demonstrated the effectiveness of a parallel buffered memories for conventional machines. Agrawal et al.,[22] have demonstrated the applicability of cache memories to highly unconventional computing structures like SYMBOL-2R.

Effective cycle computation

One of the most important figures of merit of a buffered memory system is the effective cycle time. Effective cycle time, by definition, is the average time required by the whole memory system to provide a memory word. Obviously the primary goal of the buffered memory system is to provide operand to the processors at a rate sufficient to keep them continuously busy.

Various parameters which affect the effective cycle time of a buffered memory system are as follows:[17]

- Hit Ratio (HR)

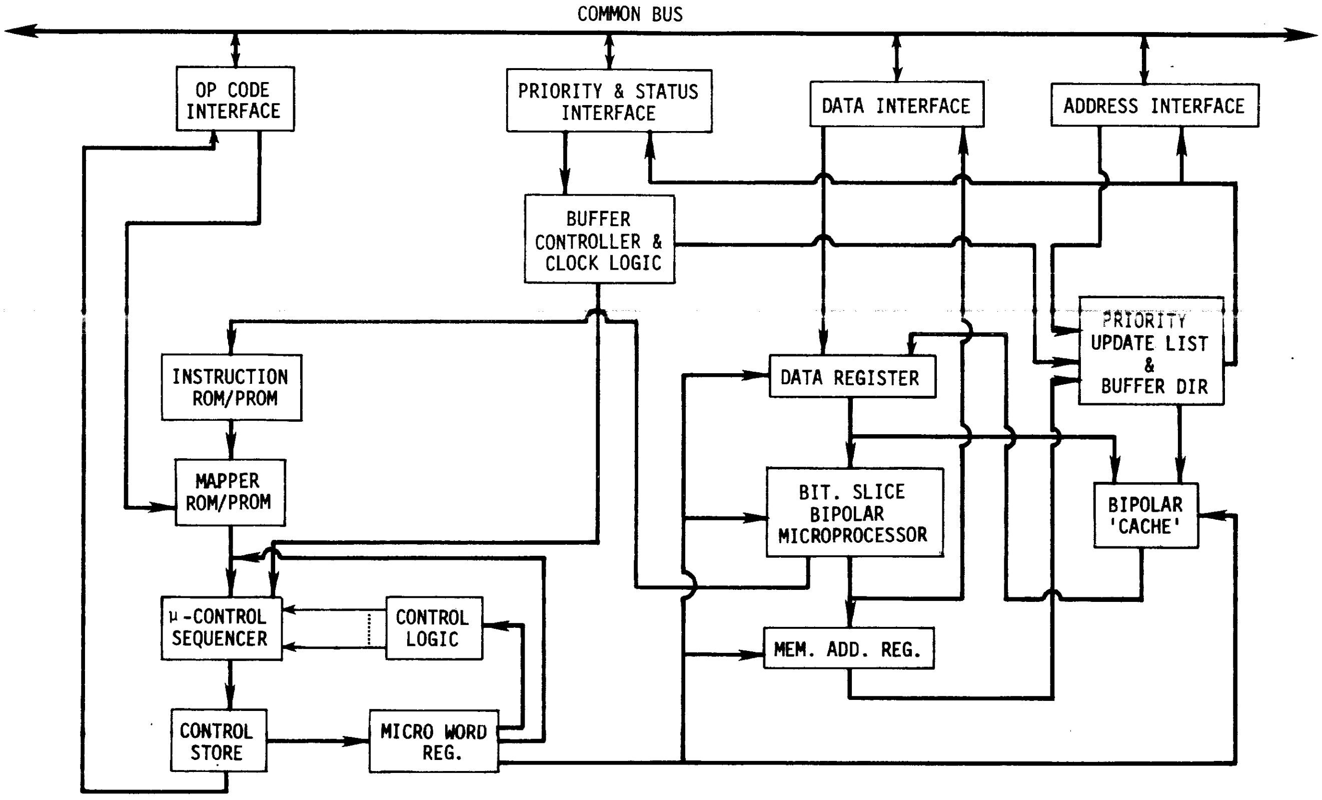


Figure 3—Block diagram of each dedicated processor

- Fraction of Read/Write ration (FR)
- Type of control algorithms
- Number of main memory modules (Block size) and how they are started.
- Effective delay or wait time for a particular module.
- Effective or average delay for all memory modules.

The performance of a multi-module buffered memory system has been evaluated for two algorithms. Write through and flagged, registered swap[18] assuming random write requests, misses and flagged words. The following notations are used.[17]

The Flagged, registered swap algorithm is generally the fastest[16,17] and is compared with the write through algorithm proposed for the system.

FR	Fraction of reads (relative to total number of memory accesses)
HR	Hit ratio
X	Fraction of buffer words written after loading from main
TSRC	Main memory read cycle
TSRA	Main memory read access
TFRC	Buffer read cycle
TFRA	Buffer read access
TDFR	Data transfer from main to buffer
TFWC	Buffer write cycle
TSRH	Directory search time
DF	Delay waiting for main memory to complete previous operation
DS	Delay waiting for a particular memory module to complete previous operation.

Effective cycle time computations for various writing algorithms are computed as follows

Write through

$$\begin{aligned}
t_{WT}=&\text{FR.HR.TFRC}\\
&+(1-\text{HR})\ \text{FR}\ (\text{TSRH}+\text{RSRA}+\text{DFA}+\text{N}*\text{TDFR})\\
&+(1-\text{FR})(\text{TFWC OR DS}+\underset{\substack{\text{of main}\\ \text{mod}}}{\text{START.TIME}})
\end{aligned}$$

which ever is greater

$$\begin{aligned}
\text{DS}=&\text{DSR}+\text{DSW}\\
&=(1-\text{HR})\text{FR}(\text{TSRC}-\text{TSRA}-((\text{N}+1)/2).\text{TDFR})\\
&+(1-\text{HR})\text{FR}(\text{TSRC}-\text{TSRA}-\text{NNN}+1)/2).\text{TDFR}+\text{TFRC}).(\text{FR.HR})\\
&+(1-\text{HR})\text{FR}(\text{FR.HR})^2(\text{TSRC}-\text{TSRA}-((\text{N}+1)/2)\ \text{TDFR}-2.\text{TFRC})\\
&+\ldots\\
&+((1-\text{FR})/\text{N})\ (\text{TSRC}-\text{TFWC})\\
&+((1-\text{FR})/\text{N})\ (\text{HR.FR})(\text{TSRC}-\text{TFWC}-\text{TFRC})\\
&+((1-\text{FR})/\text{N}))\ (\text{HR.FR})^2\ (\text{TSRC}\ -\text{TFWC}-2.\text{TFRC})\\
&+\ldots\\
&+((1-\text{FR})/\text{N})\ ((1-\text{FR}(\text{N}-1)/\text{N})(\text{TSRC}-\text{TFWC}-\text{TFWC})\\
&+((1-\text{FR})/\text{N})((1-\text{FR})(\text{N}-1)/\text{N})^2(\text{TSRC}-\text{TFWC}-2.\text{TFWC})
\end{aligned}$$

Flagged Register Swap

$$\begin{aligned}
t_{FRS}=&\quad \text{FR. HR. TFRC}\\
&+(1-\text{HR})\ \text{FR}\ (\text{TSRH}+\text{TSRA}+\text{DFD}+100+\text{N}*\text{TDFR})\\
&+(1-\text{FR})\ \text{H. TFWC}\\
&+(1-\text{FR})\ (1-\text{HR})\ (\text{TSRH}+\text{TSRA}+\text{DFD}+\text{ln}+(\text{N}-1).\text{TDFR})\\
\text{DFD}=&\text{DFDR}+\text{DFDW}\\
\text{DFDR}=&\text{DFCR}\\
\text{DFDW}=&\\
&+(1-\text{FR})\ (1-\text{HR})\times(\text{TSRC}-\text{TFWC})\\
&+(1-\text{fr})\ (1-\text{HR})\times(\text{TSRC}-\text{TFWC}-\text{TFWC}).\text{HR}\\
&+\ldots\\
&+(1-\text{FR})\ (1-\text{HR})\ .\ (1-\text{X})\ (\text{TSRC}-\text{TFWC})\\
&+(1-\text{FR})\ .\ (1-\text{X})\ (\text{TSRC}-\text{TFWC}-\text{TFWC}).\text{HR}\\
&+(1-\text{FR})\ (1-\text{HR})\ (1-\text{X})\ (\text{TSRC}-\text{TFWC}-2.\text{TFCW})\ .\ \text{HR}^2\\
&+\ldots
\end{aligned}$$

(Assuming TFRC=TFWC)

Figures 4 to 7 illustrate the effect of the two writing algorithms on the effective cycle time for the memory systems for the variation of HR with fixed fraction of read/write ratio, for different block sizes and different speed discrepancy factors. This speed discrepancy factor is defined as the ration of main memory cycle time to buffer cycle time. These figures illustrate that, for smaller speed discrepancy factor, WT algorithm tends to give a similar performance to the FRS algorithm for all block sizes; however, as the speed discrepancy increases the FRS algorithm tends to give better performance. Because several buffers would be expected to operate in parallel, the effective speed descrepancy factor would be large.

Figures 8, 9, and 10 illustrate effective cycle time variation for the case where the block size is 8 words and the main memory is interleaved 8 ways. The effective cycle time is plotted in units of TFRC rather than some absolute numbers. Without considering the basic processor cycle time, if just the effectiveness of buffer memory itself is considered, then we see that even for a lower hit-ratio the performance improvement is attractive. The effectiveness of buffer increases with an increase in the speed discrepancy factor. Figure 8 illustrates the case for example of a 200 ns buffer with a 800 ns backing store and Figure 10 illustrates the case of a 50 ns buffer with 800 ns buffer. If for example eight processors requiring a memory word every 0.5 μsec are operated in parallel, an effective memory system cycle time of .0625 μsec is required to provide operands at a rate which keeps all processors busy.

Figure 10, for example would correspond to the case of eight processor, with 0.4 μsec buffers operating into a common 0.8 μsec main memory. Note a hit ratio of .94 would give an effective cycle time of 90 nanoseconds and a hit ratio of .98 would give an effective cycle time of .65 nanoseconds. In the case of identical processors working on different problems of a normal mix, Mattson's[11,12] and other experimental data indicate a buffer of about 8192 bytes for each processor would provide a hit ratio of .98.

It has been shown[20] that for highly dedicated processor systems a judicious provision of 2K bytes of buffer only, is

NUMBER OF MODULES = 4

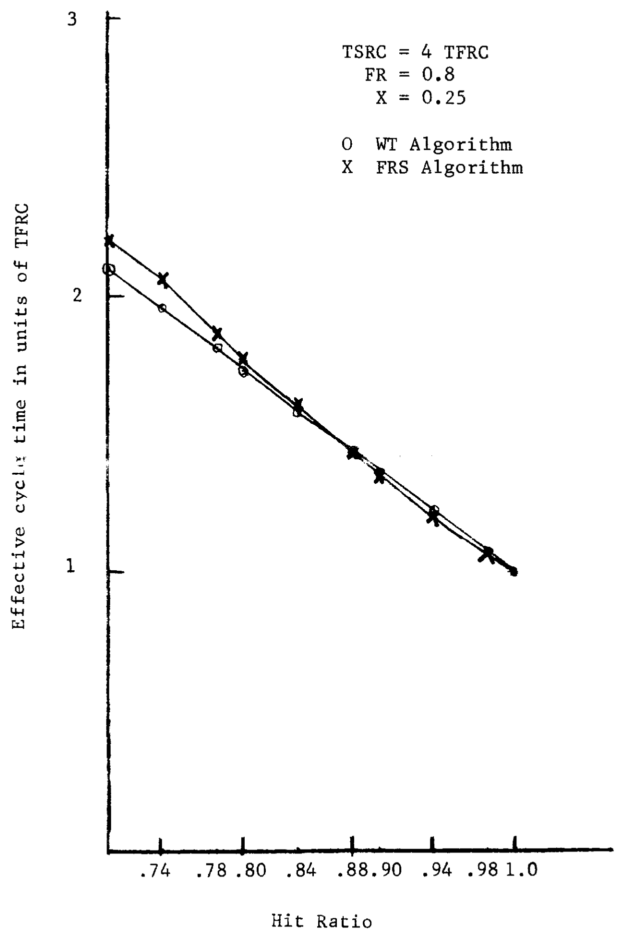


Figure 4—Speed vs. hit ratio

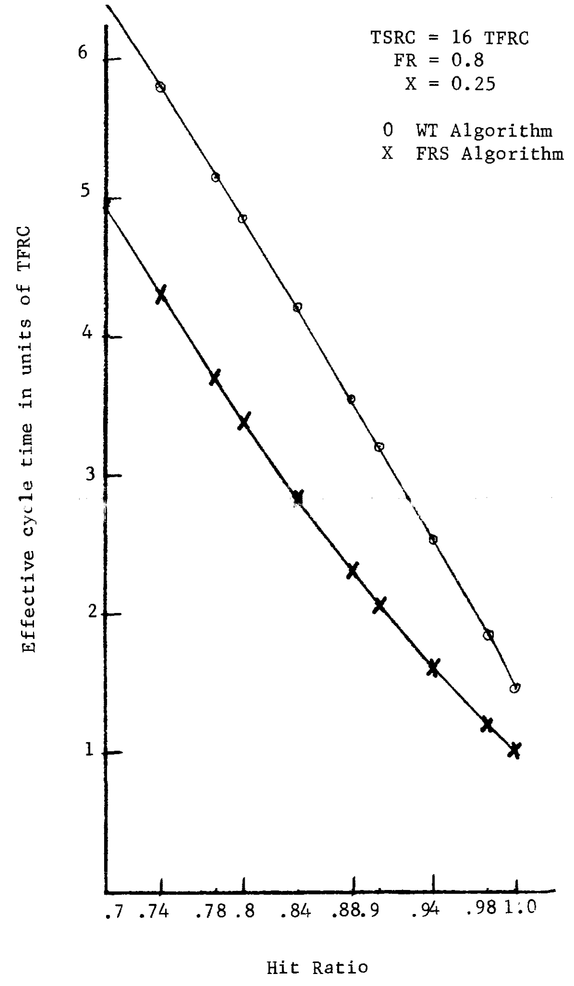


Figure 5—Speed vs. hit ratio

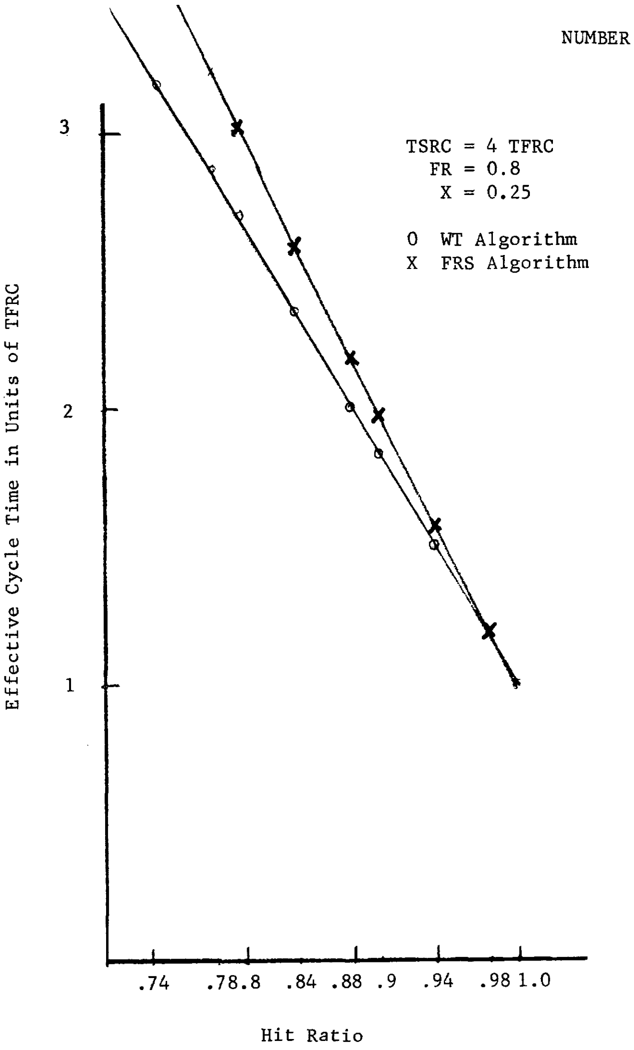


Figure 6—Speed vs. hit ratio

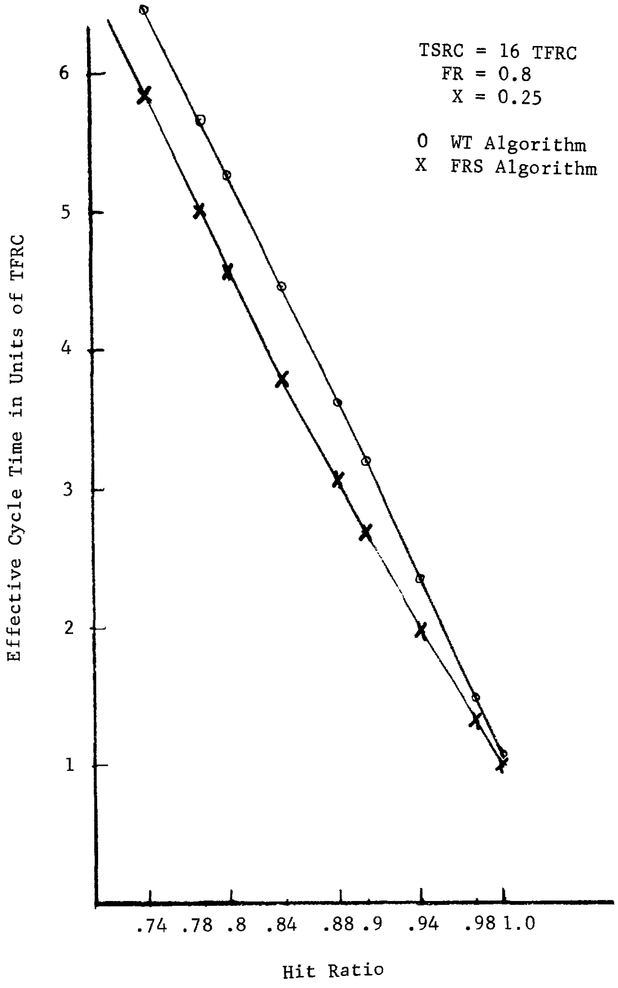


Figure 7—Speed vs. hit ratio

enough to achieve a hit ratio of 95-97 percent, thus improving the effective speed of the whole system. Thus it appears dedicated processors make more effective use of buffer space. This appears reasonable because code required to do the dedicated task would tend to remain resident.

ECONOMIC ANALYSIS

An economic analysis is given in Table I for a variety of buffered memory systems to illustrate the economic effectiveness. The costs which are assumed for the analysis are given in Table I and represent 1976 typical OEM price levels. The example computed fits the worst case in which identical processors requiring larger buffers are used.

Cost analysis

Let

N	= number of main memory modules
M	= number of pages/module
B	= number of blocks/page
W	= number of words/block
n	= number of blocks in the buffer
f	= number of flag bits/buffer directory word
s	= number of bits/word
B_{cost}	= Buffer cost/bit
M_{cost}	= Main memory cost/bit
P_{cost}	= Priority update list cost/bit
D_{cost}	= Directory cost/bit

TABLE I—Memory System Costs

Main Memory	0.3¢ (0.8 pe sec)
Buffer Cost	1.5¢ (400 ns)
Priority Update List	5¢/Bit
Directory	5¢/Bit
Buffer Controller	$400

Main memory		Buffer							
No. of Modules=8		1024 Blocks 65,536 Bytes		512 Blocks 32,768 Bytes		128 Blocks =8.192 Bytes		256 Blocks =16,384 Bytes	
Total Cap	Cost	Cost	% of Main	Cost	% of Main	Cost	% of Main	Cost	% of Main
512K Bytes	$12,582					$1617	12.85	$2829	22.5
1,024K Bytes	$25,165	9556	38%	4828	19%	$1630	6.43%	$2855	11.25%
2,048K Bytes	$50,331	9700	19.2%	4745	9.5%	$1643	3.22%	$2881	5.6%
4,096K Bytes	$100,662	9800	9.7%	4800	4.8%	$1656	1.61%	$2907	2.8

Using these notations
Main Memory Cost=M_{cost}(N. M. B. W. S)
Buffer Cost =Buffer memory cost
+Buffer directory cost
+Priority update list cost

All of the sample computations were made assuming modules large enough for large computer application. The basic word lengths is 64 bits and the virtual memory is organized to be as big as 16 million pages (with each page being 2K bytes). The main store modules are assumed to

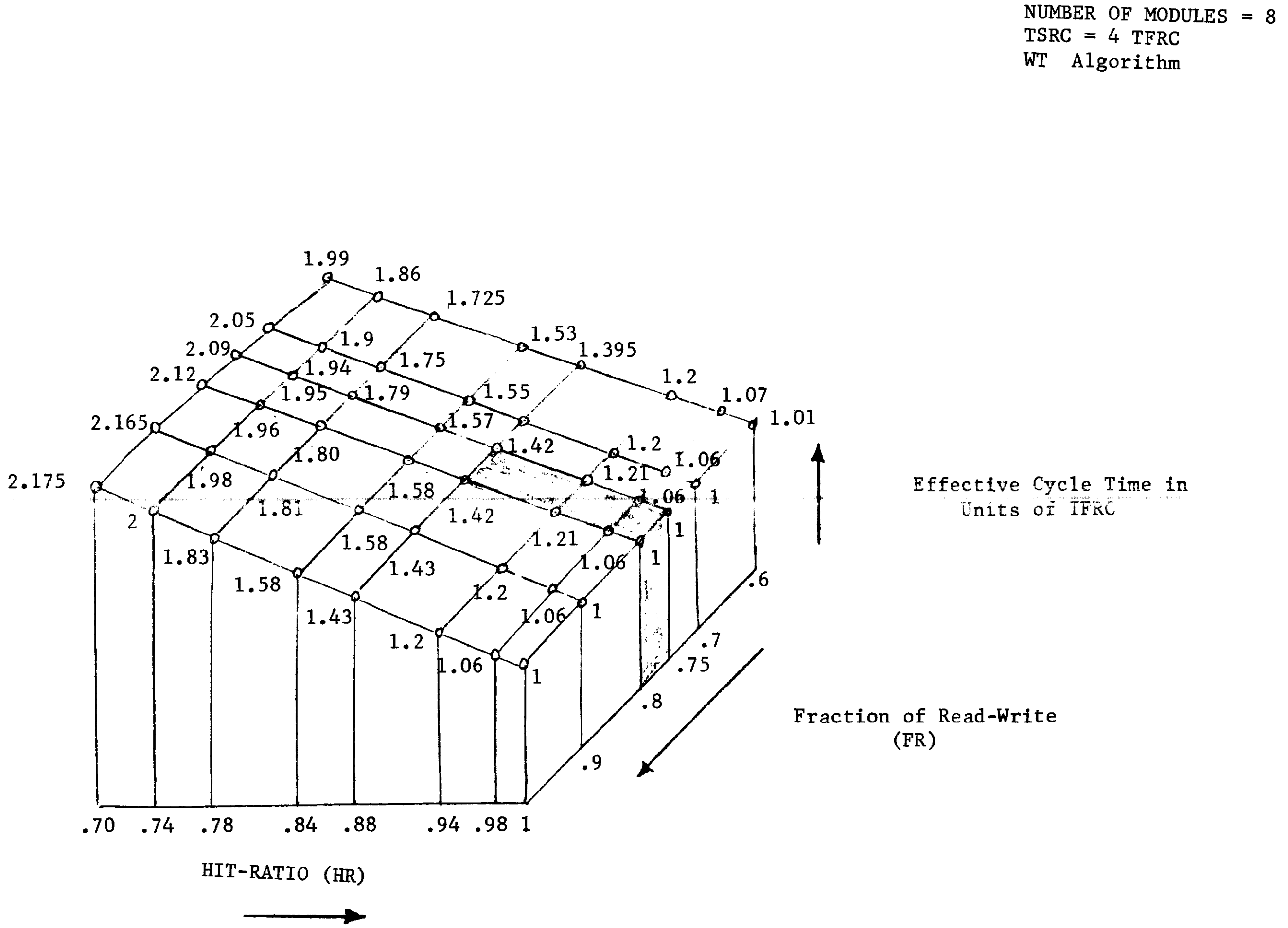


Figure 8

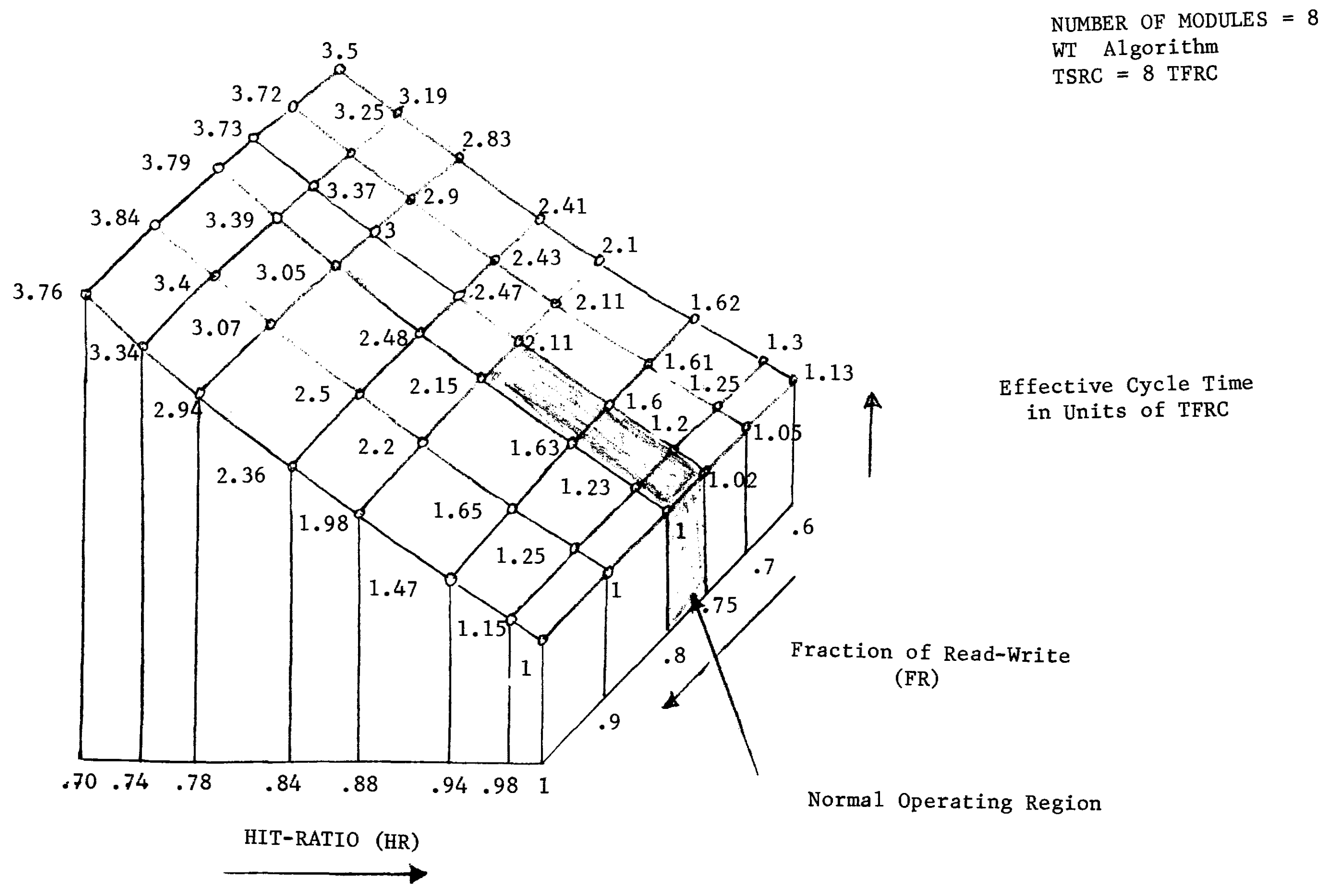


Figure 9

have .8 μsec. cycle time. The total number of blocks in the buffer of a particular dedicated processor is highly dependent on the particular processor, its behavior and the duration of its use by any particular job. However, the cost computation is carried out with 8 main memory modules when 8 modules are interleaved, it is assumed that about 3 modules can be kept busy continuously.

The cost analysis was carried out for a multiclass cache organization as well. Buffer cost is a function of the size of the buffer directory and update list and directory cost is a function of the type of address translation schemes. The major cost of the buffer memory is the memory price itself. Directory update list and switching network typically represent less than 50 percent of the cost. As shown in Table I, adequately sized buffers can be supplied to each processor for modest increases in the cost per bit for a memory system. Main memory can be of modest performance and each individual buffer can be of modest performance which reduces the cost of the memories.

CONCLUSIONS

Multiprocessor systems consisting of a collection of standard processors or dedicated processors can employ a cache memory for each processor to achieve the required memory system performance.

By use of a write through algorithm, an updated copy of all information is maintained in the main memory and multiple directory searching is not necessary. Simultaneous searching in a group of buffer memories would require extreme speed and would require all buffers to be located physically together to limit propagation delays.

In the memory system, discussed both the main memory and the cache memories can be of modest performance. In the design example shown, the main memory was assumed to have a cycle time of 0.8 μsec and each of the eight caches have a cycle time of .4 μsec. In terms of memory throughput, the arrangement provides 10 times the memory performance assuming worst case cache memories sizes of 4096 to 8192 bytes each.

For main memories of one to four million bytes in an eight processor system, the memory system cost/bit is increased from 20 to 5 percent respectively over that of a single 0.8 μsec main memory in the worst case.

Experimental evidence indicates much smaller cache memories can be used with systems with dedicated processors than with systems with general purpose processors. Overall, the system has the attractive feature of not requir-

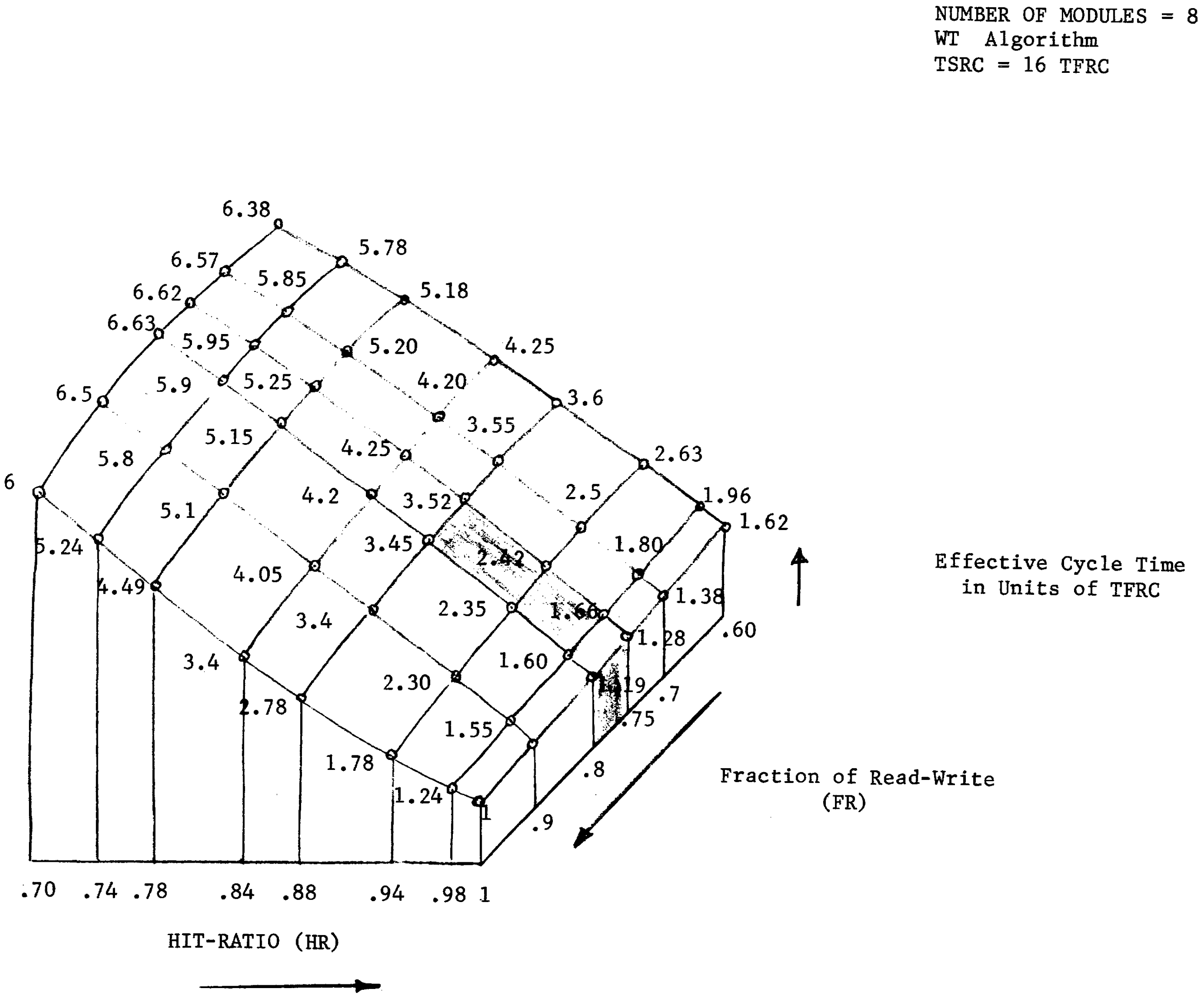


Figure 10

ing any memory bus or memory to have a very high bandwidth.

ACKNOWLEDGMENTS

The authors would like to thank Drs. T. A. Smay and R. J. Zingg for helpful discussions.

REFERENCES

1. Smith, W. R. et al, "SYMBOL—A Large Experimental System Exploring Major Hardware Replacement of Software," in *AFIPS Conference Proc.* (SJCC) March 1971.
2. Rice, R. and W. R. Smith, "SYMBOL: A Major Departure from Classic Software Dominated Von Neumann Computing Systems," in *AFIPS Conference Proc.* (SJCC), March 1971.
3. Wensley, J. H., "The Impact of Electronic Disks on System Architecture," *IEEE Comp.*, Feb. 1975, pp. 44-48.
4. Bloom, L., M. Cohen, and S. Porter, "Considerations" in the Design of a Computer with High Logic-to-Memory Speed Ratio, *Proc. Gigacycle Computing Systems,* AIEE Special Publ., Vol. 5-136, 1962, pp. 53-63.
5. Lee, F. F., "Look Aside Memory Implementation," Project MAC Memo., MAC-M-99, Aug. 19, 1963.
6. Wilkes, M. W., "Slave Memories and Dynamic Storage Allocation," *IEEE Trans. Electron Comput.*, Vol. EC-14, April 1965, pp. 270-271.
7. Gibson, D. H., "Considerations in Block-Oriented System Design," in *AFIPS Conference Proc.* (SJCC), March 1967, pp. 75-80.
8. Liptay, J. S., "Structural Aspects of the System/360 Model 85, II The Cache," *IBM Systems J.*, Vol. 7, No. 1, 1968, p. 15.
9. Lee, F. F., "Study of 'Look-aside' Memory," *IEEE Trans. Comput.* Vol. c-18, Nov. 1969, pp. 1062-1064.
10. Conti, C. J., "Concepts for Buffer Storage," *IEEE Comp. group news,* Vol. 2, Aug. 1969, p. 9.
11. Meade, R. M., "One Memory System Design," *AFIPS Conf. Proc.* (FJCC) Vol. 37, Nov. 1970, p. 33.

12. Mattson, R. L., J. Gecsei, D. R. Slutz, and I. L. Traiger, "Evaluation Techniques for Storage Hierarchies," *IBM Syst. J.* Vol. 9, 1970, p. 2.
13. Mattson, R. L., "Evaluation of Multilevel Memories," *IEEE Trans. Mag.,* Vol. MAG, 7, Dec. 1971, pp. 814-819.
14. Bell, C. G. and D. Casasent, "Implementation of a Buffer Memory in Mini Computers," *Compt. Design,* Vol. 10, Nov. 1971, pp. 83-89.
15. Kaplan, K. R. and R. O. Winder, "Cache Based Computer Systems," *IEEE Comp Soc.* Repository paper. R72-215.
16. Bell, J., D. Casasent and C. G. Bell, "An Investigation of Alternative Cache Organizations," *IEEE Trans. Comp.,* Vol. C-23, April 1974, pp. 346-351.
17. Pohm, A. V., O. P. Agrawal, C. W. Cheng and A. C. Shimp, "An Efficient Flexible Buffered Memory System," *IEEE Trans. Mag.,* Vol. MAG-9, Sept. 1973, pp. 173-179.
18. Pohm, A. V., O. P. Agrawal and R. N. Monroe, "The Cost and Performance Trade-Offs of Buffered Memories," *Proceedings of the IEEE,* Vol. 63, No. 8, Aug., 1975, pp. 1129-1135.
19. Pohm, A. V., "Cost/Performance Perspectives of Paging with Electronic and Electromechanical Backing Stores," *Proc. of IEEE,* Vol. 63, No. 8, Aug. 1975, pp. 1123-1128.
20. "Electronic Replacement for Head-per-Track, Drums or Discs," *IEEE Comp.,* March 1976, pp. 16-20.
21. Agrawal, O. P., R. J. Zingg and A. V. Pohm, "Applicability of Cache Memories to SYMBOL-2R Like Computing Structures," paper to be presented at *Comp Con Conference* 1977, (Feb. 1977)
22. Agrawal, O. P., "Applicability of Buffered Main Memories to SYMBOL-2R Like Computing Structures," Ph.D. Thesis in Electrical Engineering, Iowa State University, Ames, Iowa 1974.

Choosing a medical billing system

by JEFFREY ROTHMEIER
University of Massachusetts Medical Center
Worcester, Massachusetts

ABSTRACT

A billing system is fundamental to the financial survival of an institution. Choosing one requires considerable managerial skill and mistakes are easy to make. Several factors have an impact on the decision-making process. These include environmental, managerial, technical and planning considerations.

Among environmental considerations, the organizational dynamics and organizational structures are most important. The system can be managed in several ways; in-house approaches, use of service bureaus and use of facilities management firms are among the most common management methods. Flexibility, on-line versus off-line systems and the use of large or small computers are major technical considerations. These also have considerable impact on costs. As in most important decisions, the relationship of the billing system to a long-range computer plan should be carefully thought out.

ENVIRONMENTAL CONSIDERATIONS

Most medical institutions make a clear distinction between the segment of the organization which deals with financial management and that segment of the organization which deals with daily administrative operations. This can cause considerable problems in implementing a general computing system because the information necessary to bill patients is a subset of the information necessary to deal with patients for purposes of medical record registration and daily care activities.

Differences in data collected have to do with the types of data that are emphasized. For daily operations in a hospital, for instance, the most important thing may be to know where the patient is located; for billing purposes, it is important to know who the guarantor is and what insurance applies. Numbering systems may also be a source of differences. Each individual ancillary service frequently maintains an accession number and it is not unusual to have different numbering systems for medical records and billing or different systems for outpatients and inpatients. These differences can affect the choice of a billing system. In order to effectively choose an overall system, it is essential that a person be identified who has some influence on all aspects of the medical organization so that differences can be reconciled.

Perhaps the most important environmental consideration is the level of patient activity. In most computing systems there is a fundamental number that determines the kind of system or the power of the system necessary to do the job. Probably the most sensitive number here is either the number of beds or the number of outpatient visits.

The scope of activity is another important environmental variable. The division between inpatients and outpatients or physician and hospital billing can be extremely important. The range of services is a very crucial consideration. Not all billing systems are complete. Consideration of whether or not records from patients who are seen outside of the institution in outlying clinics introduce additional administrative problems. In clinics or community hospitals, the scope of activity is much less than those of a medical center. The computing system requirement is correspondingly decreased.

There may also be operational restraints that greatly affect the choice of the billing system for a particular environment. These are in part, a product of the administrative organization but there may be a valid historical precedence which the individuals who are providing the services are committed to. Attempting to change these by administrative fiat may cause more problems than they solve.

Finally, in many environments, change is frequently a difficult concept. This means that the consideration of installing a new system may be blocked by those people who are comfortable in the jobs that they currently have. The dynamics of an environment can be an important variable in determining whether installation of a new system is at all feasible. If the level of inertia is high, a complex billing system should not be considered.

MANAGERIAL CONSIDERATIONS

There are three basic ways that one can install a billing system within a medical environment:

1. One can run the system on one's own facilities within the institution.
2. One can purchase services from a service bureau. This

can be done in several ways. Terminals can be connected to the service bureau or mail receipt of and transmission of documents between the service bureau and the facility can be used.

3. A vendor can set up a computer center within your facilities and establish a price for their people to do the job.

Each of these approaches has advantages. The particular approach that should be taken depends upon the environment one is in. In general, the in house approach is more attractive to larger environments, i.e., environments of more than 300 beds or more than 5000 outpatient visits per month. Whereas, the shared service approach is generally desirable for smaller environments. The in-house approach has more in common with the facilities management approach than using an outside service. It does remove from the management of the institution the responsibility for the facility and may make the costs more visible since they are a visible part of accounts payable. However, management does have less control over the program than with a totally in-house system. Many observers also believe that though the costs are more visible, they may also be more expensive. This consideration would depend upon the efficiencies and abilities of a particular institution to manage a computer facility. Obviously, if an institution has no in-house expertise and does not choose to develop any, a shared service or a facilities management team should be considered. In-house approaches are particularly important when the system is designed to be very responsive and innovative to local users. Shared services can only implement systems that many users will also find useful. They will usually charge a great deal for changes which appear to them to be of value to a single institution. In many situations, a combination of these approaches may be appropriate. For example, if registration is fundamental to both billing and medical record storage and retrieval, this may be a function that would appropriately be done in house. This system could feed information to a billing system for physicians since it may have unique characteristics in which one may not want to develop in-house expertise. In some instances, technical considerations, like expertise with physician billing, expertise with outpatient billing or familiarity with inpatient billing may determine the approach for choosing the system in each of these three areas. In other instances, pressures from individuals within or without of the institution may be the fundamental considerations. It is very likely that most of the choices depend on combinations of pressures. This may be a fundamental reason why there are so few examples of outstanding billing systems in the United States today.

Finally, it would seem that with the many years that billing systems have been in existence that they would be refined to the point that it would be immediately obvious as to when one system was superior to another. This is however, not the case. The inability to agree on payment priorities for third party billing have also had a major impact on the variety and quality of systems. Billing within the health industry differs from that in other industries, primarily because the person to whom the bill is sent is not usually the one who pays the bill. More often it is a third party who will pay all or part of the bill. Someone has to decide which third party should pay first. It is often possible to separate the quality of one billing system from another by carefully exploring the ability to do prorating.

Reporting requirements from third parties, particularly those of state insurance and federal insurance agencies can also be important considerations in choosing the billing system.

TECHNICAL CONSIDERATIONS

There are a variety of technical considerations which can go into the decision for choosing a billing and accounting system. Perhaps the most fundamental of these is whether an online system or a batch system is desired. As in the case of the divisions between in-house systems, shared systems and facilities management systems, a combination of the two can also be considered. The current trend is toward online systems, but there are still only a limited number of vendors who make them available. In addition, the terminology is not very precise and one vendor's online system may be another vendor's batch oriented system. For instance, simply entering the data online at a terminal does not necessarily mean that the system is online. If in fact the information is not merged into a file for twenty-four hours, then such things as inquiry as to the current status of a patients bill or in particular, having a reasonably accurate bill for an outpatient before they leave the facility will not be possible. It is fair to say that for many aspects of billing, it is not necessary to be totally online. However, as the costs of the hardware continue to decrease, the gains that can be achieved from being online, particularly in the areas of file editing, which insure that files are accurate and for inquiry purposes by a cashier are not to be underestimated.

Another technical consideration that has an impact both on the style of computing and on the cost of computing has to do with whether to choose a minicomputer or a large scale computer. Until very recently, for hospitals in excess of 300 beds, only large scale computers were available. This is no longer the case. Furthermore, the complexity of large systems has become such that it is no longer true that if you double the size of the computer, you will increase the efficiency by a factor of 4. At a certain level, it is even questionable as to whether the efficiency increases at all as the size of the computer increases because of the large overhead costs as the system grows. This applies both to people and to the support of a multiplicity of systems and languages. In addition, if there are too many functions on a large scale computer, it can be difficult to establish priorities. If billing and accounting is one of many functions and payroll is another, there may be sharp disagreements within an institution as to which should come first if equipment or other problems require a delay in schedule. The background of people who relate to small computers or large ones is generally different. There are few computer specialists today who feel comfortable with both minicomputers and large scale machines. The above two fundamental technical decisions can greatly influence the options which a particular institution has available for a billing system.

Another technical consideration is related to the expertise in the billing office. The extent of the reporting that is required is directly related to the sophistication of the people using the reports. Also, some people are willing to do some things manually rather than require the computer to do it. There is a delicate balance between whether the energy to implement a function on a computer justifies it when a person may be able to accomplish the same function with a couple of days of effort each month. Generally, there is a way to distinguish one system from the other by comparing the number and quality of management reports that the system produces.

The timeliness with which the reports are produced and the operational difficulty required to produce them are other considerations. Proration is another important consideration. Very few systems today do a complete and accurate job of prorating the hundreds of different insurance plans which many institutions have to deal with. There are some managers who feel that they would prefer not to have the computer perform this function. This is especially true if it is poorly done. As computing systems do this more accurately and produce complete and usable claim forms, this undoubtedly will be an important part of the decision process in choosing billing systems.

The comparison of systems by the types of reports produced can be very difficult. Generally, the best way to do this is to spend the time with the vendors to hear their point of view and also rely on managerial people within the institution who have had extensive billing experience. Probably the most important thing is to obtain the opinions of other users who have had experience with the billing system under consideration. This can be done in several ways. The ideal way is to make an in-depth visit to the system under discussion. If this is not possible, names of people who have had experience can be obtained and they can be called. Finally, one can try a written survey. Both of the latter two methods may not give sufficient information in order to make a reliable judgment.

Flexibility is another fundamental technical consideration. It is probably one of the most difficult to measure. Nobody would like to admit that he isn't flexible and generally, if one asks specific questions as to whether this report can be changed or that can be changed, one will get from vendors at least a guarded probable yes. It is only when one asks for a commitment in writing that one finds how flexible the system really is. It is very important to write a contract which spells out in as much detail as possible those important considerations and how they will be accommodated.

Reliability is fundamental to all operations. If possible, one should obtain the up time characteristics for any system under consideration. This is very difficult to do and only through discussion with other users can one truly tell whether a system is or is not reliable. Lack of reliability can cause some of the greatest frustrations among the operations people within a computing department and can be a fundamental reason for lack of performance within an institution.

Finally, one of the often talked about technical issues and this is both a technical and a managerial issue, is security. Recent fair information practice legislation has made it difficult to interpret how much security is really desirable within a medical computing system. Nonetheless, it ought to be a conscious decision to make the information available and not something that happens by accident. Security is related both to the technical and managerial aspects of the system. A batch oriented computer system with tight controls on the distribution of the hard copy is the most secure. A dial in time sharing system with no security codes on any files is the least secure. It should not however, be assumed because there are terminals on a system, that it is less secure than one without. Each computing environment can take steps and provide security even with a large number of terminals.

PLANNING CONSIDERATIONS

The choice of a billing and accounting system ought to take into consideration long range planning. This can be done by trying to project total costs over a suitable time period in the future. This should include costs for equipment, outside contracts, internal people costs and operating budget for supplies. A typical example of such a projection for a 400 bed hospital with 100,000 outpatient visits per year who has a desire to interface the billing system with a patient information system can be seen in Figure I. Each of the items within a plan should have more detail which breaks it down into individual functions and individual

	Patient information	Billing and accounting	Pharmacy	Laboratory	Radiology	Total
FY77	72,500	—	—	—	—	72,500
FY78	87,100	74,172	52,200	—	—	213,472
FY79	71,200	74,172	36,300	234,000	—	415,672
FY80	56,800	74,172	11,800	24,000	106,000	272,772
FY81	61,400	74,172	11,800	24,000	21,800	193,172
Totals	349,000	296,688	112,100	282,000	127,800	1,167,588

Figure 1—Summary of direct costs

people and equipment. This will make it possible for a managerial review team to determine whether the costs are in fact being accurately projected and review on a regular basis the progress of the plan.

Finally, the type of billing and accounting system one chooses can be affected a great deal by the relationship of the billing and accounting system to other systems. I mentioned before the importance of registering patients. This is a good place to begin computerization. Hospital programs which do this are generally referred to as admission, discharge and transfer systems. Use of such a system provides the fundamental base for many other activities within the institution.

It would also seem that starting with information systems that are helpful to the physicians, the nursing staff and other clerical people can obtain support and finally result in a successful installation of a billing system. Thus, beginning with systems that provide timely lab results, timely x-ray reports and smooth pharmacy operations, have considerable merit. Frequently, however, it is the case that the individuals who function in these service areas have little experience with the use of computers. Increasingly, their colleagues in other institutions may tell them that they could function more effectively with this tool. However, it is still most common to begin computerization in a medical environment with billing and accounting. In summary, the choice of a billing and accounting system must take into consideration several important factors. Perhaps the most important one is the individual environmental peculiarities. I think we can expect these to be less significant in the future as federal regulations increasingly begin to create uniform data basis, uniform reporting requirements and control many operations previously within the purview of individual institutions. The complexity of a billing operation will continue to make choosing a system a difficult venture for some time in the future.

Designing software for the minicomputer business data processing environment—A case history

by JOHN M. HEMPHILL and RONALD L. LANCASTER
Bowling Green State University
Bowling Green, Ohio

ABSTRACT

The design of software for the minicomputer business data processing environments poses significant practical design problems. This paper deals with the history of the design and implementation of a specific system for that environment. Changes in the system design due to problems in the operational environment are examined. Conclusions are made concerning the design of the system.

INTRODUCTION

The problems experienced in designing and implementing software for the small business environment provide important lessons that illustrate the problems involved in applying computer science principles to solve real life demands on a computer. In this case, the application of interest involves the design of a general-purpose accounting system for medium to large scale automotive dealerships. Certain aspects of the accounting system design (such as selection of hardware, programming language, and operating system) were not negotiable. This was due to the fact that the accounting system to be developed was to run on an existing minicomputer system already being marketed to automotive dealerships.[1]

THE DESIGN GOAL

We were required to implement a double-entry accounting system with on-line data collection, validation, and posting of accounting transaction data. Current account balance information was to be available for demand inquiry at terminals. In addition, the system had to produce generalized schedules of account activity as well as conventional general ledger and journal reports. The system had to be able to retain information about 30,000 to 60,000 account debits and credits.

HARDWARE/SOFTWARE ENVIRONMENT

The system to be developed was to operate within the framework of an existing operating system and application program. The software runs on a Data General NOVA 3/12 minicomputer. The software system supports an automotive warranty system, a payroll system, and a parts inventory system, as well as the accounting system. The language used in this development is BASBOL (BASic Business-Oriented Language). This is an extended version of a multiuser BASIC, with extensions appropriate for business data processing. BASBOL contains variables and constants of extended precision as well as elaborate output formatting capabilities. BASBOL is an interpretive system.

Typical systems in the field have four or five interactive user terminals on one NOVA 3/12 CPU. In addition to having a terminal in the business office, terminals are often available in the parts department, the service department, and in the office of the owner or business manager. A typical system is illustrated in Figure 1.

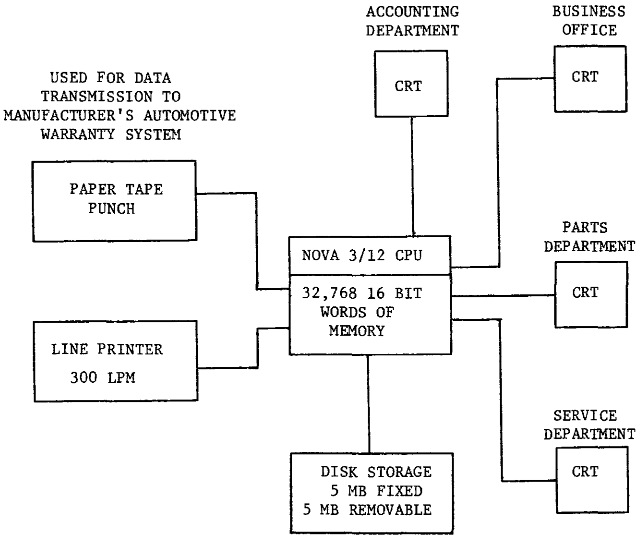
The file system is maintained on a 10 million byte disk and provides access to two types of files: (1) random-access files that are byte addressable, and (2) limited indexed-sequential files that allow quick access to a specific record but require batch updating for addition of new records.

It is important to note that, at the time of the initial system specification, we were told that a tape drive would become available for the real-time journalizing of accounting transactions entered on-line.

Because of constraints imposed by the available memory space and due to the size of the operating system, no accounting program could be larger than 5000 16-bit words in length. An overlay feature was available in BASBOL but it was too slow to allow for frequent use.

INITIAL DESIGN DECISIONS

As stated, this system is designed primarily for use by automotive dealerships. Until the introduction of this system, the accounting and other functions were either done manually or by a "service bureau" which would receive information from the dealership and return reports according to some schedule, usually a brief daily report with more detailed weekly, monthly, and/or annual reports. Such systems typically utilize large sorts and data selection software supported on large computer systems with much


ACCOUNTING DEPARTMENT
BUSINESS OFFICE
CRT
CRT
USED FOR DATA TRANSMISSION TO MANUFACTURER'S AUTOMOTIVE WARRANTY SYSTEM
PAPER TAPE PUNCH
PARTS DEPARTMENT
NOVA 3/12 CPU
32,768 16 BIT WORDS OF MEMORY
CRT
LINE PRINTER 300 LPM
SERVICE DEPARTMENT
DISK STORAGE 5 MB FIXED 5 MB REMOVABLE
CRT

MB - MEGABYTE
CRT - TELETYPE COMPATIBLE CRT TERMINAL
LPM - LINES PER MINUTE

Figure 1—Typical system configuration

I/O capability. In our case, we were to use much smaller, slower hardware with limited I/O capability. Demands by other users for input/output and the limited I/O band width of the hardware made heavy I/O demands involving the disk prohibitive from a time standpoint. We needed to choose a disk structure for our system that would keep the number of disk accesses required during execution to a minimum.

Double-entry accounting systems have two major data structures. The General Ledger contains one entry for each account. A typical automotive dealership will have from 300 to 500 accounts. For each account, the system must have available the current balance figure. In order to produce the required reports, the system must actually retain much more information than this for each account. Other information retained included the account title, the balance at the start of the month, the account "password" (used for protection of information in the interactive environment), and information telling which journals have caused changes in the account balance.

The second major data structure contains the journals. Journals contain the accounting transaction entries—modifications to individual accounts. These entries modify the account balances in the general ledger. Journals are usually defined to contain only transactions of a specified type (e.g., new car sales journal, parts & accessories journal, cash disbursement journal).

In a manual system, information is entered regularly into the various journals. Periodically, the general ledger is updated to reflect recent journal entries. For our automated system, we decided to transmit journal information to the general ledger immediately, so that general ledger balances were always current and available for inspection from any terminal. As a result, journals in our system contain information which is historical, in the sense that the entries had already been reflected in the general ledger.

It should also be noted that an individual transaction (e.g., an individual car sale or purchase) will result in no net change in the general ledger. That is, amounts are transferred from one account to another or the same amount can be added to one account and subtracted from another, but the sum of all account modifications for an individual transaction must be zero.

It was necessary for our system to provide a data structure for retaining transaction information so that "schedules" of account activity could be printed. Basically, a schedule is a record of all transactions which affect a given account or group of accounts. All transactions involving the specified accounts appear on the schedule, regardless of which journal was used to enter the transaction. An example of a common schedule is the accounts receivable schedule. An entry is made to the accounts receivable account when a charge purchase is made for which the car dealer will later issue a bill. By scheduling this account, a listing of all transactions involving that account can provide much information about which customers have paid their bills and which customers have an outstanding balance. Typically, the information on the schedule is sorted so as to facilitate use of the information. Additionally, information gained from processing schedules may be used to remove transaction data which is no longer of interest (such as removing detailed information about a bill which has already been paid).

To implement the system we used two files. The first is an indexed-sequential file that contains the general ledger accounts and balances. It is not often that account numbers are added to or deleted from the general ledger, so the need for a batch run to accomplish this was not a serious disadvantage. The primary purpose of the general ledger is to be able to access information about individual accounts, including account balances.

The second file is known as the transaction data base. This is a random access file which contains all of the individual accounting transactions maintained for historical purposes. This is a relatively volatile file that must provide for quick record deletion and addition. To achieve this goal, plus the goal of maintaining journal and schedule structure among the entries, we selected a linked list structure. The structure used is shown in Figure 2. For easy deletion of records, all lists are doubly-linked, except for the free record list in which deletions and additions are always done at the head of the list.

The initial choice of the general ledger and transaction data base formats proved to be useful. The structures remained essentially unchanged from inception of the project through production use, even though other parts of the system changed.

One unfortunate side effect of using the linked structure in the transaction data base was the necessity to develop a number of support programs to initialize the file, to dump its contents for diagnostic purposes, and to provide list manipulation facilities for other programs in the system.

The time from system design to the shakedown phase

TRANSACTION DATA BASE FILE FORMAT

PROLOGUE	DATA AREA

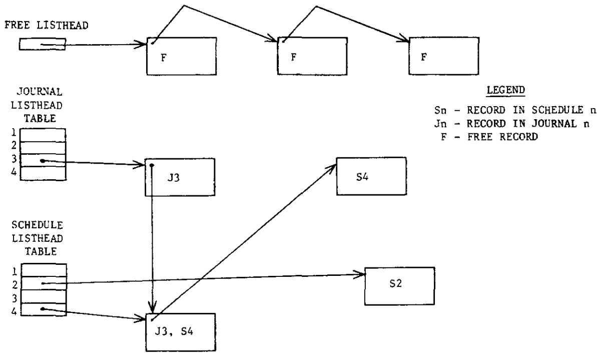
PROLOGUE - CONTAINS INFORMATION THAT DESCRIBES THE FILE. IT CONTAINS LISTHEADS FOR THE JOURNAL LISTS, THE SCHEDULE LISTS AND THE FREE RECORD LIST.

DATA AREA - CONTAINS ALL DATA RECORDS, BOTH FREE AND ALLOCATED.

DATA RECORD FORMAT

JOURNAL LINK FIELD	4 SCHEDULE LINK FIELDS	DATA FIELDS

DATA RECORD - CONTAINS BOTH DATA FIELDS AND LINK FIELDS. EACH LINK FIELD CONTAINS A FRONT AND BACK POINTER. A RECORD CAN BE ON ONE JOURNAL LIST AND FOUR SCHEDULE LISTS SIMULTANEOUSLY OR ANY COMBINATION THEREOF.



THE FREE RECORD LIST IS A SINGLY LINKED LIST. BOTH THE JOURNAL AND SCHEDULE LISTS ARE DOUBLY LINKED. THE MAXIMUM NUMBER OF ENTRIES IN EACH LISTHEAD TABLE IS DETERMINED DURING INSTALLATION OF THE ACCOUNTING SYSTEM AT A PARTICULAR SITE.

Figure 2—Transaction data base file structure

consumed approximately three months of part-time work. When we began serious system testing, we discovered that one of our most dread suspicions had indeed been well founded. The program which collected accounting entries on-line, posted the general ledger and updated the transaction data base was much too slow. The program needed to run fast enough to allow input of data to the terminal with only a delay of a few seconds between account transactions. Instead, delays of 40 to 50 seconds were occurring between transactions involving only six or seven different accounts. This was not acceptable, especially since it was not uncommon for transactions to involve that many accounts. Thus, we had a problem. The difficulty was in the great number of disk arm positionings required to perform the list manipulations in the transaction data base. Since all additions to lists are done at the start of the lists, the solution decided upon was to implement a software supported cache to contain the free listhead and heavily-used journal and schedule listheads in core, updating them on disk when on-line data entry activities were completed.

The cache solved the problem of program execution time by reducing drastically the number of disk operations required to perform the needed list manipulations. At this point in system development, the system was distributed to two customer sites for installation. Little did we suspect the nature of the problem that would next confront us!

Even though we had been told that there would be a tape drive with which to record on-line data entries for use in recovery from system crashes, the drive did not materialize. At first this did not seem to be a serious problem. Then we began having system crashes on our computer, due to hardware malfunctions. Since we were knowledgeable computer users and understood how to restore the disk, such problems were only a minor set-back to us. But in the field at the production sites, there were no users knowledgeable in the ways of the computer.

System crashes at production sites when the cache was used spelled disaster for the integrity of the transaction data base. Since the updated listheads were core resident, a crash meant that the new listhead values were lost even though the list elements themselves were already modified. The net result was that the free listhead on disk (the old value) now pointed to a record that was linked onto a journal list. The transaction data base was now a trap for the unaware user! When the system crashed and destroyed the data base's integrity, the file could still be used for a short period of time for data entry without signs of its being defective. The user had no idea that disaster had befallen his file, rendering it useless in producing meaningful reports. Since some of these files at the point of failure had accumulated 10,000-20,000 transactions, the need for reentering the data tended to render the customer somewhat less than enthusiastic about the system.

Our response to this problem was twofold. First, a systematic backup procedure was instituted. None of the other software systems on the computer was as sensitive to machine failure as the new accounting system. Before the accounting system was delivered, users had backed up the disk on a rather casual basis. Second, we developed a simple data structure testing program known as VERIFY, to give the user a way to determine whether or not the transaction data base is intact. VERIFY performed the rather simple function of checking to see that all of the records on the free list were indeed free. This is possible since a record not allocated to a journal or schedule has a null back pointer value in its link fields. Also, VERIFY made sure that the sum of all account balances in the general ledger was indeed zero. System failure in the middle of updating account balances for a transaction could have made the general ledger totals inconsistent. Thus, VERIFY could warn a user if the transaction data base or the general ledger was damaged, and the user could then restore from a backup disk, losing only transactions entered since the previous backup.

A computer being used by knowledgeable people is in friendly hands. Our system was being used by people with absolutely no computer knowledge and was in hostile territory. First, the users refused to follow backup procedures conscientiously. Second, they continued to enter data even when VERIFY informed them that something was wrong.

At this point, we were somewhat astounded as to what had become of our attempts to assemble a relatively simple on-line data entry system. Two things became clear to us. First, for economic reasons, the system would not have a tape drive for retaining the entered data. Second, we had to reduce drastically the amount of time that elapsed while the transaction data base list manipulation routines were actually being used in order to reduce the possibility that a system crash would damage the integrity of the data base.

Our answer was to collect the accounting transactions in a disk file and defer making the entries in the transaction data base until a later "batch" run. In the original on-line design, a person might spend three to four hours at the terminal entering accounting transactions. During this entire time period, the critical listhead pointers would be maintained in core. A hardware problem at any time during this period would have the potential of destroying one or both of our main files.

By collecting the entries into a file and adding them to the data base in a batch mode, we were able to reduce our vulnerable period to only 10-20 minutes each day. One interesting result was that the person performing the data entry noticed no change in the operation of the system, except that the data entry process was much faster. This was due to the fact that list processing was not being performed at data entry time. Instead, we were simply collecting entries in a sequential file after insuring (by checking the general ledger) that only valid account numbers were being used and that the sum of the dollar amounts for each transaction totalled to zero. This insured that adding the amounts to the general ledger during the "batch" update would not make the ledger out of balance. By collecting entries in this way, the user could run the "batch" program to transfer entries to the two main files at any time during the day. Additionally, in the case of a system disaster that did damage the files, backup was usually possible with no loss of data since all of the current day's entries were in a sequential file that was generally not affected by a system crash due to its simpler structure. At the end of the day, users were instructed to run the VERIFY function. VERIFY would now insure that all entries into the sequential file had been transferred to the transaction data base and that the account balances in the general ledger had been updated.

SUMMARY

The system has been in production use for over one year. It has proven to be a reliable system which has been extended to meet needs discovered since the original design. With the change from on-line file update to deferred batch updating, system failures have not been a serious problem.

The data structures chosen initially proved to be suitable. In retrospect, our major design error was failing to plan properly for the environment in which our system would be running. The problem of developing a reliable software product for the business-oriented minicomputer is not one to be undertaken lightly. We should have initially spent much more time designing the system to be durable in the face of hardware and software crashes as well as less than perfect behavior by the user.

Small business oriented computing is an area that is growing at an increasing rate. However, success in this area seems to be greatly affected by the reliability of the systems that can be delivered. In such systems the designer should first view the system design from the standpoint of reliability. Without operational reliability and durability, the system will be of little value to its users.

REFERENCE

1. Fulton, D. L. and R. T. Thomas, "A Minicomputer-Based Information System for a Small Business," *Computer*, 9,9, September, 1976, pp. 22-28.

What to look for in distributed (source) data processing

by W. HARRY VICKERS
ENTREX, INC.
Burlington, Massachusetts

ABSTRACT

Non-traditional concepts about distributed data processing can change your way of thinking and planning for a system. In this paper, it is suggested that the term *source data processing* is more descriptive of what this subject is all about. It tells everyone immediately where the processing is taking place—at the source of the data, not at some arbitrary place widely dispersed from the central processor. It presents some guidelines for integrating this technique into existing systems and strongly recommends the need to insist on simplicity of design so that everyone can take full advantage of this management tool. Cautioning against limiting one's scope to only the COBOL world, it encourages readers to take a look at the newer software systems and to keep an open mind relative to technology.

INTRODUCTION

Some rather important concepts have recently been made available on computers designed for source data processing. Typically, these concepts are non-traditional, non-standard. But, with source data processing, where there is a high degree of personal interaction with a data base on a small system, the traditional concepts are not always the best concepts. In fact, distributed processing itself is non-standard!

In reading this paper, I hope you will keep an open mind relative to technology. By this, I mean more than hardware technology. Software technology (usually in a lower profile) is the subject that today challenges some traditional concepts held by the EDP establishment. ANSI, CODASYL and similar groups have not really gotten their heads out of the trees long enough to look at some of the newer, more exciting non-traditional, non-standard concepts available on computers designed for distributed (or source) data processing. Don't limit your scope only to the COBOL world or you'll miss out on a good bet. Remember, the traditional concepts are not always the best.

As a matter of fact, the traditional *words* describing what we are talking about do not always do the best job. The semantics in the field of data processing can often cause even the most sophisticated computer professional some confusion. Does anyone know, for example, if distribut*ed* processing, distribut*ive* processing, dispersed processing, and computer networks mean the same thing or something different? Consistent terminology is a key to understanding the plan of action to be discussed. In this paper, the term "Source Data Processing" is used to portray—quite vividly—*where* the data processing is taking place. It means, at the source of the data or in the using department. The action is not happening at some arbitrary place, widely *dispersed* or *distributed* from the central processor through a computer network.

Granted, some data processing may make use of a network arrangement, but you actually *think* differently when you do your planning while standing at the central processing unit (implied by the term distributed) than you do if you put yourself right at the source of the data. When you think and plan from the source point of view, some interesting developments take place. Before looking at these, it might be well to review some reasons for wanting to process the data at the source.

WHY PROCESS AT THE SOURCE OF DATA?

Some companies firmly established with a centralized computing system, may question the need for processing data anywhere else. The same can be said for other companies who may be designing their first system. So, the valid question—"Why process at the source of data?"

One of the factors in the trend toward source data processing is pressure by departmental managers for their own computing capability. Depending upon how far a company has gone toward having P & L oriented departments, the magnitude of the pressure varies. The more an organization says, "the manager shall be sensitive to his own profit," then the more that manager will respond by saying he needs some tools to help run his department efficiently. The computer happens to be one of those tools.

Also significant in allowing the trend toward source data processing is the availability of computer knowledge and of system software. As few as five years ago, the lack of this availability would have made source data processing out of the question, whereas today, it is completely realistic to plan and implement in this direction.

Programming knowledge, in particular, is much more widespread than it was five to ten years ago. This is a standard subject in most high schools and colleges, and the computer industry itself has trained a lot of people. A lot of managers and clerks in using departments now have at least a minimal understanding of computers and are not afraid of them. And the maturity of systems software today means that you can have many features of a mainframe operating system running on a small minicomputer. Features such as virtual memory, high level languages, data base technology, and simultaneous operation of many functions are available in today's minicomputers.

These are intangible reasons to consider processing at the source of data. However, unless the underlying economics dictates that this is the best plan, it will never happen. Used in this context, the word "economics" means: the economics of running a total business. This is partly the economics of collecting and using *accurate* data. It also includes the cost of delay, the cost of stockouts, the cost of customer ill will—all due to *inaccurate* or out of date data.

I stress the economics of *accurate* data because there is a big distinction between accurate data and any other kind.

Accurate data is expensive to get, compared to the other kind. You have to check it for consistency several ways; have it visually approved by someone; and finally checked against your master files. You have to catch errors that can creep in at any point where human voice, hearing, sight, or hand is involved.

When you consider all the clerical personnel involved in these tasks as well as in the typing, filing, and transporting of data; when you consider the computer time devoted to preprocessing for validity checking; and when you consider the costs of correcting misteaks (sic) that sneak through anyway, you will agree that the cost of collecting *accurate* data is indeed the largest cost in any data processing operation.

There are several cost trends that affect your decisions today. Hardware costs have been in a very steep decline for a number of years. The microprocessor has brought about dramatic cost improvements, and over the next few years we are likely to see considerable speed improvements.

Even the large computers are using small computers internally. Semiconductor memory has pushed down the prices of all forms of main memory. Disk storage costs have dropped by a factor of 25 since 1970. Data communications costs have also dropped, although not nearly so dramatically as computing hardware.

On the other hand, labor has gone up an average of eight to ten percent a year; and, in many places, it is hard to find, train, and keep quality people. So quite clearly, the economics are saying, "Use more hardware if it will save labor or, if it will make labor more efficient or more effective." The fact that profit sensitive managers are pushing for source data processing, or their own computer, is good evidence that the economics are in favor of it today.

SOME NON-TRADITIONAL CONCEPTS

With that in mind, and the feeling that source data processing is a viable management tool—one that you may end up with whether or not you plan for it—let's explore some of the nontraditional concepts available today.

One important concept is using the computer to guide the operator through all his tasks, including tasks that are part of the basic system as well as tasks that are part of the specific application. Typically, this is done with a question and answer approach called *menus* or *help* lists. This allows a new operator to quickly learn operational techniques in a step-by-step approach.

Another concept is that the data structure and programming language should be different for a multi-user transaction processing environment than for a batch processing environment. Unfortunately, most "standard" languages have evolved from a batch processing environment. The result is an attempt to add a data base structure on top of a batch processing structure rather than starting over. That is fine if you want to sell hardware, especially memory. A system designed from the ground up can work in $^1/_{10}$th of the main memory of a traditional system.

In addition to these concepts, there are several fundamental guidelines to follow if you intend to decentralize your computing operations or go to source data processing.

First, although it seems axiomatic, *you should plan*. Plan now for source data processing before it sneaks up on you and you get it by default. If you wait, you will end up with a hodgepodge of equipment and no single source to refer to for knowledge about the overall operation. This is not necessarily a bad thing, but you lose some of the flexibility that you can have by planning in advance.

A second guideline that will help you "make it happen" is one that is easy to overlook. That is—*insist on simplicity*. At first blush, you might think this unimportant. Some of your own people might disagree with attaching much importance to this.

By simplicity, I mean simplicity of vendor-supplied software that supports your application, rather than your application software itself. The reason to insist on simplicity is so that everyone can make full use of its potential. The line managers must understand it at more than just a superficial level.

Although they don't have to know how to program a whole application, they should know how the applications work. They should know how the data is structured so that they can determine which are the easy questions to ask the source data processing system, and which questions may be difficult.

For example, a manager might want to know the top ten overdue accounts, or the ten largest orders this week, or the ten employees who have the highest absenteeism. This type of question will come up many times—after the application is programmed. The line managers should not have to go to the EDP group and wait for programming time. The manager doesn't care if the report has a nice heading, or if it is spaced neatly on the page, or even whether he gets extra information or more than ten names.

He wants specific information, and he usually wants it fast. He should be able to get it. With today's source data processing system, based on correct emphasis from the top to "keep it simple," he can write (or have a clerk write) and execute any program in a very short time and have the report. The clerks do not have to be programmers—they need only know a few rules and simple logic. I have seen this work in many companies much to the pleasure of the management. By the way, the execution speed is not nearly as important as the speed with which the program can be correctly written.

From my experience, installation goes much more smoothly when non-EDP people are involved with determining what a "simple" system is. The reason for this is that data processing people, per se, have a traditional data processing education and work experience that makes it difficult to step out of the trees they know so well and see the forest as a non-EDP person sees it. Now, this is not unreasonable and it is certainly not a criticism. It is just that we are back to the problem of semantics again. The DP person interprets the word simplicity from a different perspective than the non-DP manager. Of course, things are simple once you understand them. The DP-er may have spent many hours of study to simplify, or become comfortable with terms like *owner, member, inverted file, chain file, data division, data dictionary, contention, embrace,* etc. To the typical line manager, however, these terms will always be just the data base mystique; some mumbojumbo that is designed to keep him out of the computer room.

But, it really doesn't have to be this way. Today, there are source data processing systems that will give you 95 percent of the functionality of a mumbojumbo system, with a structure that the typical non-EDP manager can understand. This is a structure that can be explained in terms of a manual filing system. Three terms are really all that is needed to explain a data base system: The concept of a *file*, the concept of an *index*, and the concept of *transactions*. A file is like a filing cabinet. An index is like a card file which cross references from one key piece of information to tell you where it is filed. A transaction is a sequential log of what changes were made to the file so the auditors can reconstruct what events took place. Managers and clerical people understand these concepts easily. A computer system can be constructed with these concepts and perform as well as any complicated traditional system.

The concept of security and control is also an important one to consider. In most companies, there are certain master files and certain programs that should not be changed except by a central programming group. On the other hand, there is a legitimate need for user departments to program certain one-shot reports. A good distributed processing system should provide for both needs. It should provide security for important things and yet still allow user access for report generation. If you select a system without both features, you will regret the choice.

If we keep in mind that source data processing is a tool for use by the non-EDP departments, we will automatically involve them in deciding whether or not the proposed system is simple to use or not. They must be able to use this tool for tasks that have not been planned.

HOW TO MAKE IT WORK

When you are convinced that source data processing is the wave of the future and you now want to convince your company that this is the way to go, here are some suggestions that may ensure your success.

The first thing to do is to take it one step at a time. This is an evolutionary process. Don't try to get it all done and then present it to the company as a completed package. However, plan to show some progress fairly quickly so that line managers don't get disenchanted. If they do, they are likely to embark on their own plan and impede your progress.

You can probably take care of the hardest part of the overall problem by solving the most pressing need. As we have seen, for most cases that is capturing accurate and timely data. If you start here, by capturing data *at the source,* you will find that some very tangible progress results in a very short time. It can be done simply and quickly. Once you know that your transaction data are accurate, you can get your data base accurate. The rest is relatively easy.

THE FUTURE

The future is clearly going to be interesting. The structure of the computer industry may be substantially different from what it is today with more vendors having a substantial market share than is now the case. Here are some areas that I see on the horizon as more companies implement source data processing systems.

- Mainframes will really be several minicomputers.
- Large mainframes will be used mainly for computational problems and major reservation systems, such as airlines, rather than for the customary business applications.
- The typical EDP department will function as an internal consultant, auditor, and coordinator of efforts being done by line departments.
- The data base administrator will specify the formatting of key data elements in all source data processing systems and also will specify the formatting of all intercomputer transfers of data, but the data base will be decentralized.

Although the data processing department of the future may be different than it is today, I expect it will be more influential because the computer will permeate to more depths of the business, it will be a more important part of the business, and the people who really understand it will have much more influence on the business.

In any event, following my recommendations to plan, keep it simple, keep an open mind, and to consider source data processing as a tool cannot help but ensure a successful system for you.

RESQ—A package for solution of generalized queueing networks

by C. H. SAUER, M. REISER and E. A. MacNAIR
IBM Thomas J. Watson Research Center
Yorktown Heights, New York

ABSTRACT

RESQ (RESearch Queueing) is a tool for solution of queueing networks. The class of networks treated includes general multi-server queues, passive queues and complex routing decisions. Multiple solution techniques are provided, including numerical solution of separable balance equations and regenerative simulation. User access is provided through both interactive dialogue and a subroutine level interface.

INTRODUCTION

RESQ allows explicit consideration of many system features which are often ignored in queueing models. The goal of RESQ is to provide facilities for convenient model construction and efficient model solution so that the user can concentrate on formulating models. The user, e.g., a system designer or developer, need not be sophisticated with respect to the methods of solution. Since several solution methods are provided, the user can use the method most appropriate to the model and can use two or more methods in a hybrid solution. The constructs of RESQ are oriented toward computer and communication system features, but the terminology is strictly in terms of queueing networks.

RESQ employs state of the art techniques for solution of queueing networks. Depending on the particular model, the solution may be obtained by numerical analysis of separable balance equations[1] or simulation.[2] The separable balance equation solutions are available for a subset of RESQ models. The simulation techniques include the regenerative method for determination of confidence intervals and a sequential sampling method for determination of appropriate run lengths. By providing a high level framework for model definition and appropriate analysis of simulation results, RESQ alleviates two of the common problems with simulation: expense of constructing simulation models and insufficient statistical analysis of simulation results. Other techniques and solution methods are being included in RESQ on an experimental basis.

All RESQ capabilities are provided through a set of PL/I programs. In addition, the user interface components of RESQ are duplicated in APL. Queueing networks can be defined, listed, evaluated and revised either interactively or by writing programs which call RESQ routines.

This paper is organized as follows: The second section briefly summarizes related previous work on application of queueing network models, solution techniques for queueing networks and solution packages. The third section describes the generalized class of networks provided for in RESQ and the dialogues for network definition. The last section considers listing, evaluation and revision of networks. Additional details are given in Reference 4.

QUEUEING NETWORKS

In analyzing the performance of computing and communication systems, one usually finds the dominant factor to be contention for resources such as processors, memory, secondary storage, communication links, etc. Therefore queueing network models can be used to characterize this contention and estimate system performance. Some of the earlier efforts in this area were those of Kleinrock,[5] Smith[6] and Buzen.[7] More recently there has been much work in this area, see References 8, 9, and 10 for examples and further references.

Corresponding to the activity in application of queueing network models, there has been much progress in the solution of queueing networks. Very complex queueing networks can be represented as Markovian processes and most of the solution efforts have done so. The direct numerical solution of these processes can be attempted for modest size problems[11] but this approach is not practical in general. Representing the solution of the process as that of a collection of separable balance equations has made possible the solution of very large problems with restricting assumptions.[1,12,13] There is hope that approximate solutions will alleviate the need for such assumptions.[8,14,15] Finally, there has been much recent effort to improve statistical analysis of simulation, in particular the simulation of regenerative systems such as Markov processes.[16–19]

There have been a variety of packages proposed and implemented for the solution of queueing networks.[20–25] However, all of these provide only a single solution technique and thus are only useful when the solution technique is appropriate to the problem. One of the major advantages of RESQ is that it provides several solution techniques. Thus the user can solve similar models and study the effects of different restricting assumptions (it is rare that the modeler can explicitly consider all system characteristics) and can construct hybrid solutions using more than one solution technique.[10]

RESQ CONSTRUCTS AND INTERACTIVE DIALOGUES

This section briefly describes the constructs of RESQ and illustrates some of them with a model of a simple terminal oriented computing system. This model is an extension of the *central server model* proposed by Buzen,[7] and is similar to models discussed in References 8 and 9.

The elements of RESQ include:

1. A population of jobs. Each job has an attached variable which can be used to retain job attributes.
2. A set of queues. There are two types of queues, *active* and *passive*. Active queues are queues in the traditional sense. Passive queues are used to represent contention for secondary resources and regulate subnetwork job populations.
3. A set of nodes. Some types of nodes are parts of queues. Other nodes are used for auxiliary functions such as creation of jobs or changing the value of a job's variable.
4. A set of routing rules. These rules allow probabilistic and deterministic routing of jobs from among the nodes of the network.

In the example, jobs represent users of the system. A job alternates between think times at a terminal and use of the computational facilities. When a job is to perform computations, it first acquires memory then alternates between use of a central processor and use of the input/output devices. The job variable is used to count the number of cycles of alternating computation and input/output. A passive queue is used to represent memory; active queues are used to represent other components. A model such as this can be used to estimate response times, device utilizations, queue lengths and other performance metrics.[8]

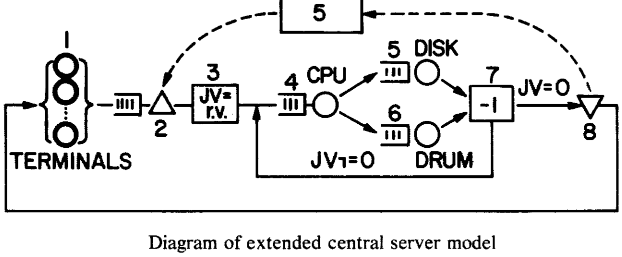


Diagram of extended central server model

In interactive usage of RESQ the command SETUP is used to define a network. As with other RESQ commands, SETUP enters the user into a dialogue where RESQ will prompt the user for information, e.g., the name of the model, queue characteristics, etc. If the user's response seems correct to RESQ, more information will be requested by RESQ until the command is finished. If RESQ discovers an error in the user's response then it will produce an error message and repeat the prompt. If the user wishes clarification of the prompt, then the user replies "how," RESQ responds with a detailed description of the information needed, and then RESQ repeats the request. If there is a default value for a particular prompt, then the user may effect the default value by entering a return of a null line. In this section, user responses will always be given in lower case.

Active queues

An *active queue* consists of a set of servers, a set of waiting areas for jobs requesting or receiving service and a control mechanism for allocating the servers to the jobs.

Single server queues

The waiting areas of the queue are called *classes*. These classes are local to the queue and are to be distinguished from the global "classes" often used in queueing literature. As described in a later section, a job is routed to one of these classes and joins the queue. Upon arrival the work demanded by the job is determined as follows: First, a sample is taken from the work demand distribution associated with the job's class. Second, the flag for job variable scaling is tested. There is such a flag for each class. Job variables will be discussed in a later section. If the flag is set, then the sample taken in the first step is multiplied by the job variable.

Once placed in the waiting area, the job remains there until all of the work demanded is completed. When all of the job's work is complete, the job instantaneously departs from the queue. The server chooses which job to serve according to the overhead mechanism, if any, and the queueing discipline.

Multi-server queues

Queues with more than one server allow all of the capabilities described above except for the cyclic priority queueing discipline. Each server has associated with it an effective rate and a set of classes which it will serve. The rate of a server may be a function of the total number of jobs at the queue. If all servers at a queue have exactly the same characteristics, the queue is considered *symmetric*. Otherwise it is *asymmetric*.

Dialogue for active queues

The first prompt is for the queue type. The subsequent prompts are strongly dependent on the queue type. In

```
setup
MODEL NAME:  ecsm
METHOD:  aplomb
```

Definition of Network Size

```
NUMBER OF:
   CHAINS:  1
   QUEUES:  5
   CLASSES:  4
   ALLOCATE NODES:  1
   RELEASE NODES:  1
   DESTROY NODES:  0
   CREATE NODES:  0
   SET NODES:  2
   FISSION NODES:  0
   FUSION NODES:  0
   SPLIT NODES:  0
   NUMBERED SOURCES:  0
   DUMMY NODES:  0

COMMENTS?  yes
COMMENT:  extended central server model
```

Definition of Routing

```
CHAIN: 1   TYPE:  closed
COMMENT:  jobs have a think time at a terminal.
:  then they request memory.
:  after being allocated a partition, they determine
:  their number of processing/data transfer cycles.
:  after this number of cycles they release their
:  partition and go back to the thinking state.
:
( 1):  1->2->3->4->5 6;.2 .8
( 2):  5 6->7->4 8;jv¬=0 jv=0
( 3):  8->1
( 4):
CHAIN POPULATION:  20
```

Definitions of Queues

```
QUEUE  1   TYPE:  is
COMMENT:  terminals (is -- infinite server)
:
CLASS LIST:  1
STME. DISTR:  5

QUEUE  2   TYPE:  passive
COMMENT:  memory partitions
:
TOKENS:  5
QDSPL:  fcfs
ALLOCATE NODE LIST:  2
         AMOUNT(S):  1
 RELEASE NODE LIST:  8

QUEUE  3   TYPE:  active
COMMENT:  central processing unit
:
SERVERS:  1
QDSPL:  ps
CLASS LIST:  4
WORK DMND. DISTR:  <.002,1>
JV SCALED:  no
```

SETUP for extended central server model

addition to the two basic queue types, *active* and *passive*, there are several special cases for simplified active queues. These simplified cases are indicated by responding with a queueing discipline to the prompt for queue type. The response to the queue type prompt of "active" allows all of the options described above. The distributions, e.g., the work demand distribution for each class, may be specified as either a single value which is interpreted as the mean of an exponential distribution, or as a pair of values in angular brackets ("⟨", "⟩") which are interpreted as a mean and coefficient of variation, respectively. If all classes of the queue are to have the same distribution, the list may be replaced by a single distribution which will be used for all classes.

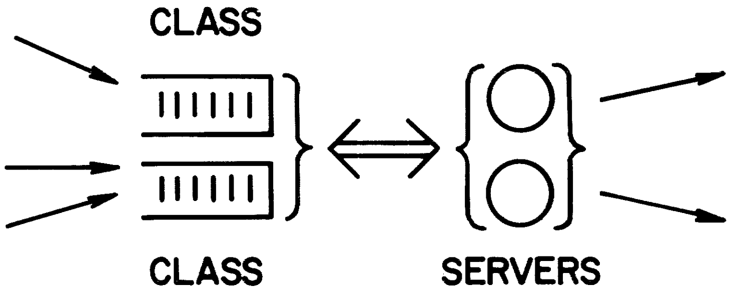
Passive queues

A passive queue consists of a pool of tokens, a non-empty set of waiting areas for jobs requesting or possessing tokens, a possibly empty set of other nodes for actions on the queue and a control mechanism for the tokens and jobs.

```
SERVER  1:
   RATE:  .1
   ACCEPTS:  all

QUEUE  4   TYPE:  active
COMMENT:  disk
:
SERVERS:  1
QDSPL:  fcfs
OVHD:  none
CLASS LIST:  5
WORK DMND. DISTR:  .044
JV SCALED:  no
SERVER  1:
   RATE:  1
   ACCEPTS:  all

QUEUE  5   TYPE:  fcfs
COMMENT:  drum
:
CLASS LIST:  6
STME. DISTR:  .008

SET NODES:  3             7
   SET TO:  r            -1

   DISTRIBUTION FOR NODE     3
   VALUES:  10    20
   PROBS:  .5    .5

END OF SETUP.
```

Definition of Set Nodes

SETUP for Extended Central Server Model



Active queue with two classes and two servers

The tokens of the passive queue are analogous to the servers of an active queue. The waiting areas are called *allocate nodes*. There are three other types of nodes which may be associated with a passive queue: *release nodes, destroy nodes* and *create nodes*. The usual purpose of passive queues is to limit or measure the population of subnetworks.

Allocate nodes

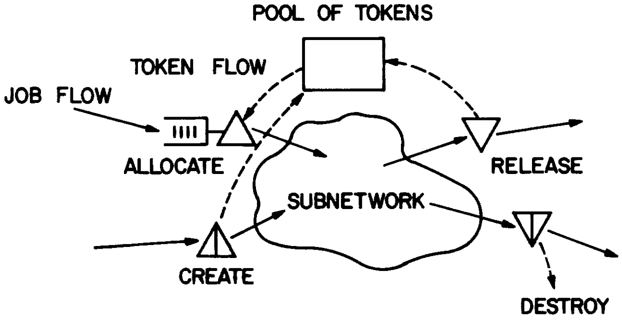
A job arriving at an allocate node requests possession of a number of the queue's tokens. If the tokens requested by a job are available at the time of arrival at an allocate node, then the request is satisfied instantaneously. Otherwise the job must wait until sufficient tokens become available and are assigned to the job. (Tokens become available through the action of other jobs at other nodes.) As soon as the request for tokens is satisfied, the job is allowed to visit other nodes of the network. However, as long as the job possesses tokens of a given queue, it is considered to be part of that queue. Thus a single job may be a member of one or more passive queues and one active queue simultaneously.

Release nodes

When a job visits a release node associated with a queue which the job is a part of, the job instantaneously returns all its tokens belonging to the queue. When a job visits a release node associated with a queue which the job is not a part of, there is no effect on the job or the queue. In either case the job's visit to the release node is instantaneous and the job proceeds without delay.

Destroy nodes

When a job visits a destroy node associated with a queue which the job is a part of, the job instantaneously destroys ~~all its tokens belonging to the queue; then the job is no~~ longer part of the queue. When a job visits a destroy node associated with a queue which the job is not a part of, there is no effect on the job or the queue. In either case the job's visit to the destroy node is instantaneous and the job proceeds without delay.



Passive queue and associated nodes

Create nodes

A job visiting a create node adds new tokens to the pool of its associated queue. The number added is determined by sampling from a discrete distribution associated with the node. There is no effect on the job; its visit is instantaneous and it proceeds without delay.

Sources

A source emits jobs one at a time. The time between a given arrival from a source and the next arrival from a source is determined by a sample from a continuous distribution associated with the source. The job variable is set to zero when the job is emitted. The description of sources is included in the dialogue describing the routing.

Sinks

Sinks are nodes which allow jobs to exit from the network. A job exiting from the network releases all tokens held, if any, and returns them to the appropriate pools. The exiting process is instantaneous. The description of sinks is included in the dialogue describing the routing.

Set nodes

A set node is used to affect the value of a job variable. A job's variable will be zero unless it has been given some other value by a set node. Job variables are useful for making work and overhead demands job dependent. They are also especially useful for effecting deterministic job dependent routing, e.g., to cause a job to cycle through a set of nodes for a predetermined number of cycles. There are five kinds of set nodes, *assignment* set nodes, *increment* set nodes, *decrement* set nodes, *change sign* set nodes and *previous node* set nodes.

An assignment set node assigns a non-negative value to a job's variable, an increment set increments a job's variable by a non-negative value and a decrement set node decrements a job's variable by a non-negative value. In any of these cases, the values used are samples from a distribution. The distribution is associated with the set node and may be either continuous or discrete. A *change sign* set node changes the sign of the job variable. A *previous node* set node assigns to the job variable the identity of the node the job just left. A job's visit to a set node is instantaneous.

Fission and fusion nodes

A job arriving at a fission node generates one or more additional jobs. The generating job is referred to as the *parent* and the generated jobs are referred to as *offspring*. A parent job and its offspring are considered to be *related* and know the identities of each other. Each of these jobs has a separate routing from the fission node. The visit of the parent job is instantaneous; the offspring depart from the node immediately after generation. The offspring do not possess any tokens; their job variables have the value zero. Combinations of fission and fusion nodes are useful for representing packetizing of messages in a communication network. They are also useful in models of computing systems to represent overlap of processing and data transfer.

A fusion node provides a waiting area for related jobs. Related jobs wait at a fusion node until they have no relatives; they then depart instantaneously. If a job arrives at a fusion node where it has a related job, one of the jobs is eliminated from the network instantaneously. If one of the jobs is the parent of the other then the other job is eliminated. If both of the jobs are offspring then it is left undefined which job will be eliminated. Any tokens possessed by the eliminated job are returned to the appropriate pool. (Fusion nodes have no effect on jobs without relatives. If a job is waiting at a fusion node and all of its relatives leave the network, the job departs from the node immediately.)

The description of fission nodes appears in the routing dialogue. Fusion nodes are identified by a single prompt for a list of fusion nodes.

Split nodes

Split nodes are like fission nodes with the difference that the generating job and the generated jobs are independent of each other; they are not considered to be related. Split nodes are useful in representing bulk arrivals. They are also useful in communication network models to represent the generation of control messages. The description of split nodes appears in the routing dialogue.

Routing

All nodes except fission nodes, split nodes and sinks may have several alternate routing paths for jobs leaving the node. Fission and Split nodes have separate fixed routing paths for the creating job and each created job. (Dummy nodes may be used with fission or split nodes to provide alternate routing paths for jobs leaving those nodes.) A node with alternate routing has a list of possible routings. Each item on the list consists of the identity of a possible destination node and either a predicate or a probability. The predicates are statements about values of job variables, availability of tokens, etc. A job leaving a node with alternate routing selects a destination by scanning the list until it finds a predicate which is true or until it succeeds at a Bernoulli trial with one of the given probabilities. If the list is exhausted without a node being selected, the results are undefined. A job travels from one node to the next instantaneously.

The nodes of a network are separated into one or more disjoint sets called *chains*. A chain is defined as the largest subset of nodes such that all nodes of the chain are connected. *Open* chains are those that include sources and/or sinks. All other chains are *closed*.

The routing is described separately for each chain of nodes. Before prompting for the routing description, SETUP prompts for the type of the chain, open or closed. Then there is prompting for routing transitions. A routing transition consists of a list of nodes, a list of alternative destinations for those nodes and a list of probabilities and predicates. Transitions may be entered individually or the concatenation of two or more transitions may be entered simultaneously. The prompt for a transition or concatenation of transitions is a parenthesized integer. The value of the integer has no significance except to number the entries the user has made during the input dialogue. The prompting for transitions or concatenations of transitions continues until a null line is entered.

Consistency checks and error messages

After the dialogue is complete, SETUP goes through a variety of consistency checks to look for errors in the model or incompatibilities with the solution technique. If errors are found, an error message (presumably a self-explanatory error message) will be given.

MODEL LISTING, SOLUTION AND EVALUATION

LIST

The command LIST produces a tabular listing of a model. The listing includes, in order, the solution method, the model name, the numbers of elements, the characteristics of the chains, the characteristics of the queues and the characteristics of the nodes. Headings are suppressed for columns which have no entries, e.g., "classes accepted" for networks with only symmetric queues.

EVAL

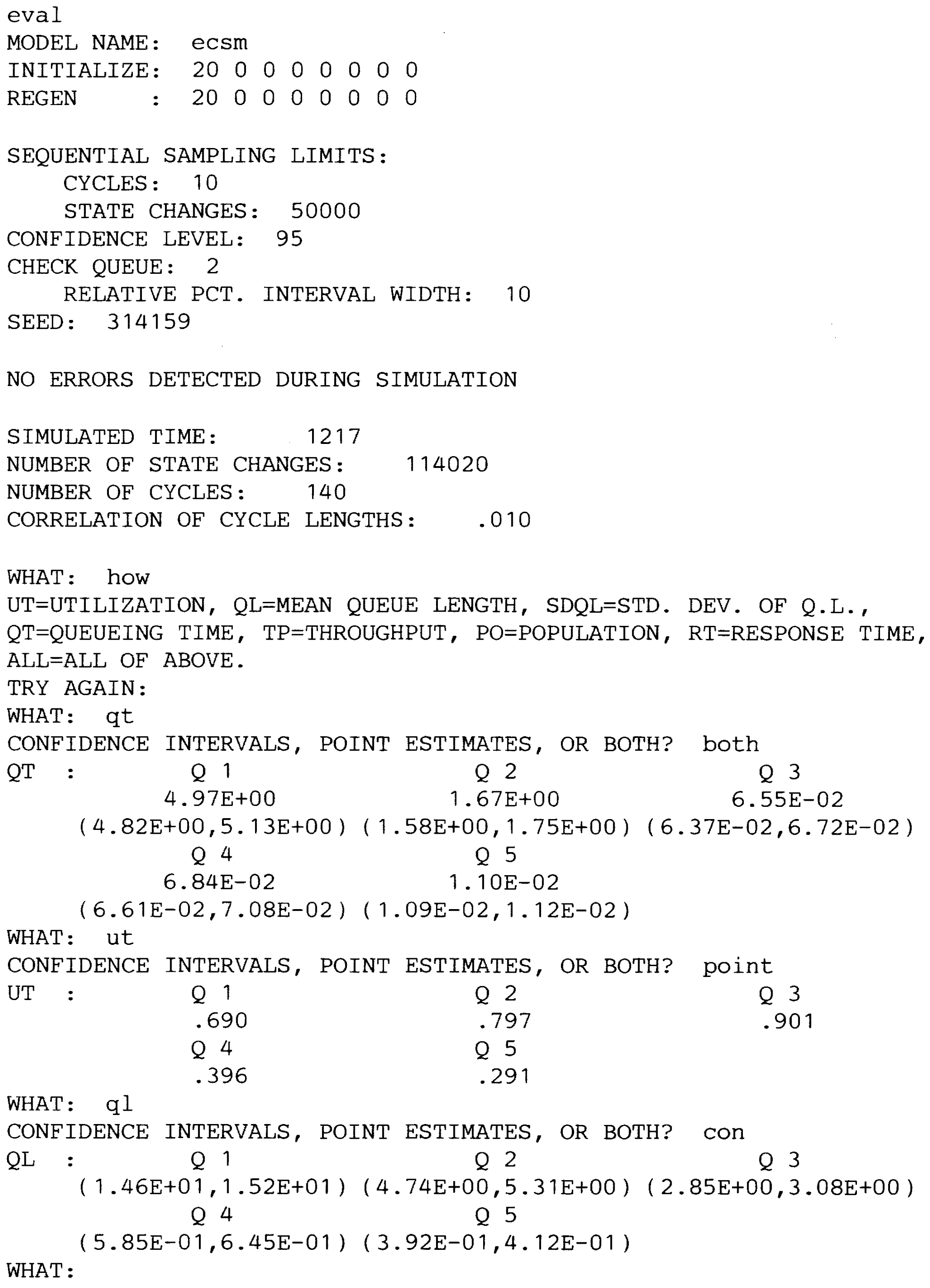
The command EVAL is used to apply the solution method specified with the model and to examine the results produced by the solution method. The most general solution technique is simulation (APLOMB). Numerical (QNET4) solutions are available for a subset of the class of models simulated.

APLOMB

APLOMB is a simulation program specifically designed for the class of queueing networks represented in RESQ. A major feature of APLOMB is its capability for determining confidence intervals for simulation results. (Note that even if this capability is used incorrectly and APLOMB is unable

```
list
MODEL NAME:  ecsm
SOLUTION METHOD:  APLOMB
MODEL NAME:  ECSM

  1 CLOSED CHAIN(S)
  0 OPEN CHAIN(S)
  5 QUEUE(S)
  4 CLASS(ES)
  8 NODE(S)

COMMENT:  EXTENDED CENTRAL SERVER MODEL

CHAIN| TYPE |POP |COMMENT
-----|------|----|--------------------------------------------------
   1 |CLOSED|  20|JOBS HAVE A THINK TIME AT A TERMINAL.
                  THEN THEY REQUEST MEMORY.
                  AFTER BEING ALLOCATED A PARTITION, THEY DETERMINE
                  THEIR NUMBER OF PROCESSING/DATA TRANSFER CYCLES.
                  AFTER THIS NUMBER OF CYCLES THEY RELEASE THEIR
                  PARTITION AND GO BACK TO THE THINKING STATE.

QUEUE| TYPE  |Q DSP |MS|MQ|RATE(S)|C.A.|COMMENT
-----|-------|------|--|--|-------|----|------------------------------
   1 |ACTIVE |IS    | 1| 1|   1.00|    |TERMINALS (IS -- INFINITE SERVER)
   2 |PASSIVE|FCFS  | 5| 1|       |    |MEMORY PARTITIONS
   3 |ACTIVE |PS    | 1| 1|   0.10|   4|CENTRAL PROCESSING UNIT
   4 |ACTIVE |FCFS  | 1| 1|   1.00|   5|DISK
   5 |ACTIVE |FCFS  | 1| 1|   1.00|    |DRUM

NODE| Q |CHN|TYPE | WD/ST|CW/S| DDV    | DDP
----|---|---|-----|------|----|--------|--------
   1|  1|  1|CLASS|  5.00|1.00|        |
   2|  2|  1|ALLOC|      |    | 1.0    | 1.0
   3|   |  1|SET  |      |    |10. 20. |  .5  .5
   4|  3|  1|CLASS|  .002|1.00|        |
   5|  4|  1|CLASS|  .044|1.00|        |
   6|  5|  1|CLASS|  .008|1.00|        |
   7|   |  1|SETDC|  1.00|0.  |        |
   8|  2|  1|RELSE|      |    |        |

FROM|  TO|INDICATOR|VALUE
----|----|---------|-----
   1|   2|  1.00   |
   2|   3|  1.00   |
   3|   4|  1.00   |
   4|   5|   .20   |
   4|   6|   .80   |
   5|   7|  1.00   |
   6|   7|  1.00   |
   7|   4|  JV¬=   | 0.
   7|   8|  JV=    | 0.
   8|   1|  1.00   |
```

LIST for central server model



```
eval
MODEL NAME:  ecsm
INITIALIZE:  20 0 0 0 0 0 0 0
REGEN     :  20 0 0 0 0 0 0 0

SEQUENTIAL SAMPLING LIMITS:
    CYCLES:  10
    STATE CHANGES:  50000
CONFIDENCE LEVEL:  95
CHECK QUEUE:  2
    RELATIVE PCT. INTERVAL WIDTH:  10
SEED:  314159

NO ERRORS DETECTED DURING SIMULATION

SIMULATED TIME:       1217
NUMBER OF STATE CHANGES:     114020
NUMBER OF CYCLES:     140
CORRELATION OF CYCLE LENGTHS:     .010

WHAT:  how
UT=UTILIZATION, QL=MEAN QUEUE LENGTH, SDQL=STD. DEV. OF Q.L.,
QT=QUEUEING TIME, TP=THROUGHPUT, PO=POPULATION, RT=RESPONSE TIME,
ALL=ALL OF ABOVE.
TRY AGAIN:
WHAT:  qt
CONFIDENCE INTERVALS, POINT ESTIMATES, OR BOTH?  both
QT  :          Q 1                 Q 2                 Q 3
            4.97E+00            1.67E+00            6.55E-02
     (4.82E+00,5.13E+00) (1.58E+00,1.75E+00) (6.37E-02,6.72E-02)
               Q 4                 Q 5
            6.84E-02            1.10E-02
     (6.61E-02,7.08E-02) (1.09E-02,1.12E-02)
WHAT:  ut
CONFIDENCE INTERVALS, POINT ESTIMATES, OR BOTH?  point
UT  :          Q 1                 Q 2                 Q 3
               .690                .797                .901
               Q 4                 Q 5
               .396                .291
WHAT:  ql
CONFIDENCE INTERVALS, POINT ESTIMATES, OR BOTH?  con
QL  :          Q 1                 Q 2                 Q 3
     (1.46E+01,1.52E+01) (4.74E+00,5.31E+00) (2.85E+00,3.08E+00)
               Q 4                 Q 5
     (5.85E-01,6.45E-01) (3.92E-01,4.12E-01)
WHAT:
```

EVAL for central server model

to provide confidence intervals, APLOMB will still provide the user with point estimates.) When the solution method specified for the model is APLOMB, EVAL prompts the user for parameters used in determining confidence intervals and in controlling the simulation.

Initialization and regeneration

In order to apply the confidence interval techniques, one must specify a system state called the "regeneration" state. The occurrences of this state divide the simulation into independent cycles. Presumably the simulation will enter this state frequently. Usually the regeneration state will be chosen so that there are no jobs in any nodes belonging to open chains. Usually the regeneration state will be chosen so that jobs of closed chains are distributed among the nodes according to expected populations of these nodes. Of course one usually will not know in advance the populations at the various nodes; reasonable guesses are usually sufficient and poor guesses are often workable. Since visits to most nodes are instantaneous, non-zero expected populations are only reasonable at classes, allocate nodes and fusion nodes. The first prompt from EVAL is for the number of jobs to be initialized at each node. The reply should be a list of non-negative integers, with as many elements in the list as named nodes in the network. The second prompt is for the number of jobs at each node in the regeneration state.

Sequential sampling procedure

The next series of prompts determines when the simulation will stop. APLOMB uses a sequential sampling procedure to run the simulation until satisfactory confidence intervals are obtained. The procedure has two limits, on the number of regeneration cycles and on the number of state changes, to control the period between samples. The sampling period ends when the first of the two limits is reached for that period. If after the first sampling period has ended too few cycles have been completed to compute confidence intervals, the simulation ends and only point estimates are provided. Otherwise, the width of the confidence interval for the mean queueing time at a given queue, relative to the point estimate of the queueing time for that queue, is compared to a threshold. If the threshold is exceeded, then a new sampling period is begun. Sampling periods continue until the relative width does not exceed the threshold. The first prompt of the series requests the sampling period limit on the number of regeneration cycles. The second prompt of the series requests the sampling period limit on the number of state changes. The third prompt is for the confidence level of the intervals. The fourth prompt is for the number of the queue to be used in determining whether to continue sampling or not. The final prompt is for the threshold for the relative width of the confidence interval for the mean waiting time at the queue. The relative width is expressed in percent of the point estimate. Notice that the sequential sampling procedure can be defeated by specifying a very large threshold, e.g. 200 percent.

Seeds for pseudo-random streams

The final prompt before simulation begins is for an integer to be used as a seed for the pseudo-random number streams. Each random variable in the network has its own stream. Each of these streams has its own seed. The user specified seed is used to start a stream which produces seeds for all of the other streams.

Simulation results

After the simulation ends, EVAL will either respond "NO ERRORS DETECTED DURING SIMULATION" or will give an error message if an error was discovered during simulation. Then EVAL gives the simulated time, the number of state changes that occurred, the number of regeneration cycles completed and an estimate of the correlation between the lengths of successive regeneration cycles. (This estimate should be near zero if the state chosen is actually a regeneration state.)

EVAL now prompts the user with "WHAT:" and is ready to provide simulation results. The user replies with a code indicating the type of results desired. Results of that type are given for all appropriate elements (queue, node or chain).

QNET4

If the solution method is QNET4, then no solution dependent information is required. The prompting begins immediately with "WHAT:" and the same codes are used as with APLOMB. There is no option for confidence intervals since the QNET4 values are exact within the limits of numerical error.

CHANGE

CHANGE allows more or less arbitrary revisions of a model. Most of the dialogues are similar to dialogues occurring in SETUP. Unless otherwise stated, the same responses may be given in CHANGE as may be given for the corresponding prompts in SETUP. Often additional information is provided for the user's reference.

SUMMARY

RESQ makes possible and convenient the solution of a great variety of queueing network models. The many constructs allowed given the modeler the freedom to study a variety of system characteristics and to determine the degree that various characteristics impact performance.

In addition, RESQ provides a variety of solution techniques. Thus the modeler can make tradeoffs between expense of solution and model accuracy. The modeler can ~~use a combination of techniques to form a hybrid solution.~~ The solution techniques provided are the best available for this class of problems. Proposed solution techniques are being included in RESQ on an experimental basis. The multi-solution technique capability is also helpful in studying these proposed techniques.

Finally, the user interface is designed for convenience for a variety of users. Interactive dialogues are provided. Subroutine level interfaces are also available for repeated or specialized usage of RESQ.

REFERENCES

1. Reiser, M. and H. Kobayashi, "Queueing Networks with Multiple Closed Chains: Theory and Computational Algorithms," *IBM J. of Research and Development*, 19,3, May 1975.
2. Sauer, C. H., "Characterization and Simulation of Generalized Queueing Networks," IBM Research Report RC-6057, IBM T. J. Watson Research Center, Yorktown Heights, New York (May 1976).

4. Reiser, M. and C. H. Sauer, "Queueing Network Models: Methods of Solution and their Program Implementation," to appear in K. M. Chandy and R. T. Yeh, editors, *Current Trends in Programming Methodology, Volume III: Software Modeling and Its Impact on Performance.* Prentice Hall, 1977.
5. Kleinrock, L., *Communication Nets,* McGraw-Hill Book Company, 1964.
6. Smith, J. L., "An Analysis of Time Sharing Computer Systems Using Markov Models," *Proceedings Spring Joint Computer Conference,* 1966.
7. Buzen, J., *Queueing Network Models of Multiprogramming,* Ph.D. Dissertation, Division of Engineering and Applied Physics, Harvard University, 1971.
8. Brown, R. M., *An Analytic Model of a Large Scale Interactive System Including the Effects of Finite Main Memory,* M.A. Thesis, University of Texas at Austin, 1974.
9. Boyse, J. W. and D. R. Warn, "A Straightforward Model for Computer Performance Prediction," *Computing Surveys* 7,2, 1975.
10. Browne, J. C., K. M. Chandy, R. M. Brown, T. W. Keller, D. Towsley and C. W. Dissley, "Hierarchical Techniques for Development of Realistic Models of Complex Computer Systems," *IEEE Proceedings* **63,** 6, 1975.
11. Wallace, V. L. and R. S. Rosenberg, "Markovian Models and Numerical Analysis of Computer System Behavior," *Proceedings Spring Joint Computer Conference,* 1966.
12. Chandy, K. M., "The Analysis and Solutions for General Queueing Networks," *Proc. Sixth Annual Princeton Conference on Information Sciences and Systems,* Princeton University, March 1972.
13. Baskett, F., K. M. Chandy, R. R. Muntz, and F. Palacios-Gomez, "Open, Closed, and Mixed Networks of Queues with Different Classes of Customers," *JACM* 22,2.
14. Brandwajn, A., "Equivalence and Decomposition Methods with Application to a Model of a Time-sharing Virtual Memory System," *Proceedings International Symposium Rocquencourt,* April 1974.
15. Chandy, K. M., U. Herzog, and L. S. Woo, "Approximate Analysis of General Queueing Networks, *IBM Journal of Research and Development* **19,** 1, April 1975.
16. Crane, M. A. and D. L. Iglehart, "Simulating Stable Stochastic Systems, I; General Multiseiver Queues," *JACM* 21, January 1974.
17. Crane, M. A. and D. L. Iglehart, "Simulating Stable Stochastic Systems, II; Markov Chains," *JACM* 21, January 1974.
18. Lavenberg, S. S. and D. R. Slutz, "Introduction to Regenerative Simulation," *IBM Journal of Research and Development* **19,** 5, 1975.
19. Iglehart, D. L., "The Regenerative Method for Simulation Analysis," to appear in K. M. Chandy and R. T. Yeh, editors, *Current Trends in Programming Methodology, Volume III: Software Modeling and Its Impact on Performance,* Prentice Hall, 1977.
20. Irani, K. B. and V. L. Wallace, "On Network Linguistics and the Conversational Design of Queueing Networks," *JACM* 18,4, 1971.
21. Muntz, R. R. and J. Wong, "Efficient Computational Procedures for Closed Queueing Networks with the Product Form Solution," *Hawaii International Conference on Systems Sciences,* January 1974.
22. Foster, D. V., P. F. McGehearty, C. H. Sauer and C. N. Waggoner, "A Language for Analysis of Queueing Models," *Proceedings Fifth Annual Pittsburgh Modeling and Simulation Conference,* University of Pittsburgh, April 1974.
23. Keller, T. W., *ASQ User's Manual,* TR-27, Department of Computer Sciences, University of Texas at Austin, 1974.
24. Reiser, M. "QNET4 User's Guide," IBM Research Report RA-71, Yorktown Heights, New York, 1975.
25. Sauer, C. H., "Simulation Analysis of Generalized Queueing Networks," *Proceedings 1975 Summer Computer Simulation Conference.*

An approach to simulation of multilevel production systems

by J. F. CLARK and D. M. COHEN
GTE Sylvania
Needham, Massachusetts

ABSTRACT

An approach to modeling interlevel activity in an hierarchical man-machine system or organization is developed and discussed. This approach involves defining a set of command messages for the higher level control subsystem; a complementary set of actions for the lower level production subsystem; a set of feedback messages for the production subsystem; and a complementary set of actions for the control subsystem. This technique tests decision models, management policies and the coordinability of subsystems. Since this paper deals only with one aspect of a general hierarchical model, a brief description of that model is provided. Then the technique is demonstrated using a two level system which has only one subsystem at each level and a small command/feedback repertoire.

INTRODUCTION

Among the approaches to systems analysis, hierarchical modeling promises to be the most effective in dealing with complex multi-level situations. For this reason a comprehensive total system model consisting of a market and a production system has been developed.[1] It was demonstrated that the model could be used to compute total resources required in meeting the demand generated by the market and that the production system could be designed to react to or, to interact with, the market. One of the major aspects of the model was a provision for dealing with interlevel effects within the production system—i.e., to study the effect of the transfer of control/feedback information on other aspects of system operation. This implies a truly bilevel model—one level concerned with production activity; the other level concerned with control activity which is capable of being affected by and of affecting production activity.

PURPOSE

There is a need to take into account the effects of management processes on the flow of work—or else to argue that their effects are negligible. The purpose of this paper is to present an approach to modeling a two-level production system incorporating decision making. It is intended that this approach would be used to explore such areas of fixed (determinate) decision making as decision models, management policies and subsystem coordinability because of the limitations on the flexibility of a computer simulation of decision making processes.

SCOPE

Such a simulation could incorporate a detailed model of a complex production level along with a rich set of choices for the control subsystem and even real-time decision inputs from the person running the simulation. This paper will only deal with a very simple system consisting of a control subsystem and a production subsystem. The concentration here is on the control/feedback process itself and how operations will be modified by it.

MODEL

Even though a complete discussion of the basic model is outside the scope of this article, the following summary should prove helpful (see Figure 1). At the highest—i.e., most general—level, the model consists of a production (functional) system driven by a market. Within the system under study, work is done in response to inputs from the market. Each demand is for a specific product and the transformation from demand to product—from input to output—follows a predefined path through the system. System resources are used as a result of this transformation. The goal of analyzing a system in this manner is to optimize the use of resources while satisfactorily meeting the expected demand.

There is no theoretical limit on the complexity of the work stations which make up the paths through the system. It is considered here that each work station represents a subsystem in the model and that it is represented by a subroutine in the computer code. A work station subroutine is given the information necessary for it to determine how many transformations it must perform and therefore what

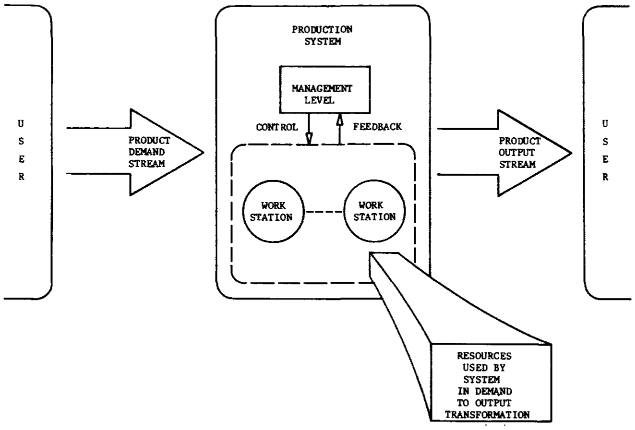


Figure 1—General model for market driven total system analysis

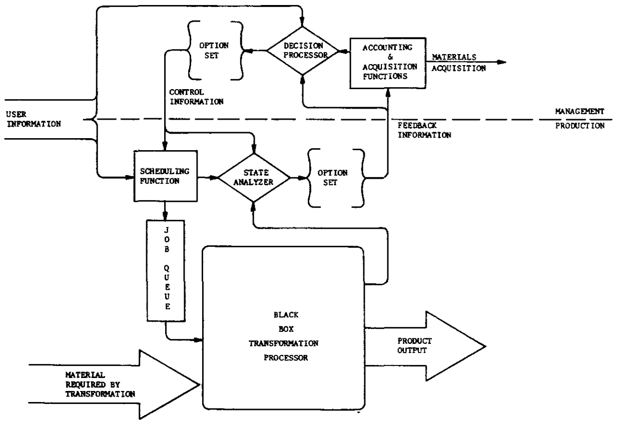


Figure 2—Detail of production system showing interlevel activity

resources it will use. It returns an accounting of the resources used and an assessment of its present state.

From the point of view of the operational subsystem/subroutine, a single level simulation involves executing a relatively unchanging set of operations as the user (i.e., market) inputs are supplied. But in a two-level simulation (see Figure 2), the control information must be scanned and taken into account before the user inputs can be operated on. Additionally, after the transformation has been completed—or sooner, if the situation warrants—a feedback report must be passed to the control subsystem/subroutine for evaluation prior to the next command output.

For the control subsystem/subroutine, the shift from single level to bilevel simulation involves addition of decision making activities to the bookkeeping. In a single level simulation, the Administration subsystem/subroutine only needs to model the paper shuffling required to keep the system running. A two level simulation, however, is specifically aimed at introducing the elements of control and decision making. Hence, the communication of command and feedback information as well as the use of resources in the decision making process must be included in the model. The information passed between levels tells the receiver what options are presently open and thereby affects the flow of the program.

AN EXAMPLE

Consider a simple two level production system that accepts user inputs Y and outputs product X. As a convenience there are only two subroutines representing the control and the production processes respectively. Assume that none of the accounting or manufacturing operations are of any interest and only the following aspects of the system need to be developed:

(a) each input Y_i carries two levels of significance which are designated management and production information respectively.

(b) within the production subroutine is a scheduling function which sets up production of output X_i on a

TABLE I—Available Control Options

Designator	Control Outputs	Complementary action
M1	Complete Order "i" Not Later Than "date"	Reorder Job Schedule to Complete Job "i" by Required Date. Note Conflicts with Already Scheduled Priority Requirements and Provide Feedback Message.
M2	Farm out Orders $i_1, i_2, \ldots i_n$	Delete Jobs $i_1, i_2, \ldots i_n$ from Job Schedule and Reorder Job Schedule. Note any Jobs Already in Progress or Completed and Provide Feedback.
M3	Reschedule Order "i" at Lower Priority	Move Job "i" to a Point in Schedule Where it Does Not Conflict with Priority Jobs.
M4	Delete Order "i"	Remove Job "i" from Schedule. Feedback if Job in Progress or Completed.
M5	Show Status	Feedback Job Schedule Status
M6	No Change	Continue Schedule

TABLE II—Available Feedback Options

Designator	Feedback Outputs	Complementary Actions
P1	Job "i" Completed as Scheduled Using Resources "j,x"	Perform Accounting Functions
P2	Job "i" Delayed Because of Event "j"	Consult Policy Table for Next Action
P3	All Jobs Delayed Because of Equipment Delay "t"	Determine Minimum Cost Alternative and Generate Appropriate Command
P4	Current Jobs Scheduled as Follows	Update Order Schedule Checking for Violations of Priority Requirements
P5	Schedule Conflict Job "i", Job "j"	Determine Minimum Cost Resolution and Generate Appropriate Command

first in/first out basis using the production information contained in Y_i.

(c) the production scheduling function will reorganize the job queue upon receipt of a command from the control subroutine.

(d) the management information component of Y_i can carry priority information or can be a cancellation of order Y_j.

(e) within the control subsystem is a decision function which determines what command output should be passed to the production subroutine based on the management information component and the last status feedback from the production subsystem.

The available control options are shown in Table I and the available feedback options are shown in Table II. Either subroutine can initiate an interchange so a protocol must be established. For example, whenever a perturbation is detected—i.e., management information requiring special treatment or occurrence of a problem event in the production subroutine—an interlevel message is scheduled after an evaluation delay; a second delay occurs before the message is received; after a third delay for evaluation the next message in the cycle is sent. If this last message requires no message response, the simulation moves ahead to the next perturbation. Otherwise, the cycle is continued until a no-op message occurs.

Possible interchanges following the receipt of a "must expedite" order start with command M1—Complete Order "I" not later than "TIME" with the variables I and TIME filled in. The production schedule attempts to comply and sends either P4—Current Jobs Scheduled as follows (order schedule) and M6—No Change, or P5—Schedule Conflict (conflicting orders listed). The latter response requires that the management decision maker resolves the conflict and generates message M3. The production scheduler again attempts to arrive at a workable schedule. Since this interchange of P5 and M3 could potentially propagate itself all the way down a lengthy schedule using up excessive resources, the simulated decision maker must be given mechanisms for resolving such conflicts in a reasonably short time in a manner that reflects actual system policy and practice.

Similarly P3—Equipment Failure all orders delayed—is potential disaster in a tightly scheduled operation. The simulated decision maker must be prepared to react in the same way that the system does (or is supposed to). In this case there is a distinction between the estimated service time provided by the production scheduler to the management decision maker and the service time that is used to simulate the down time for repairs. The decision maker has to act on the estimate provided by the lower level subsystem.

SUMMARY AND CONCLUSIONS

This paper has developed a general approach to using simulation as a tool for studying interlevel effects in organizations. The method is applicable to a wide variety of situations since no assumptions were made about the levels being modeled—other than the existence of interactions between them; no assumptions were made about the complexity of the system being modeled; and no assumptions were made about constraints on the system. The general model which was the basis for this analysis is primarily concerned with multi-level, constrained systems. Other aspects of this approach have been developed elsewhere. It has been demonstrated here that simulation of a true bilevel system—and therefore of any multi-level system[2]—only requires a well defined communication set for each level, a well defined and complementary set of actions for each level, and a communication protocol. All of these elements can be derived from a systems analysis using an hierarchical model such as the Systems Analysis and Integration Model.[3]

REFERENCES

1. Clark, J. F. and D. M. Cohen, "A Production Systems Approach Using a Simulated Hierarchical Organization," *International J. Systems Science*, Vol. 5, No. 5, pp. 425–433, 1974.
2 .Mesarovic, M. D., D. Macko, and Y. Takahara, *Theory of Hierarchical, Multilevel Systems*, Academic Press, New York, 1970.
3. Shapero, A., and C. Bates, Jr., *A Method for Performing Human Engineering Analysis of Weapons Systems*, Wright-Patterson AFB, Ohio, Wright Air Development Center, Technical Report 59-784, September 1959.

Low cost data acquisition and control systems for the computer hobbyist

by RALPH TENNY

Pavco Electronics, Inc.
Dallas, Texas

ABSTRACT

This paper outlines the basic data acquisition system as it might be configured for the home computing system. Allowance is made for data acquisition, interactive control with display, temporary data buffering, cassette storage of data, and computer control of external devices.

Cost/performance trade-offs are examined in each area where a variety of choices are available. For control, a simple keyboard and seven-segment display gives adequate results. Memory requirements are minimized by appropriate choices of data rates, operating system, and design of process control parameters.

The maximum cost savings are possible in the choice of A/D and D/A components, by careful decisions on converter speed and resolution, and on test design. The hobbyist has time/cash trade-offs options not available to industrial designers, thus hardware/software trade-offs can save either time or cash, depending upon which is most available to the hobbyist.

INTRODUCTION

In general, the computer hobbyist interested in instrumentation projects has few kits to choose from if he wishes to build a general purpose data acquisition and control system. Worse, the kits available are not especially flexible in application. The purpose of this study is to review the problem areas he will encounter in choosing components for a home-brew general purpose system, and to give guidelines for choosing components of a custom, low cost system. The discussion to follow assumes the hobbyist has a functioning micro-computer with the following capabilities as a minimum requirement: self-start after reset, bootstrap to cassette loader, and random access memory (RAM) sufficient to handle data and control as outlined below.

Figure 1 shows the basic general purpose instrumentation system, with all system peripherals interfacing directly to the micro-computer (uC) bus. Basic to the system under consideration is an interactive control section; without this, the system would operate only on stored program and would be more properly a data logger.[1] Depending upon the experimenter's budget, the interactive control section can be as simple as a keyboard and display, up to a teletype or video terminal. Careful planning of the system software is essential to retain maximum flexibility; at this level, the command structure is far more important than a sophisticated terminal. A number of kit and modular uC's have broad versatility with six digit or eight digit displays and keyboards consisting of no more than 25 keys.[2] System architecture and software expertise can provide versatility with low cost.

MEMORY CONSIDERATIONS

The available memory has been divided into Control Memory and Data Buffer. It is desirable, but not mandatory for all the Control Memory to be RAM. By means of cassette loading, it is simple to start up the system, and by keeping all programs in RAM it is possible to quickly modify the operating system for different tasks. The Data Buffer can be RAM, shift register or First-In-First-Out (FIFO)[3] buffer, depending upon how the data is to be stored for later study. The total amount of system memory can be greatly minimized if the operating system allows individual data block storage on cassette as the data is received. The choice of RAM, shift register or FIFO will depend entirely upon the incoming data rate. For example, if the cassette system accepts data at 300 baud (30 characters per second), data rates faster than 300 baud require RAM storage. A steady data rate of 300 baud would allow direct storage on tape, with RAM only for temporary storage while each byte of data is formatted for transfer to tape. Perhaps 95 percent of all routine data acquisition applications will have a lower data rate, so that data can be received, formatted and stored in data blocks using a shift register. For example, a 1024-bit static shift register will store 128 8-bit characters; at 300 baud, it takes just over four seconds to dump the shift register onto tape. Between the extremes of 300 baud and 15 bytes per second, the FIFO can replace RAM as a data buffer. This is because the FIFO will accept and transfer data asynchronously at both input and output ports, so long as the average input rate does not exceed the output rate.

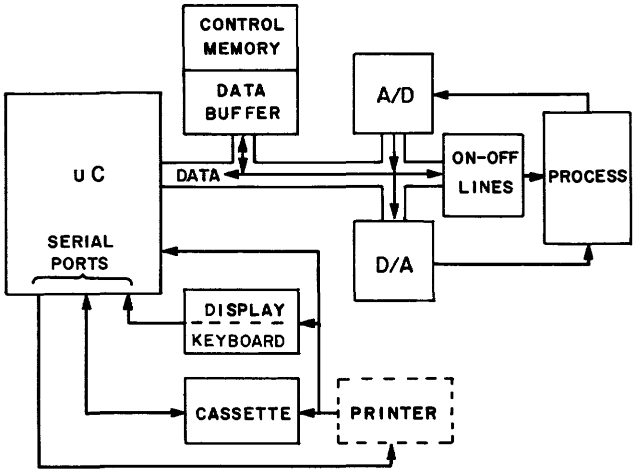


Figure 1—Basic computer-controlled instrumentation system

One special case also dictates the use of RAM for data storage: if the entire block of test data must be manipulated, analyzed or normalized in terms of the total test result, it is usually far more efficient to store the entire block of data in RAM rather than store it incrementally on tape. In the latter case, the tape might have to be played back a number of times before the calculations have been completed. The data storage can be minimized with data compression techniques;[4] the total amount of RAM needed will still have to include workspace area for the data manipulation.

The previous data discussion was simplified by the implied assumption that data comes only in single-byte packets. This is rarely so; even if the output is from a single, 8-bit A/D converter, such isolated quantities are essentially meaningless. One example of the simplest case might be monitoring a single temperature. If temperature variation with time is to be recorded, the test can be arranged so that a measurement is taken every ten seconds. By recording the test starting time, the time of each sample can then be computed and need not be recorded. A much more common instrumentation problem requires recording of two or more variables from each test condition. One parameter will be the independent variable and all the others are dependent variables. (Dependent variables change as a result of changing the independent parameter.) One example: a single-tone test of an amplifier-speaker system. A tone is fed to the amplifier at varying levels (independent variable) and at each input level measurements are made of Total Harmonic Distortion (THD), temperature of the output transistors, and sound level from the speaker(s). This is a case where all three quantities would need to be measured and the computer might monitor THD or transistor temperature and vary input level accordingly. Thus, it depends upon the test whether the independent variable can be calculated (as in the time vs. temperature test above) or must be recorded. Careful test design can therefore minimize hardware, software and memory requirements.

It is now reasonably clear that the cassette operating system (software and hardware combined) should be such that start-stop operation of the tape is possible. The data would be output in a standard recording format such as the Kansas City Standard.[5] The data in the tape output buffer should consist of sync characters (especially important for recording short blocks), ID characters, data characters, checksum characters (or other error detection scheme) and an ending character. As a result, the 128 characters stored in a 1024-bit shift register might represent only a few data points. If 8-bit A/D conversion gives insufficient accuracy, or if a BCD converter is chosen (a number of 3½ digit modules are available), data storage requirements will increase. In general, the uC will handle output from 10-bit, 12-bit and 3½ digit converters as two data bytes. Some tape routines store each byte as two ASCII characters, which could further limit the number of data points stored in the tape output buffer.

DATA CONVERTERS AND THEIR INTERFACES

Much of the cost of the data acquisition system can be in the data converters and their interface circuitry. Fortunately, there are a great number of low cost and medium cost monolithic and hybrid modules available; these have adequate accuracy and interesting combinations of features to make the choice difficult.[6-8] A number of factors affect the decision process; required conversion speed, accuracy and resolution, microcomputer architecture, and location (remote or local).

The two potentially most costly A/D parameters are conversion speed and accuracy/resolution, and the unit prices have fallen rapidly in recent months. Basically, the *range* of conversion speed is determined by the type of conversion—integration or successive approximation. Dual slope or multiple slope integrators with one to fifteen conversions/second and 8 bit resolution are quite inexpensive; moving to 500 conversions/second roughly doubles the price. Speeds beyond two milliseconds/conversion typically require successive approximation, and the speed jumps to roughly 20 microseconds/conversion. Faster conversion speeds are available but are a needless cost unless the microprocessor is much faster than the typical hobby machine. Increased accuracy and resolution jumps the price quickly, and typically slows the conversion at the same time. For example, one manufacturer's price increased 30 percent and speed decreased by a factor of three in moving from 8-bit resolution (.4 percent accuracy) to 10-bit resolution (.1 percent accuracy), for the same type of conversion. The distinction between accuracy and resolution with regard to A/D converters is a topic beyond the scope of this paper. Some recent articles[6-8] have made detailed explanations of A/D and D/A specifications.

The architecture of the microcomputer may exert some influence on the choice of A/D converter. If the converter will be expected to communicate directly with the data bus, the converter will need to have tri-state output lines or must be connected through tri-state buffers. On the other hand, many uC systems have programmable interface devices which allow direct communication with the converter. The same interface device furnishes the address decode function, and some interfaces allow handshake and interrupt capability and can initiate the conversion process with a strobe line.

Figure 2 is a partial block diagram for a system with programmable interface circuits. Typically, these interface circuits reside in memory space (are addressed with memory instructions) and therefore pre-empt some memory addresses. Since virtually all hobby computers address at least 32 kilobytes of memory, this poses no problem for most data acquisition systems. Note that system components shown in Figure 1 will supplement the sub-system shown in Figure 2; Figure 2 merely illustrates how the programmable interface allows a much greater range of A/D component choices. This versatility usually allows lower priced converter components to be used.

If the data converters are remotely located, power must be furnished, control signals must go out and data must be returned. Since remote operation of the data bus is essentially impossible due to propagation delays, the send/receive circuitry which services the remote converter becomes a separate peripheral device. Consequently, the converter should be capable of autonomous operation on a single, low-power power supply and produce a serial data output. Reference 11 details systems which meet many of these requirements.

Because there is a bewildering array of different types of A/D converters which are normally reported, one type is often overlooked. This is the voltage-to-frequency converter (V/F), which has a number of advantages for the computer hobbyist.[9,10] In particular, where conversion speed can be on the order of one second, 12-bit resolution (one part in 4096) is almost routine at very low cost, and 16-bit (one part in 65k) converters are available for about $50. This type of device is ideally suited for remote location in that its output is a pulse train whose frequency is directly proportional to the instantaneous input voltage or current. Its power requirement is relatively low and low power versions are available.[9] At least one ultra-low power circuit using two standard IC's and a single power supply voltage has been reported.[12] Finally, because the V/F is an integrating device, it tends to reject random noise, and has a wide dynamic range of operation.

To avoid a totally rosy picture, the V/F has two major disadvantages: slow speed and the data output stream. Because the output frequency is the parameter of interest, this signal must be counted. This is accomplished by having the uP gate the input to a counter (typically one second time intervals), and then reading the counter output lines as data (see Figure 3). A new problem now exists: resolution of even a garden-variety V/F module is 10 bits. To retain the full V/F resolution with an 8-bit uC, it is necessary to multiplex the counter output lines and read in two data bytes. However, IC counters which multiplex a number of digits into a bit-parallel, BCD word serial format are available. The BCD format is easily manipulated by most uC's with software—decimal arithmetic instructions. If the full V/F resolution is not needed, set the time interval to avoid overflowing an 8-bit binary counter. Besides simplifying the interface, the data conversion time is reduced by the same ratio as the counter gate interval.

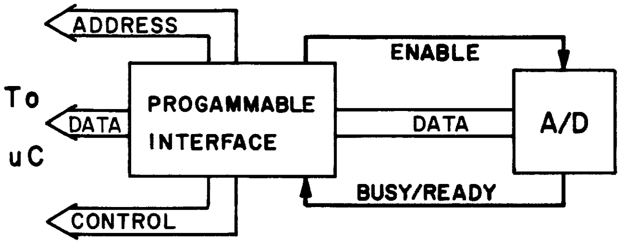


Figure 2—Programmable interface sub-section

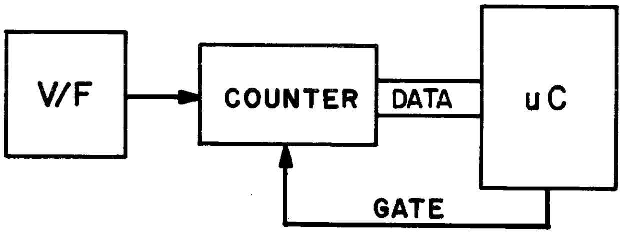


Figure 3—Normal V/F-computer interface

The fact that a V/F is a virtually perfect integrator simplifies measurements such as average quantity or total quantity, (for example, average temperature over a time interval or total energy used by a device). In the first case, a temperature related voltage[13] serves as input to the V/F. A suitably scaled total count then represents the average temperature for the time interval of the count. In the second case, the V/F input must represent the power used by a device under test.

The final components in Figure 1 are the D/A converters and discrete control lines. The discrete control lines have been dealt with in a number of reports,[15-17] but D/A converters have perhaps been slighted in home computing literature. Once again, the recent reports[6-8] give a bewildering array of devices, but the spread of useful features is not so great. To be effective, D/A input data must be latched. Almost universally, low cost D/A modules do not have input latches, regardless of the device resolution. For this reason, the programmable interface (Figure 2) should be programmed as an output port to serve as data bus interface and data latch. Figure 4 shows low-cost alternatives. Figure 4A uses presettable counters (address decode and load strobe needed) to drive the D/A input lines, while 4B shows up/down counters serving the same need. Note the different output patterns (Figure 4C): the presettable counter (quad or hex latches are also suitable) gives a step-function output and the up/down counter version ramps to the final output. D/A converters have two serious faults—glitches and overshoot. A detailed treatment of these problems (cause and cure) is available,[18] but in simplified terms, glitches are the result of unbalanced propagation times in the digital logic circuits and overshoot results from incomplete compensation of the analog output of the converter.

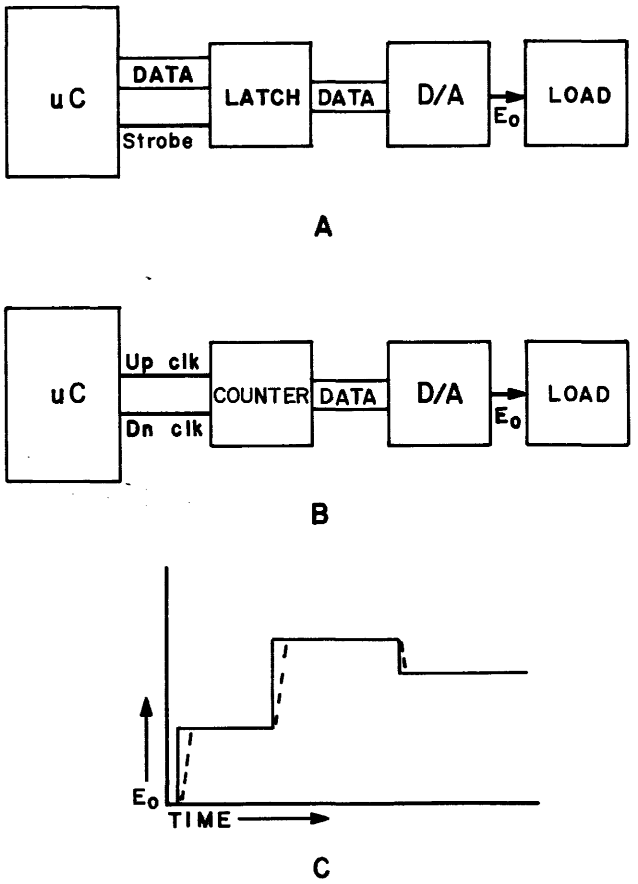


Figure 4—Comparison of strobed vs. clocked performance of D/A's

CONCLUSION

In summary, this paper has dealt with a complete, custom instrumention concept for the home computing system, as represented in Figure 1. The intent has been to guide the designer toward cost-cutting decisions in microcomputer architecture, but with emphasis on low cost hardware. Obviously, if the desired system is to be less comprehensive, the cost drops immediately. It is also possible to trade software for hardware to effect cost savings. Reference 14 gives a number of ideas on hardware/software tradeoff's, but speaks from the standpoint of the cost-effective industrial designer. The home computer, when approached from the hobby standpoint, will have a different economic justification (usually to combine fun and accomplishment in a technical area, at an affordable cost). In other words, the hobbyist may always have to use his personal time instead of cash to accomplish particular goals. His computer may have a 5 uSec cycle time instead of 250 nSec (he can afford to wait), and if he enjoys software, hundreds of hours of program coding and de-bug to save $50 cash is an excellent investment. For the hardware hacker, adapting a published circuit or idea to work with junk-box parts is an exercise in frugality which also pays off handsomely in exercise of creative talent.

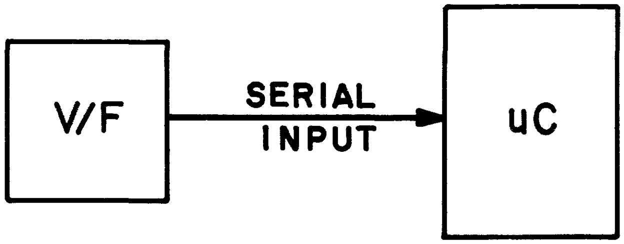


Figure 5—Hardware/Software trade-off on V/F interface

One example of software replacing hardware: let's update Figure 3 to Figure 5. The 10-bit V/F operates at 10 kHz; simply monitor the V/F output with a single input line and use a software loop to increment a totalizing register and a timer. Read the total counts at the end of the time interval, and Figure 3 becomes Figure 5.[14]

If the computer has a programmable timer, the input port of Figure 5 can be an interrupt line. If the timer can also set an interrupt, the computer has considerable time to mind other tasks while measuring a digital quantity. The program uses the V/F interrupt to increment a counter register until the timer times out. In this case, the faster computers will give better absolute accuracy because interrupt service is faster.

REFERENCES

1. "Log data under uP control," *Electronic Design,* 10, May 10, 1976, p. 94.
2. "Low cost design aids keep pace with growing microprocessor field," *Electronic Design,* 25, December 6, 1976, p. 20.
3. *The Memory Data Book,* First Edition, Texas Instruments, Inc. p. 147.
4. "Don't Waste Memory Space," *Byte,* December 1976, p. 58.
5. "Byte's Audio Cassette Standards Symposium," *Byte,* February 1976, p. 72.
6. "Specifying A/D and D/A Converters," *Electronic Products,* November 1976, p. 46.
7. "Designers are Looking Closely at New Monolithic DACs and ADCs," *Electronic Design,* 13, June 21, 1976, p. 28.
8. "Focus on Data Converters," *Electronic Design,* 19, September 13, 1976, p. 68.
9. "Voltage-to-Frequency Converters: A/D's with advantages," *EDN,* June 5, 1974, p. 49.
10. ."Consider V/F Converters," *Electronic Design,* 24, November 22, 1976, p. 160.
11. "Low Cost Data-Acquisition Systems," *Electronic Design,* 24, November 22, 1976, p. 152.
12. "Precision Voltage-to-Frequency Converter uses Only Single Supply Voltage," *Electronic Design,* 21, October 11, 1976, p. 82.
13. "Don't Sweat with Thermocouple Thermometers," *Electronic Design,* 24, November 22, 1976, p. 146.
14. "Designers Must Know When to Make Hardware/Software Trade-Offs," *EDN,* November 20, 1976, p. 289.
15. "Notes on parallel output interfaces in memory address space," *Byte,* November 1975, p. 52.
16. "Controlling External Devices with Hobbyist Computers," *Byte,* April 1976, p. 42.
17. "An Octal Front Panel," *Byte,* May 1976, p. 38.
18. "When D/A Converter Glitches Rear their Heads, Check the Application," *Electronic Design,* 22, October 25, 1973, p. 100.

Diskomania—A small-system floppy disk operating system

by WAYNE SEWELL
Dallas, Texas

ABSTRACT

This paper describes the fundamental structure of DISKOMANIA, a floppy-disk operating system designed for the small system user. The important properties of the system are:

(1) ability to interface to a wide variety of non-standard I/O devices.
(2) a relatively small amount of memory dedicated to the permanently-resident portions of the system.
(3) a simple yet flexible command structure for data transfers and operating system functions.

One thing that can be noted readily about the similarities between computer systems belonging to individuals is that there aren't many similarities. While it would be rash to state unequivocally that there are not two identical personal computer systems on the face of the earth, it would not be out of line to say that they are extremely rare. Lacking corporate funds, hobbyists are scroungers by nature. Two enthusiasts buying systems from the same manufacturer will have identical configurations at first, but will quickly take divergent paths as different priorities and different opportunities to acquire equipment take effect.

As a result of this phenomenon, it is extremely difficult to design an operating system which will work with any configuration, especially when such non-standard devices as Baudot printers, flexowriters, and selectric typewriters are involved.

One solution (?) to the problem is to incorporate into the system the capability to interface to any I/O device in existence, no matter how obscure. This is an ingenious approach, except that the resultant system will take several megabytes of memory and an entire wall of disk drives to implement.

Another problem to be resolved in personal computer operating systems is how much of the system is permanently resident in main memory. The more sophisticated the command processor (or parser, or instruction decoder, or whatever you wish to call it), the greater the amount of memory usurped by it and permanently withheld from the user. Even if most of the actual work is done by nonresident programs called in from disk and afterward deleted, the command processor normally has to be able to determine the proper routines to call, the proper time to call them, and the proper parameters to pass them, based on commands entered by the operator.

Please notice that I said "normally." If a different approach is taken in the relationship between the command processor and the nonresident portion of the system, a powerful, flexible, and memory-economical operating system can be created.

Such a system is DISKOMANIA.[1] The resident portion, the command processor, is comparatively small, yet is quite powerful when backed up by the nonresident portions. In addition, it can easily be modified by the user to accommodate the most bizarre combination of unorthodox I/O devices.

The basic philosophy behind DISKOMANIA is based on UNIX, an operating system for the PDP/11, described by Ritchie and Thompson in 1974.[2] DISKOMANIA is not the full UNIX system, or even a significant portion of it (using UNIX on a micro system is like using a bazooka to swat flies), but it does embrace the basic concept of UNIX: flexibility.

The memory-resident portion of DISKOMANIA, the nucleus, or command processor, is really not very smart. It is unaware of what is going on most of the time. It only sets things up for the more cognizant routines on disk.

The key element in the entire system is the directory. Each diskette contains a sector or more containing the pertinent data for every file on that particular diskette. Each entry in the directory contains the name, type-identifying attribute, and starting address (track/sector) of a particular file.

The command processor actually does very little real processing of a command line presented to it. It does a limited scan, just enough to identify the file names involved, then it searches the directory and retrieves the disk address and attribute of each file.

It is the file attribute which indicates to the command processor what operation it is to perform on the file. The attributes fall into two rough categories: data attributes and driver attributes. Data files are just that: files which contain some type of data. The command processor doesn't care whether the data is pure binary, ASCII, object code, or nothing but zeroes. It performs no special processing on data files other than reading or writing them from or to disk.

The command processor's reaction toward driver files is totally different. A driver file is a relocatable subroutine stored on disk, a program written to communicate with a certain device (such as a paper tape reader) or perform a certain operating system function. When the command processor realizes that it is dealing with a driver file, it stops treating it simply as data. Instead it loads the driver into memory and executes it, although it has no idea what the program does. The drivers perform all I/O in the system (except the disk itself) and all higher-level functions of the operating system.

A DISKOMANIA command is in the form:

> Parameter 1 (Subparm, Subparm, . . .), Parameter 2 (Subparm, Subparm, . . .)

where Parameter 1 and Parameter 2 are file names, each of which is stored in the directory of one of the drives currently on line. The files are not necessarily on the same drive. Sub-parameters may or may not be present.

The data flow is always from Parameter 1 to Parameter 2. This is true whether either file is a data file or a driver file. The sub-parameters are not acted on by the command processor; it only notes whether or not any are present and records the addresses in the command buffer at which they start.

Parameter 1 is the input parameter. It could be a data file, in which case the command processor only reads it. On the other hand, it could be a driver file, which means that the command processor loads it into memory and permits it to handle the generation of the input from an external device.

Parameter 2 is the output parameter. Similarly, it can also be either a regular data file or an output driver file, a program to output to an external device.

Unsurprisingly, the length of a disk block, which is equivalent to one sector, is the basis of all data transfers throughout the system. There is a block of main memory permanently set aside for this purpose. This buffer acts as a common interface between Parameter 1 and Parameter 2. The 1st parameter inputs data into the buffer for the 2nd to dispose of.

As the command processor identifies the file name for each parameter, it sets pointers to the starting track and sector if it is a data file and loads it into memory (making a note of the starting address) if it is a driver file.

After pointers and/or entry points are established, the actual data transfer begins. The command processor goes into a loop in which it causes data to pass from Parameter 1 to Parameter 2 until an end-of-file is reached on Parameter 1.

On the first pass through the data transfer loop, the command processor once again checks the file attribute, which was stored in memory during the identification phase. If the Parameter 1 file is a data file, the command processor reads the first sector of the file into the data buffer previously mentioned. After the read is finished the pointers are updated so that they point to the next sector in the file. If, however, Parameter 1 is a driver, the command processor branches to it, not knowing or caring what type of input device it interfaces to. All the main program knows is that the driver is a routine that will accept data from a device, move it into the common data transfer buffer, and return control to the mother program when the buffer is full. Note that the final result of Parameter 1, whether the file is a data file or a driver file, is the same: a memory buffer, exactly the length of a disk sector, containing input data.

At this point, Parameter 1 has completed its portion of this cycle of the data transfer. It does not care what happens to the data it has acquired. The command processor now turns to Parameter 2 and follows the same procedure in reverse. If Parameter 2 is a data file, the data in the transfer buffer previously filled by Parameter 1 is written to disk. If it is instead a driver, the mother program branches to it, confident that it will output the data in the common buffer and return when finished. Once again, the main program has no idea what the driver actually does. In addition, Parameter 2 doesn't care how the data got into the interface buffer; it simply takes what is there and writes it.

After the data in the common buffer has been disposed of by Parameter 2, the command processor repeats the procedure. The common buffer is filled by Parameter 1 the same as before, either by reading from disk the next sector in the data file or by re-executing the input driver, and Parameter 2 writes it to the next sector of the output data file or outputs it to an external device via the output driver.

This loop continues until an end-of-file is reached on Parameter 1. A flag is set when this condition is detected, either by the input driver, or by the command processor itself (in the case of data files). This flag tells the command processor that the operation will be completed at the end of this pass through Parameter 2, at which time an end-of-file will be placed on Parameter 2 and the driver(s) will be deleted.

The most obvious advantage of DISKOMANIA is the tremendous flexibility, even with a comparatively simple command processor. Parameters 1 and 2 are completely independent of each other and are also mutually transparent as far as data transfer is concerned. One puts data into the common buffer and the other removes it, neither caring how the other accomplishes its end of the operation.

If both parameters are data files, the end result is one file copied to another on disk. If Parameter 1 is an input driver and Parameter 2 is a regular data file, the input device writes straight to the disk, the data automatically formatted into sectors. Conversely, if Parm 2 is the driver and Parm 1 is the data file, the entire file is dumped in its entirety to the output device, deblocked and transmitted in a continuous stream. If *both* parameters are driver files, a sort of in-one-ear-and-out-the-other function is accomplished. The input driver places data into the common buffer and the output driver sends it to the appropriate device without really involving the disk at all.

Each of these four functions is invoked in exactly the same way: by entering two alphanumeric labels on the command line. The function actually performed is controlled by the file attribute assigned to each of these labels in the directory.

Of special interest to the small system owner is this important fact: in order to customize the operating system for his own configuration, he does not have to change the entire system: he only has to change individual device drivers. If a driver supplied with the system will not work, the user can modify that one driver and store it under the same filename and the command processor will never know the difference. Conversely, the original driver could be retained and the modified driver stored on disk under a *different* name, enabling the user to access either version at any time. New devices, standard or unorthodox, can be added at any point by writing new drivers and storing them on disk to be accessed by name like any other driver.

When the command processor passes control to a driver, it assumes that the driver performs some type of I/O operation via the command data buffer. In truth, however, the command processor is in total ignorance of what occurs inside the driver. Taking advantage of the command processor's tunnel vision, we store special non-device drivers on disk to perform the higher level operating system functions and they are executed just like any other driver.

For example: we wish to delete a file named, unimaginatively, FILE. We have previously stored on disk a non-device driver containing the file deletion routine under the name DELETE. Upon receiving the command line

FILE, DELETE

the command processor searches the directory and determines that FILE is a true data file, so it sets the pointers to the first sector. Further investigation identifies DELETE as a driver file, so it is loaded into memory like any other driver. During the Parameter 2 portion of the data transfer loop, the command processor branches to DELETE, naively thinking that DELETE will dump FILE to an output device. Instead FILE is deleted from the system and removed from the directory. Before returning control to the command processor, DELETE sets the end-of-file flag to end the operation. The important fact is that *the command processor never knew the difference.* It has no way of distinguishing a device driver from a system driver. ALL DISKOMANIA system functions are performed in the same way, by deceiving the command processor.

Another valuable feature of DISKOMANIA is that it never takes any more memory than is absolutely necessary to perform a given function. The command processor allocated just enough memory space to contain the temporary drivers and immediately returns it back to the user when they are no longer needed. The command processor itself, the main resident routine, is comparatively small due to its simplemindedness.

When the operating system is initialized upon power-up or reset, the resident portion is read from disk and loaded into the maximum available RAM addresses, placing it at the very top of read-write memory. It then loads in the permanent drivers, SYSIN and SYSOUT, which handle input and output respectively, between the operating system and its command terminal. These resident drivers are loaded immediately ahead of the command processor in memory, at the next lower series of memory locations. The command processor and the two permanent drivers make up the resident portion of the system. Much of the time, the amount of memory allocated to these routines is the total used by the operating system. This is the case when both Parameter 1 and Parameter 2 are both data files and no temporary drivers are required. Drivers, if required for either or both parameters, are loaded into memory just ahead of the resident drivers, still allowing the operating system to control only as much memory as is actually necessary and to reside in one contiguous cluster of modules. After the temporary drivers complete the function for which they were loaded, they are immediately deleted, and the operating system again shrinks to the resident portion, returning the space the drivers occupied to the user. There is a control field in the operating system which contains the address of the boundary between the operating system and user memory. The user can test this field at any time to learn how much memory is currently being used by the system.

The I/O drivers are not restricted to being accessed by the command processor only; any device driver in the system can be loaded and executed by a user program just as easily by calling the directory-search and driver-load subroutines in the command processor. This makes it possible to write applications programs that are virtually device independent.

It is also possible to load and execute the non-device drivers, those that perform operating system functions, from a user program, but this is not advised without a detailed knowledge of how the driver works and how linkage to it is accomplished. In fact, it is perfectly permissible for any driver, device-type or not, to load any other driver with the same restrictions. This can continue for multiple levels as long as memory lasts.

In conclusion, it is very easy to see how DISKOMANIA can be implemented on a very small system with only a single mini-floppy and a comparatively small amount of main memory, and yet run with a surprising amount of power, flexibility, and capability for customizing. It can easily be adapted to non-standard devices, even the operating system control interface (i.e., a totally Baudot system). Flexibility, economy of memory, power—for what more can a small system ask?

ACKNOWLEDGMENT

I wish to thank Dr. Brian Johnson, University of Texas at Dallas, for my introduction to the UNIX philosophy.

REFERENCES

1. Soon to be available from PerCom Data Co., 4021 Windsor, Garland, Texas 75042, (214) 276-1968.
2. Ritchie, Dennis M., and Ken. Thompson, The UNIX Time-Sharing System, *Communications of the ACM*, Volume 17, Number 7, July 1974, pp. 365-375.

Neighborhood computer stores—The answer to microcomputer marketing

by PAUL TERRELL
Byte Incorporated
Sunnyvale, California

ABSTRACT

Fifteen years ago, only an eccentric individual would consider assembling a computer at home. The technology simply would not permit it. Today, the streams of technology and retailing have met. Where the tributaries come together we have a unique and growing concept, the neighborhood computer store. As represented by the Byte Shops (now numbering 40 world-wide), this new concept brings technology and qualified persons to interpret that technology into a convenient and informal format that anyone can understand and participate in. The result can only be a greater demand for the ever more affordable computer technology of today, and a more personal and powerful interest in computer operation and applications not only in the home, but in small businesses. The neighborhood computer store may soon mature to become the only reasonable retail outlet for manufacturers of microcomputers and peripheral products.

INTRODUCTION

Fifteen years ago, if someone had said he was putting a ~~computer together at home, we would consider him owner~~ of an overactive imagination. We might have thought he'd been watching too much Saturday morning TV with his kids—that the Jetsons and Star Trek were taking over his mind. Even if a person had wanted such a do-it-yourself project, he would need to be independently wealthy, be able to control his air to a clean and perfect 68 degrees, absorb an astronomical power bill, and sacrifice an area the size of a large living room to the venture. Even with the above, there would be no way to make proper use of all the bulky equipment to justify even an eccentric millionaire's time and money. In other words, it would have been virtually impossible.

As the years rapidly passed, and compounding technological advances ensued, computer hardware became less bulky and began to sweep a complete spectrum of applications. Some systems grew in complexity, becoming extremely sophisticated with storage and manipulative power to computerize a nation of tax records. Other developments went the opposite direction, becoming smaller and considerably less complicated.

One event, however, seeded the idea of putting computer power in the hands of someone other than trained technicians. With the advent of computer assisted instruction and the concept of timesharing, the mystique was finally broken. Sharing the computer among many users (timesharing) made a large central computer (mainframe) seem to slow down and patiently wait for its human user to type in or respond to information. The beauty was that the incredibly high speed machines could service multiple users without the human element feeling neglected. Timesharing made better and more economical use of large machines, but more importantly, it satisfied the psychological need for a personal rapport between man and machine. We no longer feared the beast and could welcome it as another tool for mankind instead of the awesome BIG BROTHER.

The trend toward miniaturization and simplification was greatly accelerated by the space program, which created a definable need for smaller computers. So we began to speed up technology, miniaturize and simplify. Computer technology was allocated enough funds to enable speedy developments to meet the space age needs.

Then, in the early 1970's, the semiconductor industry ~~came to meet the computer people more than halfway. Intel~~ Corporation designed a single chip—the 8008—an 8-bit microprocessor which unleashed a whole new era by solving the money and size problem that had previously stood in everyone's way. That leap forward in 1972 paved the way for putting computing power in the hands of anyone who wanted to give it a try. Through the development of comparatively less expensive minicomputers, microprocessors and peripheral equipment, the results of that rapid progress is what we're involved in today.

During this same period, another phenomenon was taking place. With apparently no organized effort, a great number of technical and professional people from within the electronics industry were becoming increasingly interested in the new small computers. On the side, at work, they were inventing games like space war and computer chess, compliments of their employers, justifying such antics as "demonstrator programs to demonstrate versatility and design capability. . ." But, at the same time, they were having fun on a minicomputer and CRT (cathode ray tube). More than a few programmers attempted to computerize their personal tax records at considerable expense to their resident universities. It wasn't any wonder that a desire for a toy of one's own was starting to gel. About this same time, electronic kits and so-called "computerized games" were sweeping the consumer market, but they lacked room for individual creativity and interesting application on the part of the user.

FIRST MICROCOMPUTER KIT

When the first microcomputer kit, the Altair 8800, was introduced in December of 1974, the stage was set for the hobbyist and personal computer marketplace. Even then we in the computer marketing business were not completely aware of the incredibly perfect timing that was about to combine with the ideal product and combust. A ripe and hungry public answered the microcomputer kit advertisements, with COD's and money orders—sight unseen. As is too often the case, many manufacturers had stepped into this new market, but they weren't ready for the onslaught and the filling of mail orders was delayed. That computer-starved public was left with fingers crossed, looking at their deflated bank balance, hoping for delivery of their new "toys."

In 1975, I was a sales representative for MITS, then the leading manufacture of computer kits. My partner and I were responding to the unusually high number of inquiries from the slow leads, and we found that our customer profile had dramatically changed. Payment was often by personal check or cash—not company P.O. These interested and frustrated hobbyists were trying to deal directly instead of by mail, to try to ensure prompt delivery of their kits. We realized our customers were buying kits for their personal use. To ease the guilt feeling of taking money without delivering the product, we ordered ahead and stockpiled a supply. When word got out that we not only had those previous kits in stock, but some software packages along with them . . . well, we were never again to knock twice and put our foot into the door for a normal sales call.

SALESMANS' NIGHTMARE

It would seem like a salesman's dream to have the customers flocking to you—but not quite. Our office literally turned into a mob scene as people browsed through the literature shelves waiting to use the demo machine. A month's coffee supply disappeared in four days. And the last straw was the sight of a Modesto mother and her 14-year-old son at our door at opening time with a broken computer. They had been waiting since 7:30 a.m.

It was utter chaos and our little office setting was becoming absurd. Since we were doing such a great business in obscurity, we wondered what would happen if we opened up a retail shop and hung out a sign. Byte means the measure of a unit of information, and since we were knee deep in the world of computer buzz words, we settled on the name "Byte Shop." We considered it eyecatching and rather significant. Except for the few people who think we're a sandwich stand, the name has done well for us. For those uneducated others, we ought to serve snacks. Our first retail store opened on December 8, 1975, in Mountain View, California, in the heart of "Silicon Valley" near San Jose. (Silicon Valley is the appropriately nicknamed area that houses one of the largest concentrations of electronic companies in the world. It got the name from silicon, the basic substance for producing semiconductors.)

Aside from creating a stir with the local press, we even managed a modest profit in our first month of operation. We also discovered we had the equivalent of a trade show 365 days a year for our principal's products. This seemed like a natural way to overcome the increasing cost of selling to the end-user, and our customers were intrigued with the thought of visiting a computer store. After three months of doubling our gross sales, we were "discovered," and on March 2, 1976, opened Byte Shop #2.

TIME FOR GOALS

As we appeared to be growing and making a profit, it became time to set up some kind of basic corporate structure and map out our goals. We settled on the name Byte Incorporated as the central distributorship, doing all the direct wholesale buying from manufacturers and stocking the Byte Shops with preferred product lines as retail outlets. Our goal was simple and modest—tomorrow, the world.

The structure and tie-in between Byte Incorporated and the Byte Shops is simple. Byte Incorporated buys in large quantity directly from the manufacturers at an OEM (original equipment manufacturer) discount. We also manufacture some products ourselves, such as the Byt-8 kit. Other products are designed to our specifications and carry the Byte label, such as the Byte-File. We purchase all the inventory necessary to completely stock all Byte Shops at wholesale discount prices, buying in volume . . . a discount for which each individual store could not qualify. Byte Incorporated then sells to the stores and charges a seven percent handling fee. We can stock a new store with $28,000 worth of merchandise, and the cost to the store owner is only about $20,000. Average inventory turnover for one shop is nine turns per year. The Byte Shop dealership agreement limits the loss to the shop owner through a merchandise buy-back agreement. I might add that this "bail out" provision has never been exercised, but we feel it's an excellent safeguard. The formula seems to be working, because we went from 0 to 40 stores in the first year and should have approximately 100 by July of 1977—all thriving and many expanding.

WHO'S RUNNING THE STORE

As with anything new, exciting and so potentially profitable, everyone wants to get in on the act. My desk is piled high with requests from all over the U.S. and a dozen foreign countries to open stores. As I have indicated, each store is owned and operated by the individual business man. Byte Incorporated acts as his distributor, offering him economical advantages, support assistance, cooperative advertising and a proven successful format. Even though he carries our name, he is an independent dealer, much like the manner in which some national service station dealerships are set up. Since the format is a proven winner, we have a definite reputation we wish to maintain, so it is to our mutual advantage to screen new shops and their prospective owners and management carefully to assure that each Byte Shop is successful.

The preferred owner fits the following simple profile: He should show a keen business sense and an appreciation of the consumer computer marketplace. It is not necessary that the owner possess technical computer knowledge as long as he wisely chooses such expertise for his management. The best manager is someone who has both technical skill and an ability to work well with customers. The ideal person to act as guiding force in a Byte Shop is a customer engineer (CE) from the computer industry. A customer engineer services, advises and often sells to his company's customers and is the real backbone of a sound sales and service program. Successful customer engineers usually understand and enjoy the technical side of computers, but not to the exclusion of the personal customer contact. It is this combination that makes him valuable as a potential store manager.

The ideal potential manager works as a customer engineer for a big conglomerate. He is frustrated having to be under the control of the large company and harbors a strong desire to be a proprietor. He enjoys working on computers and possesses considerable technical skill and creative ability. Most important of all, he enjoys working with people and likes helping them learn and develop their creative interests.

By separating the profiles of owners and managers, I am not saying that the two cannot be combined. If someone has the unusual combination of abilities—technical skill, personality and business sense, and can establish a firm financial base, then it could probably work well. More than likely the customer engineer profile could be met, and with bank financing and a good business advisor, could establish a workable and successful business setting. Byte Incorporated needs to be assured that all the bases are covered before it OK's use of its name on a new door. Our success rate is still 100 percent and we intend to keep it that way.

If all the personnel criteria are met, then we get down to the financial details and choice of location. A Byte Shop can be opened for approximately $30,000, depending on location. This includes the $20,000 worth of initial inventory purchased from Byte Incorporated. The remaining $10,000 is for prepaid store rent, furniture, leasehold improvements, sign, salaries, phone, operating capital, and the one million little extras necessary to open. Nobody can guarantee short or easy hours, but we can tell you that every shop owner and manager so far has enjoyed it immensely and profited considerably, both in dollar return and personal growth.

MARKET STRATEGY

Each Byte Shop has a common goal—to provide a service to the public and to make money for its owners. With the opening of each store, we see more and more retail professionalism. The physical format that we advocate, and the one that is proving especially successful, is not like our original start-up stores, which were rather bleak. Future Byte Shops will be warm, comfortable, and carpeted with attractive comprehensive displays. Each store will have three sections: one, a sales display of small peripheral products such as boards, cards or interfaces; the second, a section for hardware display; and the third section for books and periodicals. Each store has, and always should have, both a demonstration area with working sample machines, and a work room. These two features serve very important functions. They permit the customer and store personnel to interact while having the product in hand. If the customer needs to learn the basics, the demonstration room can answer many questions.

The work room is where technical problems are tackled. A customer and store technician can lock brain power and soldering iron to repair a faulty CPU (central processing unit) board or test a new I/O (input/output) device. The physical set-up described above enhances the philosophy that we feel is at the heart of the Byte Shop movement—service.

THE IMPACT OF EDUCATION

Education is probably the most significant service we can offer, since the man on the street is basically unaware of the microcomputer revolution taking place all around him. His curiosity is stimulated when he sees a sign describing the Byte Shop as an "affordable computer store." Until now computers were for business and institutions and were anything but affordable. When he walks into the store, he finds a color television, much like his own, with a computer attached, playing color graphic programs loaded from an audio cassette tape recorder. The scene looks very much like his home entertainment set-up in the family room. An audio amplifier and speaker come into view as the sound of digital music fills the air. Slowly the idea begins to form that computers may be meant for him, too. The education process is about to begin. First, he purchases books and magazines and possibly enrolls in a Byte Shop-sponsored weekly course on the introduction to microcomputers. The computer bug has bitten and "computeritis" quickly sets in. The victim's old life style is now of short duration.

The most significant service is the "handholding" that begins even before the unit is purchased. "Which computer should I buy for my application?" is the most asked question, and the answer has to be right for the store owner's future peace of mind. A computer that can't be expandable to meet the needs of someone with visions of grandeur will only create problems at a later date and the conflict will be face to face, over the counter, Byte Shop manager to customer, and not through the mails.

When the purchase is consummated, the hand-holding becomes more technical. Purchasing of kits may require numerous helpful hints on assembly, whereas buyers of assembled units need less attention—providing the unit works well. Whichever the case, it is always the responsibility of the shopkeeper to make sure the machine works. A few statistics at this point may be of interest. Of the thousands of kits sold to date by the Byte Shop, *no one has ever had a bad CPU chip*, and not more than 10 percent of the purchasers have come back to the store for assistance in assembly or troubleshooting. This indicates either quality products or that our customers to date have been a tough breed of cat. Needless to say, we stock only products which our stores can back up with confidence. A solid manufacturer's guarantee helps.

Service after the sale is an extension of that initial handholding during the selection and assembly process and is every bit as important to both customer and shop. It is during this time that the customer concerns himself with his personal application, and is more likely to require advice and additional equipment. Our credibility is tremendously important. If our advice leads to expanded sales, then terrific, but we aren't there just to push equipment—we're supporting a whole idea and stake our reputation on every bit of advice. With this philosophy, we can't afford to lead a customer down a rosy buying spree of non-necessities. Fortunately, we are becoming noted for our credibility.

For that new prospect we lured in off the street, what started out to be a modest investment in home entertainment is taking shape to compete with the computer center at work, at a fraction of the cost. The price of a microcomputer, which I will define as CPU card, power supply and chassis, ranges from $350, and is expanded with memory cards and I/O's from $100. When asked to comment on the computers we sell in the Byte Shop Computer Stores, my answer is "higher level languages playing on hardware for less than $1000." This implies eight thousand words of memory and an input/output interface along with the computer. The average home computer is a $1,500 investment—easily cost comparable to investments in home stereo systems and numerous other hobby fields.

THREE ASPECTS DEFINED

There are three definite aspects to any computer exercise—hardware, software and applications—all of which should be addressed. Many enthusiasts concentrate on just one—unfortunately neglecting the others. If total three-phase investment is impossible, then one can cover the bases by substituting some packaged products. But by becoming involved in all three, one learns the secret of how to indulge in a full and enjoyable dose of computer mania. First comes the hardware aspect, which we have already discussed. If a customer's talent doesn't bend in that direction and he can afford ready-made products, the customer can get right down to phase two—software. The electrical genius that makes his machinery almost from scratch from IC's, wires and boards can pleasantly forestall this step for as long as his tinkering continues. But when everything is up and running it comes down to actually doing something with the creation.

Software work turns many beginners into creative fanatics. Given this type a book on BASIC, coding sheets, a keyboard, and a printer or video display and he's off and running. Just throw him a sandwich and some new pencils periodically, let him out for airing on Sundays, and you have a happy new convert. For those less creatively bent, packaged software is now available, on cassette or in books. The software aspect has been long underplayed. No matter how great the hardware, it is useless without good software.

Fundamentally, the whole discipline of software is the problem. There exists a lack of real feel for how to write good software correctly. The tremendously fast-paced progress in hardware made accompanying software a sorry step-child. Good inexpensive software packages will take time. Unfortunately our market is hungry for it now. One of the solutions is for our customers to self-educate themselves in BASIC to enable "do-it-yourself programming." Byte is involved in providing comprehensive classes in microprocessor programming and feels such a step will not only help solve the problem, but further provide more independent creativity for our customers.

The fact that packaged software can be easily duplicated for distribution dictates that it begins to be offered cheaply for mass distribution. If a package is inexpensive, the user will pay the price in order to get documentation and to be on mailing lists for updates. At the same time his conscience will remain clear.

The final aspect is application. Everyone wants to build a better mousetrap and now the chance has come. The applications are as practical or as futuristic as the mind can conceive. Man's creativity is the only limitation, and that is what is most stimulating about the whole field of personal computing. Typical applications run the gamut: for home use, menu planning and shopping, household budget and income taxes, homework and computer assisted instruction. Anything that can be controlled electrically can be run by a computer. So most appliances can become programmable and automated if you so desire. For recreational enthusiasts there are a myriad of games in existence with an equal number for one to self-invent—music synthesizers, computer-run ham radio stations, model airplane and electric train control are a few examples. The *Star-Trek Game* is tremendously popular, and that can be just a beginning. All those games you play for 25¢ a go can be put on a private microprocessor for the whole family to enjoy.

Most of those unique applications make their way back to our Byte Shop counters. Ten percent of the people who come into the store are there to sell their creations back to us, not to buy. They represent themselves, and what they have to offer is a better widget, designed in their back bedroom or garage. And, be it hardware or software, until the day of the retail computer store, there existed no outlet for that creativity. Two hours worth of programming effort on our store demonstration computer netted a Northern

California programmer over 100 sales of his program to date, at $15 a copy.

The other 90 percent of our customers come from all walks of life: a housewife buying a Christmas present for her husband in real estate; a college student; a programmer who always wanted to own his own machine; the guy who has everything; and the girl who is giving up boys for toys. Our present customers are often associated with the electronics industry in some fashion. With CB radio making everyone more familiar with electronics, the trend is toward much more wide-spread popularity.

Computers in the home will unleash talent and creativity of the magnitude required to support the intelligence revolution, and the neighborhood computer store, acting as liaison between the electronic manufacturer and the consumer, will be the focal point of activity. Computer clubs are forming, with store owners in the center of club activity—a vested interest since the growth of such clubs is significant to their profit or loss. Introductory courses sponsored by the stores will supplement community education, primarily because there will be more computers in the home than in the schools. The day of the home computer is now.

BYTE INTO THE FUTURE

The most exciting future for Byte Shop owners is in the field of business. Anyone in a small business could use a low cost microprocessor-based computer system. Stock market simulators, inventory control, updating of mailing lists, personalized form letters, bill collecting, information retrieval and computer assisted design. Every doctor and dentist could use one for patient records, billing, and insurance forms. Lawyers, judges and researchers could save hundreds of hours by computerizing access to information. The hobby or personal computing marketplace is fun, but from a potential sales viewpoint, the small business applications are really exciting.

This brings us to taking a realistic look at the future of the neighborhood computer store. Right now it would appear, on the surface, as though computer stores might remain the place for hobbyists to gather and kibitz, buy a goodie or two, shoot the breeze, or get a problem unraveled. That will, of course, continue to be one of their important functions. We all love to sit around the old IC cracker barrel and swap CPU stories or boast the latest application tale. But the Byte Shops will mature to be much more than that.

We can all recall when calculators were bought directly from manufacturers, then slowly became available in retail outlets at such exorbitant prices that only wizard engineers and mathematicians could afford them. Technology befriended the consumer and reduced the price such that you can now pick one up at the drugstore and give it as a stocking stuffer to a little kid.

No one today would consider buying a basic calculator directly from the manufacturer, mainly because there comes a time in marketing when it simply isn't worth a company's time to deal directly with a consumer. And therein lies the future marketplace for the neighborhood computer store. It will be the only reasonable retail outlet for manufacturers of microcomputers and peripheral products. The only major competition tomorrow may be similar to that which stereo equipment stores experience—department stores carrying computers. But just as there is more consumer safety in dealing with specialty stores for stereo sound equipment, the consumers will probably prefer the expertise they will get only from neighborhood computer stores.

The challenge is that the computer stores like the Byte Shops will have to be very good at what they do. A hobbyist may be forgiving if his toy doesn't work quite right for a few weeks, but the computer that becomes the backbone of a small business must stay functioning or the small businessman starts to see dollars fly out his window. Good service is the key.

Service has been nearly as big a problem in the computer field as good economical software. The problem can be solved by competence. That's why we're so particular about who runs a Byte Shop. We know that companies are full of good competent technicians who can keep a small business system whirling and churning out data with one hand tied behind their back. Such people have moved mainframe mountains—a little microprocessor or terminal is a "piece of cake." The 10 years he's been training and tinkering will all pay off as he acts as diagnostician, friend and A-1 repair mechanic to his clientele. And if he runs into problems he can't solve, he can hire his old buddy from down at the plant. There's always a new batch of brilliant engineers around—and, quite honestly, the equipment we're handling isn't that complicated.

If the computer industry gets its proverbial act together and standardizes hardware the way will be clear for retailers to enter the service world with little trepidation. Good applicable guarantees from manufacturers and a healthy supply of interchangeable parts will have to be readily available to equip the service person with the tools he needs to properly service customers. Hopefully we're now on our way to such a reality.

PRESENT SALES AND LONG RANGE PROJECTIONS

At the Byte Shops our best sellers in microcomputers are the 8080 based kits with the standard S100 bus. There are a variety of good selling lines including our own BYT-8 machine. The important thing to note about existing microcomputers, however, is the standardization problem that we have just discussed. Our customers want to use peripheral and support equipment interchangeably, thereby assuring a better dollar buy. We're getting there, hopefully. Other big sellers are the TV typewriters that either fit into your computer or can be interfaced later as a stand alone unit. Books and magazines are moving at a staggering rate, for example: Vol. I, Introduction to Microcomputers by Osborne . . . has sold 10,000 copies through our stores.

Our volume of sales generally exceeds even optimistic projections. Nothing ever sits long enough to collect dust on either warehouse or store shelves. As a result, I hesitate to make future sales projections. We can definitely see trends though. In memory we've gone from 4K to 8K and 16K to 64K is around the corner. There appears to be a movement toward the fully assembled and tested machines. As good software becomes more available, our customer's interest will probably lean more toward applications activities. That software will be available primarily in cassettes for easy input on home tape recorders. But with hardware costs quickly coming down, there should be good affordable floppy disk systems for the hobbyist as well as the small business user. In general, the trend is for both hardware and software to evolve from the developmental "hobbyist" state to the simplified, easy to operate "consumer" stage.

Right now the profit is available. The Pasadena store gross sales in its first three weeks was $26K. Portland: $14,500 in the first month. Mountain View, our first store and going without publicity or advertising whatsoever, was $6,400 in virgin territory. This thing has caught hold so fast that Jeff McKiever, our dealer in the Phoenix area now has three stores in Arizona and is planning others. Dick Moule in Lawndale, California, expanded to Westchester, and the new Byte Shop in downtown Tokyo is going great. We have little doubt that the personal computer market will insure our future.

A CONCEPT THAT BECAME A COMPANY

The time frame has been short between the technological availability of hobbyist computer components and programs, and their commercial availability. The Byte Shop, like the food super market, simply was a natural merchandizing evolution whose time had come. When technology and marketing meet, you will always find great sociological interaction at the intersection of the two lines that have made mankind thrive—the lines of innovation and commerce. That's the point where Byte took hold.

1977 NATIONAL COMPUTER CONFERENCE COMMITTEES

PROGRAM COMMITTEE

Chairman

Robert R. Korfhage
Southern Methodist University
Dallas, TX

P. Bruce Berra
Syracuse University
Syracuse, NY

Lori Capadonno
Bell Laboratories
Whippany, NJ

Karen Duncan
Medical University of South Carolina
Charleston, SC

Roger M. Firestone
Sperry Univac
St. Paul, MN

Frank Hubans
General Dynamics
Forth Worth, TX

Richard Lott
Bentley College
Waltham, MA

Betty Maskewitz
Oak Ridge National Laboratory
Oak Ridge, TN

Roger L. Mills
TRW Systems
El Segundo, CA

C. V. Ramamoorthy
University of California
Berkeley, CA

Eugene Smith
U.S. Department of Agriculture
Beltsville, MD

Raymond T. Yeh
University of Texas
Austin, TX

Jean Yue
Cyprus, CA

FINANCE COMMITTEE

Chairman

Edward V. Resta
E-Systems
Dallas, TX

Earl F. Hale, Jr.
Carrington, Coleman, Sloman,
Johnson and Blumenthal
Dallas, TX

COMMUNICATIONS COMMITTEE

Chairman

Norman P. Teich
Teich Communications Co.
Dallas, TX

Andre P. Beaupre
Strayton Corporation
Wellesley, MA

Larry V. Beckman
Tano Corporation
Metaire, LA

G. Thomas Catherines
Mohawk Data Sciences Corp.
Parsippany, NJ

John A. Dillon
General Automation, Inc.
Anaheim, CA

Thomas H. Edwards
Sycor Inc.
Ann Arbor, MI

David L. Flack
International Marketing
Communications, Inc.
Denver, CO

Francey Freeman
Motorola Data Products
Carol Stream, IL

Daniel Fullerton
Texas Instruments, Inc.
Houston, TX

Ann L. Harrell
University of Texas Health
Science Center
Dallas, TX

George Harrison
Innovative Electronics Systems
Miami Lakes, FL

Rick Johnson
MRI Systems Corp.
Austin, TX

Jerry L. Kalman
Honeywell Information Systems
Phoenix, AZ

Margie Kimbrough
Management Science America, Inc.
Atlanta, GA

Connie Magne
Intel Corp.
Sunnyvale, CA

Ruth M. McQueen
Amarillo College
Amarillo, TX

Kent R. Nichols
Control Data Corp.
Minneapolis, MN

Carol J. Richardson
Hill and Knowlton Inc.
Dallas, TX

Dennis L. Sullivan
KeyTronic Corp.
Spokane, WA

Jean Wilkins
Atlantic Research Corp.
Alexandria, VA

Anita Williams
The Exonomy Co., Triple I Div.
Oklahoma City, OK

Kathy Wilson
IST Datasystems
Memphis, TN

EXHIBITS COMMITTEE

Chairman

Jerry Johns
Texas Instruments, Inc.
Houston, TX

Richard Adams
Modcomp
Ft. Lauderdale, FL

Gary Brunner
Harris Corporation
Dallas, TX

Gabe d'Annunzio
Prime Computer, Inc.
Framingham, MA

Paul Eisner
General Automation, Inc.
Anaheim, CA

Alfred Erickson
Dataproducts Corporation
Woodland Hills, CA

Ben Hayes
Datapoint Corporation
San Antonio, TX

Richard Klain
Memorex Corporation
Santa Clara, CA

Paul A. Kraska
Data 100 Corporation
Edina, MN

Ted E. Lorber
CalComp
Anaheim, CA

Robert E. Maddy
Tally Corporation
Kent, WA

Connie Magne
Intel Corporation
Sunnyvale, CA

Lynn McDaniel
Floating Point Systems
Portland, OR

Michael J. O'Rourke
Varian Graphics
Palo Alto, CA

Ken Price
Data General Corporation
Southboro, MA

Henry Sacks
Modern Data Services, Inc.
Hudson, MA

Peter Shaw
Megatek Corporation
San Diego, CA

Della Smith
Pertec Corporation
Chatsworth, CA

Matt Stein
Computer Automation
Irvine, CA

Barbara J. Wiggins
MRI Systems Corporation
Austin, TX

Len A. Zaw
Teletype Corporation
Skokie, IL

LOCAL PROMOTION COMMITTEE

Chairman

Alex A. J. Hoffman
Texas Christian University
Fort Worth, TX

Mary Ann Chapman
Delphi Data Systems
Houston, TX

Richard Priest
University of Tulsa
Tulsa, OK

Gary Hammon
Austin, TX

E. Z. Million
E. Z. Million Associates
Oklahoma City, OK

J. T. Randolph
Little Rock, AR

William Roberts
Fort Worth, TX

OPERATIONS COMMITTEE

Chairman

Robert L. Wade
Continental Trailways
Dallas, TX

Howard Albertson
Compass Computer Services
Dallas, TX

Joe C. Duncan
Dallas, TX

Herb H. Starnes
Dallas Water Utilities
Dallas, TX

REGISTRATION COMMITTEE

Chairman

Wayne Churchman
Hewlett-Packard
Richardson, TX

Johnny Lamb
City of Dallas
Dallas, TX

Libbie Neleigh
Braniff International
Dallas, TX

William M. Parker
North Central Texas Council
of Governments
Dallas, TX

Pam Stevenson
Hewlett-Packard
Dallas, TX

SPECIAL ACTIVITIES COMMITTEE

Chairman

C. E. Rodriguez
East Texas State University
Commerce, TX

Sub-Committees

Programming Contest

C. E. Rodriguez, Chairman
East Texas State University
Commerce, TX

Duane Dean
Dallas Independent School District
Dallas, TX

Science Film Theatre

Marvin Talbott, Chairman
Texas Instruments, Inc.
Dallas, TX

David Pearce
Texas Instruments, Inc.
Dallas, TX

William M. Lively
Texas A&M University
Bryan, TX

Laurence A. Madeo
University of Texas at Dallas
Dallas, TX

Gabrielle Wiorkowski
Consultant
Dallas, TX

Fred Homeyer
Angelo State University
San Angelo, TX

Technical Tours

Paul Patak
TRW Systems
Dallas, TX

Sightseeing

Ann Korfhage
Dallas, TX

Karol Frailey
Dallas, TX

PUBLICATIONS COMMITTEE

Chairman

Gabrielle Wiorkowski
Consultant
Dallas, TX

PROFESSIONAL DEVELOPMENT COMMITTEE

Chairman

Ronnie G. Ward
University of Texas at Arlington
Arlington, TX

Dan Smith
E-Systems
Greenville, TX

Ben Jarboe
Logic Inc.
Dallas, TX

PERSONAL COMPUTING COMMITTEE

Chairman

Harold A. Mauch
PerCom Data Company
Garland, TX

Ric Martin
The Micro Store
Richardson, TX

AMERICAN FEDERATION OF INFORMATION PROCESSING SOCIETIES, INC. (AFIPS)

OFFICERS AND BOARD OF DIRECTORS

President

Theodore J. Williams
Purdue University
W. Lafayette, IN

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 Papers, Inc.
Wisconsin Rapids, WI

Executive Director

Robert W. Rector
AFIPS
Montvale, NJ

ACM Directors

Herbert Grosch
Sunnyvale, CA

Carl Hammer
Univac Federal Systems
Washington, DC

Stuart Lynn
University of California
Berkeley, CA

IEEE Directors

Samuel Levine
Westport, CT

Stephen S. Yau
Northwestern University
Evanston, IL

Merlin Smith
T. J. Watson Research Center
Yorktown Heights, NY

Data Processing Management Association Directors

Barry D. Lynch
Mercantile National Bank
Dallas, TX

William J. Moser
Venture Group, Inc.
Greensburg, OH

Ralph J. Leatherman
Hughes Tool Corporation
Houston, TX

Institute of Internal Auditors Director

William E. Perry
The Institute of Internal Auditors
Orlando, FL

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

George Minich
World Bank
Washington, DC

Instrument Society of America Director

Theodore J. Williams
Purdue University
W. Lafayette, IN

Society for Computer Simulation Director

Paul W. Berthiaume
N. Y. Times Information Bank
Parsippany, NJ

Society for Industrial and Applied Mathematics Director

Donald L. Thomsen, Jr.
SIAM Institute of Mathematics
New Canaan, CT

Special Libraries Association Director

Herbert S. White
Graduate Library, School University
Bloomington, IN

American Institute of Aeronautics and Astronautics Director

Ram K. Khatri
Fairchild Space & Electronics Co.
Germantown, MD

American Society for Information Science Director

Harold Borko
UCLA School of Library Science
Los Angeles, CA

Association for Computational Linguistics Director

A. Hood Roberts
Center for Applied Linguistics
Arlington, VA

NATIONAL COMPUTER CONFERENCE BOARD MEMBERS

Chairman—AFIPS Representative

Albert S. Hoagland
IBM Corporation
San Jose, CA

Treasurer—AFIPS Representative

Walter A. Johnson
Consolidated Papers, Inc.
Wisconsin Rapids, WI

Vice Chairman—SCS Representative

Ralph Wheeler
Lockheed Missiles and Space Corp.
Sunnyvale, CA

AFIPS Representatives

Theodore J. Williams
Purdue University
W. Lafayette, IN

Harold Borko
UCLA School of Library Science
Los Angeles, CA

DPMA Representative

James F. Towsen
The Statesman Group
Harrisburg, PA

ACM Representaiive

Smith Dorsey
Rockwell International
Anaheim, CA

IEEE—CS Representative

Lowell Amdahl
Northridge, CA

NATIONAL COMPUTER CONFERENCE COMMITTEE

Chairman

Russell K. Brown
Moore Paper Company, Inc.
Houston, TX

Morton M. Astrahan
IBM Research Laboratory
San Jose, CA

Al Hawkes
Sargent & Lundy Engineers
Chicago, IL

Jeffery D. Stein
On-Line Business Systems, Inc.
San Francisco, CA

Harvey L. Garner
Stanford University
Stanford, CA

Jerry Koory
HW Systems
Van Nuys, CA

Irwin J. Sitkin
Aetna Life & Casualty
Hartford, CT

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

Abrams, Marshall D.
Altschuler, Gene
Amdahl, Lowell D.
Aronofsky, Julius S.
Avedon, Don M.

Banerji, Ranan B.
Betz, Nancy A.
Bonnette, Della T.
Brocato, Louis J.
Brown, James E.
Burns, William
Butler, Margaret K.

Cantrell, William E.
Capraro, Gerard T.
Chang, Philip Y.
Chen, Peter P. S.
Cochran, Anita
Cotterman, William W.

Dale, Alfred G.
Dubnow, Art
Duvall, Lorraine M.

England, Gordon R.

Fenwick, William A.
Firestone, Roger M.
Frailey, Dennis J.
Fuller, Donald W.
Fuller, Samuel H.

Gates, William H.

Han, Yih-Wu
Harris, Fred H.
Harvill, J. B.
Henry, J. Shirley
Horne, William J.

Ichikawa, Tadao

Jeffery, Seymour
Johnson, Marilyn
Johnson, Olin G.

Kar, Saroj
Kennedy, John R.
Korfhage, Robert R.
Knowles, Ben

Lackmann, John
Landstein, Julie E. K.
Lott, Richard

Mathur, Francis P.
Matous, James O.
McClain, William J.
McGill, Michael J.
Merwin, Richard E.
Miller, Edward F.
Mills, Roger L.
Morgan, Howard L.

Newpeck, Frederick F.

Osborne, Adam

Parker, Donn B.
Peterson, Lynn L.
Pogue, Richard
Poh, Susan S.
Poppel, Harvey L.

Rao, T. R. N.
Resta, Edward V.
Rhoten, Ronald P.

Safford, Herbert B.
Schantz, Herbert F.
Scheuermann, Peter
Smith, Eugene
Statland, Norman
Stone, Jack
Stonebraker, Michael
Storch, Nancy A.
Summers, William P.
Suttle, Jimmie R.

Thayer, Richard H.
Thiess, Helmut E.
Towsen, James F.
Turn, Rein
Turner, Nat

Vick, Charles R.

Whinston, Andrew B.
White, Robert
Wiederhold, Gio
Wilson, David
Woods, Larry D.
Wu, Y. S.

Yau, Stephen S.
Yormark, Beatrice

Zakin, Noel

PARTICIPANTS

Airapetian, A. N.
Airhart, T. E.
Albertson, L.
Allen, G. R.
Allison, D.
Amdahl, G.
Anderson, R. D.
Aronofsky, J. S.
Austing, R. H.
Avedon, D. M.

Balasubramanian, K.
Barry, T.
Barton, G. S.
Bell, D. H.
Benton, J. R.
Berkey, J.
Berra, P. B.
Berson, J.
Bigelow, R. P.
Blazie, D.
Bolnick, F. I.
Borko, H.
Bowie, J.
Braun, L.
Burns, W. J.

Carlstrom, D.
Carter, W. C.
Cary, T.
Casper, G. G.
Castelberg, M. J.
Chamberlin, H.
Clarke, L.
Cole, C.
Colvin, N.
Cornell, J. A.
Cotterman, W. W.
Couger, J. D.
Cox, J. A.
Cragon, H.

Dejka, W. J.
Dodd, G.
Dratch, J.
Duncan, K.
Dunstan, E.

Earle, J.
Edwards, J.
Eger, J. M.
Elfenbein, L.
Elspas, B.
Elwood, W. F.

Faber, E.
Felsenstein, L.
Feng, T. Y.
Flores, A.
Fly, W. W.
Fossum, B. M.
Foster, C. C.
Frame, R. J.
Frenzel, L.
Fry, J. P.
Fu, K. S.
Fuchs, H.
Fuller, D. W.
Furr, C.

Galler, B. A.
Garrison, O.
Gianola, J.
Goddard, A.
Goguen, N.
Goldberg, J.
Grosch, H. R. J.
Guzeman, O.
Gwinner, R.

Harmon, G.
Harris, D. K.
Harris, F. H.
Harrison, H.
Haseman, W. D.
Heiser, R.
Held, G.
Helmers, C.
Henry, S.
Hoagland, A.
Hollow, D.
Hopkins, A. L.
Horne, W. J.
Howden, W. E.
Howell, T. A.
Hsiao, D. K.

Jamison, S. L.
Jeffrey, S.
Jenson, D.
Joseph, E. C.

Kar, S. K.
Kent, B.
Kent, W.
Kildall, G.
King, W. F.
Kohlmeir

Lackmann, J.
Lazarus, R.
Leavitt, D.
Lee, R.
Lehmann, J.
Little, J. C.
Lum, V. Y.

MacLean, J. D.
Manola, F.
Maxmen, J.
McCloud, R.
McCracken, D.
McDonald, J. F.
McKemie, G. W.
McLean, J. D.
McLeod, J.
McNair, E. A.
McNulty, J.
Meyer, D.
Miller, J.
Mills, H.
Mills, R. L.
Moore, G.
Morelock, T. J.
Musser, D. R.

Neidell, N. S.
Nelson, T. H.
Norman, S. L.
Nyborg, P. S.
Nycum, S. H.

Osborne, A.
Osborne, J.
Oyer, P. D.

Painter, J. A.
Palmer, C.
Panko, J. W.
Paul, J. T.
Peaceman, D. W.
Peck, J. C.
Peebles, R.
Perkins, C.
Poland, S.
Poppa, R.
Posdamer, J.
Purdy, G.

Rakel, R. E.
Rector, C.
Reddy, D. R.
Ruder, B.
Rule, J. B.

Sacerdoti, E.
Salisbury, A. B.
Sandman, M. D.
Schantz, H. F.
Schaster, S. A.
Shear, R. D.
Short, R.
Silberschatz, A.
Sklansky, J.
Smoot, Oliver R.
Solomon, L.
Spadaro, F. G.
Spaniel, R. D.
Steel, T. B.
Stevenson, T. Q.

Thiess, H. E.
Thomas, R.
Trimble, J.
Tsichritzis, D.

Uiterwyk, R.

Venetta, B.
Vick, C.
Voorhees, E.

Walker, M. R.
Wallace, R.
Waller, R.
Webster, S.
Weissman, N.
Whipple, D.
Williams, B.
Wilson, S.
Worlton, J.

Yao, S. B.
Yeh, R. T.
Yormark, B.
Young, B.

Zilles, S.

NCC 77 REFEREES

Abbey, Duane C.
Abbey, Mary W.
Abrahamson, Howard
Abrams, Marshall D.
Ackerman, L. V.
Agrawal, Dharma P.
Aicher, J. R.
Aiken, Robert
Aines, Andrew A.
Albers, Glen
Alexiou, John K.
Allen, John R.
Allen, Rodney H.
Amarel, Saul
Amdahl, Carlton G.
Andersen, Niels C.
Anderson, Henry D.
Anderson, Peter G.
Anderson, Richard J.
Anderson, Robert H.
Anderson, Thomas C.
Andree, Richard V.
Antal, J. R.
Archibald Jr., J. A.
Armer, Paul
Arterbery, Vivian J.
Ash, Alvin
Astrahan, Morton M.
Atwood Jr., Delbert W.
Augustin, Donald C.
Aupperle, Eric M.
Austin, Donald M.
Austing, Richard H.
Ayer, Nancy L.
Ayers Jr., Lawrence F.

Baer, J. L.
Baird, George N.
Baker, Bob E.
Baker, F. T.
Baker, James A.
Baker, Robert L.
Baker, Ronald A.
Balkovich, Edward E.
Ball, N. A.
Baltzer, P. K.
Banerji, Ranan B.
Barlow, Allen E.
Barnes, Bruce H.
Barnes, Robert F.
Barnett, Octo
Barr, William J.
Barrett, William A.
Bassler, Richard A.
Bate, Roger R.
Bates, Madeleine
Bauman, Burton L.
Baxter, Fred
Beall, W. H.
Bearden, G. D.
Beck, Leland
Beguelin, J. L.
Belford, Geneva
Bell, Thomas E.
Belzer, Jack
Bemer, R. W.
Bennett, John L.
Berg, Frank A.
Berg, John L.
Berger, Ralph
Berk, Toby
Berning, Paul T.
Bernstein, George B.
Bernstein, M. I.
Bernstein, Ralph
Berra, Bruce
Betz, Nancy
Bewley, William L.
Bezalel, Gavish
Bigelow, Robert
Billingsley, Fred C.
Bilyk, Walter
Binder, Richard D.
Binford, Thomas O.
Bise, Robert G.
Bitterli, Charles V.
Black, Donald V.
Blanc, Robert P.
Blomgren, George H.
Bloomfield, James A.
Blue Sr., Richard B.
Blum, Joseph
Bodoin, Morris J.
Bollenbacher, Roger L.
Bono, Peter R.
Booth, Grayce M.
Booth, Taylor
Bork, Alfred
Bouknight, Jack
Brackett, John W.
Braithwaite, William R.
Braun, Randal R.
Brekhus, Harry E.
Bressler, Robert
Brociner, Betty B.
Brown, John R.
Brown, Russell K.
Browne, Peter S.
Bryan, G. E.
Burlakoff, Mike
Burns, Joseph L.
Burns, Lawrence E.
Burns, William J.
Burton Jr., William D.
Buscher, David J.
Butler, George
Butler, Robert S.
Buxbaum, Richard J.

Cady, George M.
Campaigne, Howard
Campbell, John B.
Campbell, Rosalie A.
Campise, James A.
Cannon Jr., George R.
Cannon Jr., Robert L.
Caplan, David
Capodanno, Lori
Capraro, Gerard T.
Carey, Bernard J.
Carlson, Carl R.
Carlson, Eric D.
Carlson, Gary
Carlson, James C.
Carlson, Richard R.
Carter, George
Case, II, Leon R.
Case, Richard P.
Cashman, Thomas J.
Cashton, Sidney
Castruccio, Peter A.
Champine, G. A.
Chan, Maynard M. W.
Chandrasekaran, B.
Chang, Donald Y.
Chang, Hsu
Chansky, Leonard M.
Charp, Sylvia
Chauhan, Rohi
Chen, Peter P.
Chen, Robert C.
Chen, Thomas T.
Chen, Tien C.
Cheung, Roger
Cheydleur, Benjamin F.
Chiaraviglio, Lucio
Chinitz, M. P.
Cho, Seon H.
Chu, W. W.
Chu, Yaohan
Clema, Joe K.
Clough, Marlen S.
Cohen, Dan
Cohen, Jack
Cole, G. D.

Coleman, Michael
Collins, John J.
Connelly, Donald
Conner, William M.
Cook, H. G.
Cook, Meyer
Cooprider, Lee
Corduan, Alfred E.
Corley, Melvin R.
Couperus, J.
Cowan, Robert J.
Cresto, John
Culpepper, L. M.

Dahm, David M.
Dalphin, John F.
Daniels Jr., Walter E.
Danner, Lee
Davida, George
Davidson, Donald A.
Davis, Al
Davis, Claud M.
Davis, John C.
Day, Paul
Day, William H. E.
De Greene, Kenyon B.
Dean, Edwin B.
Deb, Rajat K.
Defiore, Casper R.
DeLutis, Thomas G.
Dettmann, C. E.
Devine, Edward P.
Dewdney, A. K.
Dixon, Louis F.
Dixon, R. D.
Dobkin, David
Dorn, Philip H.
Drattell, Alan
Ducasse, Edgar
Dumey, Arnold I.
Duncan, Karen
Duran, Joe W.
Dutka, Jacques
Duvall, Lorraine
Dylewski, T. J.

Eccles, William J.
Elfant, Robert F.
Elkins, Bryce L.
Elliott, Glenn
Elman, Stanley A.
Emerson, E. J.
Engel, Diana
Engel, Gerald L.
Enslow Jr., Philip H.
Erickson, Raymond
Ernst, George W.
Esch, John
Estock, Richard G.
Estrin, Thelma
Euler, Ruth S.
Evans, W. B.
Evert, Carl F.

Farmer, Nick A.
Fazekas, Diane
Feingold, Robert S.
Feng, T.
Ferguson, C. W.
Ferrari, D.
Feurzeig, Wallace
Feyock, Stefan
Fike, John L.
Firestone, Roger M.
Firschein, Oscar
Fleck Jr., Robert A.
Fly, William W.
Fogel, Marc H.
Fong, Elizabeth
Foster, Caxton C.
Foster, David F.
Foulk, Clinton R.
Fowler, Bruce R.
Fox, Phillip W.
Frank, Howard
Frank, Werner L.
Franke, Richard
Freeman, Martin
Freiman, C. V.
French, Larry J.
Friedman, Daniel P.
Friedman, Lee A.
Friedman, Richard B.
Fritz, W. B.
Froom, Jack
Fuchs, Henry
Fujiwara, Harry A.
Fuller, Samuel H.
Futrelle, R. P.

Gabrieli, E. R.
Galler, Bernard A.
Gallo, Arpad
Gammon, William H.
Gannon, John
Gantner, George
Gardner, Williard H.
Garrett, R. E.
Gates, G. W.
Gates, Roy
Gaudot, Frank J.
Gerle, Mario
Gibb, Kenneth R.
Giesa, Charles E.
Gilchrist, Bruce
Gilliland, B. E.
Glanc, Alois
Glaseman, S.
Glasser, Robert G.
Glick, Norman
Glorioso, Robert M.
Goetowski, Charles R.
Goetz, Martin
Goldberg, Adele
Goldberg, Jack
Goldhirsh, Isadore L.
Golding, E. I.
Goldman, Neil M.
Goldstein, Charles M.
Gonzalez Jr., Mario J.
Gorgone, John T.
Gorman, Donald F.
Gorsline, G. W.
Goshen, Robert J.
Goti, J. C.
Gould, John D.
Goulk, Clinton R.
Grace Jr., Alonzo G.
Graham, G. S.
Grampp, F. T.
Gray, Jim
Greaves, John O.
Green III, Duff
Green, Teresa O.
Greenawalt, E. M.
Greene, Lynn
Griffin, John
Grobstein, David L.
Groner, Gabriel F.
Grosch, Audrey N.
Gruhn, Ann M.
Guetzkow, Harold
Guiteras, Joseph J.
Gumb, Raymond D.

Habib, Stan
Hall, Carlton S.
Hall, Wayne A.
Hallblade, Shirley A.
Hamblen, John W.
Hamilton, Dennis E.
Hammer, Fred E.
Hammer, Michael
Hammer, Preston C.
Hammond, W. E.
Hampel, D.
Hanna Jr., William E.
Hansen, Gilbert J.
Hansen, John C.
Hardgrave, W. T.
Harmon, John B.
Harper, Jackson D.
Harris, Floyd O.
Harris, Richard D.
Harris, Roger L.
Hartford, Donald L.
Hartley III, Dean S.

Hartwick, R. D.
Hattery, Lowell H.
Hays, Bill
Hedrick, G. E.
Henne, Randy L.
Henschen, L. J.
Hernon, James A.
Hertlein, Grace C.
Hess, George J.
Higgins, Alan N.
Ho, Siu-bun F.
Ho, Thomas I.
Hobbs, Jerry R.
Hodge, Bartow
Hodge, Thea D.
Hodges, Ann G.
Hoffman, Lance J.
Hoffman, Robert H.
Holden, Alistair D.
Holme, Dorothea R.
Holmes, Harvard
Hook, Harvey O.
Hoover, L. R.
Hopewell, Lynn
Hopper, Grace M.
Hopwood, Gregory L.
Hord, R. M.
Horne, William J.
Howell, Jo Ann
Hoyt, Patrick M.
Huang, H. K.
Hubans, Frank
Huckell, Gary R.
Humphrey, Timothy L.
Hunt, Hurshell H.
Huntwork, Paul K.
Hurst, Len
Hutchison, John S.
Hwang, F. K.

Ingerman, P. Z.

Jacobs, Stanley E.
Jacobus, Gilbert C.
James, Thomas A.
Janac, Karel
Jefferson, David K.
Jensen, Alton P.
Jensen, Raymond A.
Jessep, Donald C.
Johnson, A. I.
Johnson, James H.
Johnson, O. G.
Johnson, Walter L.
Jones, Anita K.
Jones, John L.
Jones, Neil D.
Joyce, James
Juhlin, Kenton D.
Julke, Robert T.

Kaber, A. B.
Kagan, Claude A.
Kahng, S. W.
Kain, Richard Y.
Kampen, Garry
Kandel, A.
Karplus, W. J.
Kasarda, Andrew J.
Katic, James R.
Katzper, Meyer
Kavach, Ladis D.
Keller, Roy F.
Kieburtz, R. B.
King, Alan S.
King, James C.
Kirshenbaum, Frank
Kish, William
Kiviat, Philip J.
Klassen, Daniel L.
Klinger, A.
Koch, Harvey S.
Korfhage, Robert R.
Kornfield, N. R.
Koss, Adele M.
Koss, Neal
Kovac, John G.
Kovach, Ladis D.
Kozik, Eugene
Kraley, Michael F.
Krishnarao, T.
Kroeger, Joseph H.
Krulee, Gilbert K.
Kuch, T. D.
Kurihara, Thomas M.
Kurtzberg, Jerome M.

LaFrance, Jacques
Lai, Kwok-Woon
Lamothe, Raymond J.
Lane, Malcolm G.
~~Larson, Arvid G.~~
Lasser, Daniel J.
Latker, Alex C.
Laurance, Neal
Lawrie, Duncan W.
Lazar, Leonard M.
Le Beux, Pierre J.
Leasure, Bruce R.
Leavitt, M. R.
Ledbetter, Hardy
Ledin, Victor
Lee, J. A. N.
Lee, Marshall
Lennon, James J.
Leung, Francis W.
Levin, Roy
Ligler, George
Lin, Wen-te K.
Lincoln, A. J.

Linden, Theodore A.
Liskov, Barbara
Liu, C. L.
Liu, Jane W.
Logan, J. J.
Logan, John
Logue, Joseph C.
Lomet, David B.
Long, Harvey S.
Long, John M.
Lott, Richard
Lovegrove, Donald H.
Lowe, Thomas C.
Lozier, Daniel W.
Lucido, A. P.
Luck, Dennis R.
Ludwig, Herbert R.
Luk, Clement
Lukas, George
Luke, Richard F.
Lutz, Michael J.
Lycklama, H.
Lykos, Peter
Lyle, Robert F.
Lynch, John T.
Lyons, W. W.

Machover, Carl
MacLeod, Franklin B.
Madrigal, Orlando S.
Madron, Beverly B.
Maguire, John N.
Maniotes, John
Mann, Richard L.
Manola, Frank
Maple, Claire G.
Marcovitz, Alan B.
Marks, Sema
Marmor-Squires, Ann B.
Maskewitz, Betty F.
~~Mason, Philip H~~
Matheny, Charles S.
Mathews, Max V.
Mathews, Walter M.
Mathison, Stuart
Matyas, Stephen M.
McClain, William J.
McCluskey, E. J.
McCready, R. R.
McCuskey, William A.
McDonald, Clement J.
McDonald, Nancy H.
McFadden, Ted
McGill, Michael J.
McGregor, P. V.
McInnis, Bayliss
McJones, Paul
McKenna, James
McKnight, R. S.

McLeod, Dennis
McMahon, J. T.
Meads, Jon A.
Mehl, James W.
Meltzer, H. S.
Merwin, Richard E.
Metcalfe, Bob
Metzner, John R.
Michael, Mark T.
Mihram, G. A.
Miles, E. P.
Million, E. Z.
Mills, David L.
Mills, Harlan D.
Mills, Lesley M.
Mills, Roger L.
Mills, Wayne L.
Mink, Thomas A.
Minker, Jack
Mintz, Daniel
Modesitt, Kenneth L.
Mamrak, Sandra
Moore, Michael
Moraff, Howard
Morgan, M. G.
Morris, Michael F.
Morton, A. K.
Moshos, George J.
Muchnik, Steven S.
Mullany, James E.
Mulroney, William C.
Murphy, Gretchen
Muzio, J. C.

Nagel, Roger N.
Nance, Richard E.
Nasem, Charles
Nash-Webber, Bonnie
Nee, David S.
Nelson, Eldred
Nemeth, Alan C.
Neurath, Peter W.
Nevins, James L.
Newenschwander, Charles R.
Newhouse, Albert
Newton, Carol W.
Niedermair, F. R.
Nievergelt, J.
Noetzer, Andrew
Nolan, Lawrence E.
Noonan, R.
Norman, Theodore A.
Norton, Richard M.
Nutt, Gary J.
Nuxall, John

O'Kane, Kevin C.
O'Neill, Dennis
Odesky, Robert I.
Olah, George T.
Oliver, S. R.
Olmer, Jane
Osher, William J.
Ossanna, Joseph
Osterer, Lorraine
Osterweil, Leon
Owens, John D.

Painter, Frank R.
Palley, N.
Palmer, Richard
Parker, D. S.
Parker, Donn B.
Parrish, Harry T.
Patrick, Edward A.
Patt, Yale
Patton, S. K.
Payne, Mary H.
Pehlert, William K.
Penderghast, Tom
Pendleton, Dave
Perry, James M.
Perry, Raymond S.
Person, Warren
Petersen, Tom
Peterson, Emery G.
Peterson, James L.
Peterson, Lynn L.
Pfleeger, Charles
Pickholtz, Raymond L.
Pinson, Elliot N.
Pizer, Stephen M.
Plauger, P. J.
Plourde, Paul J.
Pogue, Richard E.
Poh, Susan S.
Pokorney, Joseph L.
Pooch, Udo W.
Popino, J. P.
Potter, Marshall R.
Potts, Jackie
Powers, Richard
Prescott, Lee R.
Press, Barry
Presser, Leon
Purdy, J. G.
Purdy, Melanie S.
Pyke Jr., Thomas N.

Quann, John J.

Raben, Joseph
Rabinowitz, Irving N.
Radre, Albert N.
Raj-Karne, D. G.
Ramamoorthy, C. V.
Redell, David D.
Reid, Robert A.
Reiss, R. A.
Reitman, Julian
Reynolds, Brian M.
Rheinboldt, W.
Rhyne, James R.
Rice, John
Rich, Robert P.
Riddle, William E.
Rieger III, Charles J.
Riley III, Winston
Rinewalt, J. R.
Risse, Joseph A.
Ritea, H. B.
Rittersbach, George H.
Robbins, Galen P.
Roberts, Justine
Robinson, John
Rogers, David F.
Rogh, R. W.
Rohr, John A.
Ronayne, Maurice F.
Rose, Lawrence L.
Roseman, Jack
Rosen, Robert
Rosenbaum, Susan L.
Rosin, Robert F.
Roth, Waldo R.
Rothstein, Jerome
Rotolo, Louis S.
Rubey, Raymond J.
Rubin, Arthur I.
Ruh, Lawrence A.
Russo, Paul
Ruth, Stephen R.

Safford, Herbert B.
Salasin, John
Salz, Fred R.
Salzman, Roy M.
Sanders, Alton F.
Sands, J. E.
Sager, Naomi
Saupe, Paul H.
Savage, John E.
Schaffner, Mario
Scharff, Leon
Scher, Julian M.
Schlegel, C. T.
Schmidt, William P.
Schneck, Paul B.
Schultz, Gaymond W.
Schutzer, Daniel
Scott, James L.
Scott, Philip
Scott, Robert H.
Scrutchin, Thomas W.
Seals, Eugene
Sedelow, Sally Y.
Sekino, Warren T.

Sevcik, K. C.
Shahin, Gordon T.
Shannon, Roger H.
Shapiro, Stuart C.
Shaw, Alan C.
Shelly, Gary B.
Sheppard, David A.
Sherman, Stephen W.
Shetler, A. C.
Shoaf, Gerald H.
Shreckengost, Raymond C.
Shuey, Richard L.
Shum, Annie
Sibley, Edgar H.
Sibley, W. L.
Sickel, Sharon
Siegel, Arnold
Silberschatz, Abraham
Silberstein, Stephen M.
Silvern, Leonard C.
Simmons, Dick B.
Simmons Jr., Edward J.
Skeel, Robert D.
Slaughter, Lawrence H.
Smith, C. O.
Smith, Cecil L.
Smith, David M.
Smith, Eugene
Smith Jr., Gerould H.
Smoot, Oliver R.
Snead, Bill
So, Hon H.
Soh, Jin W.
Sokol, George M.
Sondheimer, Norman K.
Sorkowitz, Al
Spaniol, Roland
Spier, Michael J.
Spinrad, Robert
Spiro, Bruce E.
Springe, F. W.
Squires, Stephen L.
Stadel, Patricia A.
Stahl, Fred
Stallings, William
Stead, William W.
Steel Jr., Thomas B.
Stefferud, Einar
Steig, Donald B.
Stelmack, Frank
Stemple, David W.
Stevens, D. F.
Stewart, W. B.
Stokes, Gordon E.
Stone, Robert L.
Story, James R.
Stranart, J. C.
Strassmann, P. A.
Stroud, William G.
Stuebing, Henry G.
Stumpf, Werner E.
Swan, Richard
Swanson, A. K.
Swearingen, John K.
Swenson, J. R.
Swigger, Boyd K.
Szygenda, S. A.

Tam, Wing C.
Taulbee, Orrin E.
Tausner, Miriam R.
Taylor, Judith E.
Taylor, Robert W.
Teichroew, Daniel
Thayer, Richard H.
Theilheimer, Feodor
Thomas, James C.
Thomas, John C.
Thompson, Ken
Thompson, Martin D.
Thurber, Kenneth J.
Tick, Leo J.
Tischhauser, J. L.
Tobagi, Fouad A.
Tonik, Albert B.
Townsend, Frederick W.
Trivedi, M. C.
Tucker, Edwin K.
Tupp, Paul L.
Turn, Rein
Turner, Jon A.
Turoff, Murray

Uhr, Leonard
Ulery, Dana L.
Umpleby, Stuart A.
Uzgalis, Robert C.

Vidal, Jacques J.

Wachal, Robert S.
Wagner, Charles R.
Wagner, Francis V.
Walford, Robert B.
Walker, Donald E.
Walker, Rob
Walsh, Patrick J.
Walters, Richard F.
Walters, T. L.
Wand, Mitchell
Ward, Wayne D.
Warner, Walter P.
Wasserman, Anthony I.
Wear, Larry L.
Weber, Larry L.
Weber, Robert A.
Wedberg, George H.
Weihrer, Anna L.
Weinberger, Gil
Weiss, Edward C.
Weiss, Eric A.
Weiss, Stephen F.
Wells, J. M.
Wells, Mark B.
Wen, Kuo Y.
Werner, William E.
Wesselkamper, T. C.
Wexelblat, Richard L.
Weyl, Stephen
Wheeler, T. F.
Whinston, Andrew
White, John R.
Whitman, Kirk
Wieselman, Irving L.
Willett, R. M.
Williams, Ben T.
Williams, Leland H.
Williams, Richard P.
Williams, Thomas A.
Williams, Thomas G.
Williamson, Gloria A.
Willman, Herb
Winters, Jack H.
Wofsey, Marvin M.
Wojcicki, Maria E.
Wolf, Eric
Woodbury, Max A.
Woodgate, H. S.
Woodson, Charles E.
Wooton, Leland M.
Worlton, Jack
Wortz, Charles
Wright Jr., Charles T.
Wright, Kendall R.
Wright, S. E.
Wulf, William A.
Wyner, Donald

Yan, George
Yao, S. B.
Yarbrough, L. D.
Yasnoff, William A.
Yeh, Raymond
Yonda, A. W.
York, Kenneth L.
Young, J. W.
Yovits, Marshall C.
Yue, Jean

Zak, Francis X.
Zelhowitz, Marvin V.
Zellweger, Andres
Zieha, Eugene L.
Zimmerman, Joan
Zimmerman, Martin B.
Zinn, Karl L.
Zislis, Paul M.
Zweben, Stuart H.

AUTHOR INDEX