# USA DECUS

## Anaheim, California

## 1987 Fall Proceedings

# Foreward

This Proceedings is published by DECUS (Digital Equipment Computer Users Society), a world-wide society of users of computers, computer peripheral equipment and software manufactured by Digital Equipment Corporation. The U.S. Chapter of DECUS has approximately 60,000 active members.

DECUS maintains a library of programs for exchange among members and organizes meetings on local, national and international levels to fulfill its primary functions of advancing the art of computations and providing a means of interchange of information and ideas among members. Two major technical symposia are held annually in the United States.

For information on the availability of back issues of U.S. Chapter Proceedings as well as forthcoming DECUS symposia, contact the following:

DECUS U.S. Chapter
219 Boston Post Road, BP02
Marlboro, MA 01752-1850

All issues of past Proceedings are available on microfilm from:

University of Microfilms International
300 North Zeeb Road
Ann Arbor, MI 48106

# Preface

This volume of the Proceedings contains papers which were presented at a symposium sponsored by the Digital Equipment Computer Users Society during the Fall of 1987.

The Fall 1987 Symposium was held at the Anaheim Convention Center, in Anaheim, California, from December 7 through 11, 1987. Over 6000 DECUS members joined together in Anaheim for a week of intensive learning, training, and sharing.

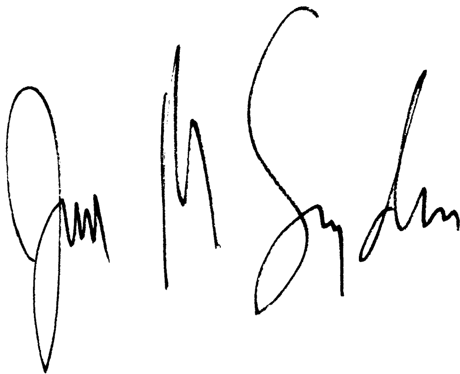
One of the highlights of this symposium was the introduction of several featured speakers. While the national symposium traditionally has been sponsored and presented entirely by users and Digital, the DECUS Symposium Committee is to be commended for going out of their way to attract a few special speakers who have presented especially exciting sessions. While these keynote and featured speakers may not have brought in-depth technical information to the symposium, their experience and breadth over a wide variety of fields gave DECUS members new insights into old problems.

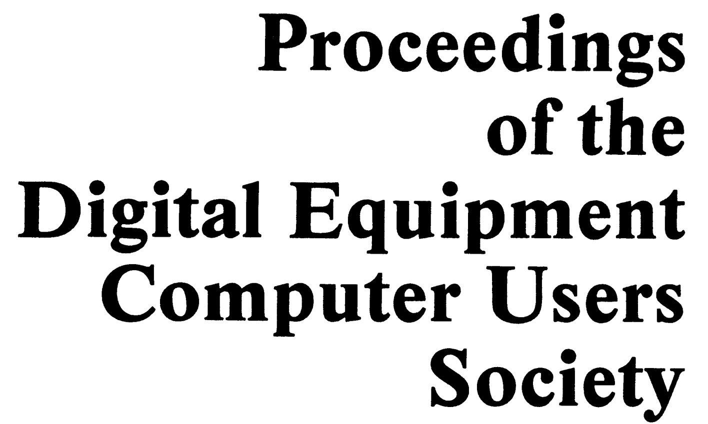
The VAX computer system project was begun ten years ago this year, and the VAX SIG held special festivities to celebrate the event. Digital pitched in by providing an almost-authentic original VAX-11/780 in the exhibition hall running VAX/VMS version 1, with an RP06 disk drive and VT52 terminals. The VAX SIG held a well-attended banquet dinner one evening, complete with design lore, anecdotes, and the usual war stories and prophesies-come-true.

Cathy Ditamore, the DECUS volunteer who has led the Symposium Publications Committee for the past year, has agreed to move her management skills and enthusiasm up the DECUS ladder, and will be serving as Vice Chair of the Communications Committee. My thanks are due to both her and Beverly Welborne for their support and hard work on behalf of the Proceedings. DECUS Board of Directors members Bill Brindley and Bob Curley also provided me with a great deal of personal guidance this past year, and their help is well appreciated.

My thanks on behalf of the attendees of the Fall National Symposium go out to Chris Wool and Emily Kitchen, the DECUS volunteers who led the Symposium Committee. They worked together with DECUS staff members led by Nancy Wilga to put together a highly successful and seamless Fall meeting. The leadership of the entire Symposium Committee is sincerely appreciated. I'd also like to thank the DECUS staff members who make my life a lot easier --- Judy Mulvey, Cheryl Smith, Beverly Dandeneau, and Gloria Caputo.

Joel M Snyder
Proceedings Editor





# Proceedings of the Digital Equipment Computer Users Society

## USA FAll 1987

## Papers Presented at Fall, 1987 Symposium Anaheim, California December 7 - 11, 1987

# Table of Content


## Artificial Intelligence SIG

Artificial Intelligence Applied to the Help Function
*Robert Stanley* .... *1*

The Use of PHIGS in an Artificial Intelligence Environment For Mechanical Engineering
*Mike Thompson, Jim Roth* .... *15*

An Overview of the Common LISP Object System
*Richard P. Gabriel, Linda G. DeMichiel* .... *23*

## Business Applications SIG

Postage & Mailing Cost Saving Computer Programs
*Richard L. Fleischer* .... *37*

## Data Acquisition, Analysis, Research, and Control SIG

Developing a CIM Architecture
*Nigel P. Weymont, Jeffrey S. Honeyager* .... *45*

Interactive Control Engineering Computer Analysis Program
*Robert L. Ewing, Sam C. Huges, Kris L. Larsen, Gary B. Lamont* .... *67*

RT-11/VMS Networking for Real-Time Applications
*Jonathan D. Melvin, Ph.D.* .... *73*

## Data Management SIG

An Automatic Source-Code Generator Generating Subroutines for Accessing an Rdb Database
*David M. Hansen* .... *79*

## DATATRIEVE SIG

Using DATATRIEVE with VAX/DBMS
*Alan H. Beer* .... *89*

Managing ALL-IN-1 with DATATRIEVE
*Bart Z. Lederman* .... *93*

Solving Equations in DATATRIEVE
*Bart Z. Lederman* .... *105*

## EDUSIG

Voice: A Computer Controlled Telephone Information System
*Lisa M. Rotunni, Edward C. Hohmann, Son V. Phan, James A. Rounds* .... *115*

## Graphics Applications SIG

Report Generation Using a Visual Programming Interface
*Tim Dudley* .... *123*

Introduction to SMG, The VMS Screen Management Utility
*Robert L. Hays* .... *131*

## Languages and Tools SIG

Customizing VAXLSE for your Language
*Jana Van Wyk* .... *139*

Automating a Software Development Environment
*Linda L. Craddock* .... *147*

## Large Systems SIG

The Internet Domain Name System
*S. Robert Austein* .... *157*

## Networks SIG

An Architectural Perspective of a Common Distributed Heterogeneous Message Bus
*Howard Kilman, Glen Macko* .... *171*

A Local Area Network for a Multivendor Environment
*Roger G. Ruckert* .... *185*

## Office Automation SIG

Personnel Computers and ALL-IN-1: Document Transfer and Translation
*C.F. Stanland* .... *193*

Basic Networking for Office Automation
*Robert Gary Mauler, Valerie Cabral Mauler* ..... 199

Office Automation Security: Closing the Doors to Your Computer System
*Robert A. Clyde* ..... 209

Writers as User Interface Designers
*Peter Donahue* ..... 215

## Personal Computer SIG

Trojan Horse Software
*Kenneth A. Stricker* ..... 223

PC Workstation to VAX Connections for Maximizing Resource Flexibility
*Robert Gary Mauler, Valerie Cabral Mauler* ..... 227

## RSX-11 SIG

Modifying FMS-11 to Provide Read-With-Timeout and Video Attribute Control
*Joseph E. Kulaga* ..... 237

## RT-11 SIG

XL/XC/CL Programming for RT-11/TSX-PLUS
*Ned W. Rhodes* ..... 243

Real World Disk Comparisons
*Robert C. Peckham, Milton D. Campbell* ..... 257

## Site, Management and Training SIG

Diary of a Novice System Manager
*Mark Roark Chartier* ..... 283

## VAX Systems SIG

Viruses, Worms, and Trojan Horses-Part II
*Robert A. Clyde* ..... 297

REMPRINT: Remote Printing for VAX/VMS
*Marty Adkins* ..... 301

SOFTQUOTA: A Diskspace Management Utility
*Shari Dishop* ..... 307

Fast Response on Overloaded Systems (or the alchemy of the VMS scheduler)
*Silvano de Gennaro* ..... 313

Evaluation of Third Party VAX/VMS Disk Compression Products
*Marian K. Iannuzzi* ..... 317

VMS Disk Performance
*Wef Fleischman* ..... 331

Coping with Full Disks
*Melcolm Dunn* ..... 341

# ARITICIAL INTELLIGENCE SIG

**Presentation Title:** Artificial Intelligence Applied to the HELP Function

**Code:** AI020

**Author:** Robert Stanley

**Title:** Senior Researcher

**Address:** Cognos Incorporated
P.O. Box 9707
3755 Riverside Drive
Ottawa, Ontario
CANADA K1G 3Z4

**Telephone:** (613) 738-1440

**Language:** English

**Audience:** Technical
Managerial
General Interest √

**Skill Level:** New User
Mid-range √
Sophisticated √

## Abstract:

Traditionally, on-line help for computer software has been based on one of two broad approaches, the on-line manual, and context-sensitive help text. Both mechanisms have spawned numerous sub-genres, many of which demonstrate considerable ingenuity and sophistication. The key factor in the development of these help facilities has been the increasing availability and decreasing cost of high capacity storage, both memory and disk, which has made it feasible to have large volumes of text on-line to computer users.

Most recently, further improvements in the price-performance ratios of computer hardware, coupled with steady progress in artificial intelligence research, have made it possible to consider harnessing so called knowledge-based technologies to the provision of help facilities. Smart help, as it is usually known, can simply be a more sophisticated version of one of the two existing approaches, but this technology also introduces the possibility of new mechanisms.

This paper describes a research project designed to develop a prototype advisor for Cognos' QUIZ report writer. An advisor is a knowledge-based program capable of conducting a dialogue with a user and answering questions within its domain of knowledge, in this case knowledge of a 4th generation computer application programming language. As well as describing the results of the research to date, this paper briefly introduces other potential applications of the technology, discusses their viability, and proposes likely time-scales for their practical availability.

COGNOS
Research

# Artificial Intelligence Applied to the HELP Function

**Robert Stanley**
Cognos Incorporated
Ottawa, Ontario

## Abstract

Traditionally, on-line help for computer software has been based on one of two broad approaches, the on-line manual, and context-sensitive help text. Both mechanisms have spawned numerous sub-genres, many of which demonstrate considerable ingenuity and sophistication. The key factor in the development of these help facilities has been the increasing availability and decreasing cost of high capacity storage, both memory and disk, which has made it feasible to have large volumes of text on-line to computer users.

Most recently, further improvements in the price-performance ratios of computer hardware, coupled with steady progress in artificial intelligence research, have made it possible to consider harnessing so called knowledge-based technologies to the provision of help facilities. Smart help, as it is usually known, can simply be a more sophisticated version of one of the two existing approaches, but this technology also introduces the possibility of new mechanisms.

This paper describes a research project designed to develop a prototype advisor for Cognos' QUIZ report writer. An advisor is a knowledge-based program capable of conducting a dialogue with a user and answering questions within its domain of knowledge, in this case knowledge of a 4th generation computer application programming language. As well as describing the results of the research to date, this paper briefly introduces other potential applications of the technology, discusses their viability, and proposes likely time-scales for their practical availability.

## Historical Background

I vividly remember the first time I saw a video display terminal which had a dedicated *help* key; finally, it appeared, a manufacturer was addressing the real needs of computer users. My initial excitement rapidly turned to frustration as I realized that the key didn't actually do anything other than generate a code that could be uniquely identified within a program, *should the programmer choose to check for it*. The experience was a salutary one, in as much as it made me examine some of the fundamental underpinnings of computer systems, which in turn sparked an interest in human-computer interaction that has never waned. But on that day in the early 'seventies, it was the frustration and anger at the thoughtlessness of system designers that predominated. How, I wondered, could an industry continually introduce potentially wonderful mechanisms that for all practical purposes have no useful application?

The answer, as always, turns out to be simply the perpetuation of a tradition that has its origins in the exigencies forced by engineering limitations in pioneering days. The commercial data-processing industry has been, and generally still is, hidebound and conservative in the extreme; only a trickle from the flood of research ever finds general acceptance, and then usually only under the ægis of a determined champion or visionary. The classic example of this in recent years has been the introduction of the Apple Macintosh, which has finally won acceptance for ideas that have been discussed for more than a decade. Of course, the ideas also required the introduction of affordable delivery technologies for their realization, and this is the heart of the problem.

In the earliest days of computing, all users were programmers, and errors manifested themselves as system failures of varying degrees of subtlety. Tracking down the source of a problem was a time-consuming and demanding exercise, and the only help available was discussion with colleagues. Non-programming users first appeared on the scene with the introduction of batch-processed applications, which were accompanied (after the need for them rapidly became apparent) by detailed user instructions for running the application, and a book full of possible error situations, each identified by a cryptic code. Hardware limitations, particularly in memory capacity, necessitated the use of codes rather than text messages. The form of the documentation, which was written by the program authors, reflected the tastes and habits of the programmers, which in turn were shaped by the development environment in which the programmers worked, which ultimately were rooted in the engineering design decisions taken by the hardware

manufacturers. The only variation on this theme was provided by the introduction of separate quality assurance or 'testing' sections, who at least verified the correctness of the documentation as well as the applications.

It was really the appearance of interactive computer terminals in the late 'sixties which first started a trend towards addressing more fundamental user needs than simply the raw mechanism for running an application. The problems faced by an unsophisticated computer user (usually described as one with no real understanding of the operating system and programming language characteristics) in attempting to successfully execute a job necessitated impractical levels of training or a radically different approach. In the short term, operators developed startling skills, and major computer centres placed all programmers on operational trouble-shooting detail, frequently on a three-shift basis. The answer was the development of interactive or on-line help facilities, but these were slow in arriving because they required vast (by contemporary standards) amounts of on-line direct-access storage.

The most widely adopted approach was that of context-sensitive help, whereby an interactive user could enter a predetermined key sequence (usually a question mark) at any point where the application was waiting on input. Of course, this only worked where the application design and the programmer had together conspired to provide some meaningful information to be displayed on request. Poor programming all too frequently resulted in a help message of the form:

```
EMPLOYEE NAME: ? "Enter the name of the employee"
```

when what the user really wanted to know was whether the initials were supposed to precede or follow the name, or if the name was supposed to match some other field, and so on. However, this system has merits, and is widely in use today, usually with multiple levels of information being made available. I.e., one request returns possible syntax of an entry, a second the value restrictions, a third a textual description of the purpose of the field. The major limitation with this approach is that the user is utterly dependent on what the program developer has decided is relevant, and has no alternative recourse except to consult any available written documentation, or to contact the program developer.

The other approach that has been widely adopted is the concept of the on-line manual, whereby the user is free to browse through a structured set of documentation made available on-line. This mechanism was slow to be adopted because of its massive requirement for on-line direct-access storage. Now that disk storage is cheap, comparatively speaking, this approach is being more widely adopted, particularly because it places little or no demand on the programmer. The on-line manual is typically accessed via one simple mechanism, and then behaves as a self-contained application. However, from the user's point of view, the on-line manual suffers from all the deficiencies of paper documentation: obscurity, poor layout, useless or non-existent indexes and/or tables of contents, inconsistency, and omissions. An interesting recent study[1] also revealed that many users fail to exploit the capabilities of even the best designed on-line information systems, because they fail to perceive the structure of the on-line information, even when it is designed to be *intuitively* obvious.

Cognos' products to date have all followed the approach of providing multi-layered, context-sensitive help facilities. However, the technical writing group have recently produced a working prototype of an on-line documentation system, which includes 'smart' indexing and searching, as well as editable examples that can be executed against a demonstration database from within the help context. Also, the research division has now produced the first working prototype of a completely different kind of on-line help, an advisory system based on artificial intelligence techniques. It is this 'advisor' that is the subject of the remainder of this paper.

## Overview

The aim of the 'Advisor' project is to create a software advisor or assistant capable of answering questions, phrased in a minimally constrained English-like language, about the use of a typical Fourth Generation Language (4GL). For the prototypes we have chosen QUIZ, Cognos' 4GL report writer, as the experimental subject, both because it is a mature and reasonably well understood product, and because there is a large body of users with a well documented history of problems encountered using the product.

The 'Advisor' is a joint industrial-academic project between Cognos Incorporated and the University of Ottawa, with grants from the National Research Council of Canada, The Natural Sciences and Engineering Research Council of Canada, and the Ontario Ministry of Colleges and Universities. The work is being carried out at the University of Ottawa's Artificial Intelligence laboratory, by a mixed team of Cognos research group members, and University of Ottawa professors, graduate students and paid assistants.

The project was started in the Autumn of 1985, and is scheduled to be completed in the late Summer of 1987 with delivery of a full demonstration prototype system. The first year of the project was spent on theoretical research, with practical experiment starting early in the Summer of 1986. A proof of concept prototype was completed at the end of October 1986, and successfully demonstrated to the various sponsoring organizations in the following month.

The current development uses two different software environments running on two different computer systems, linked by a 10 megabit/sec Ethernet: Quintus Prolog running on a Sun/3 workstation, and IntelliCorp's KEE™ (Knowledge Engineering Environment) running on a Xerox 1186 (Dove) InterLisp machine. This separation was deliberately chosen to ensure that the research project investigated the relative merits of a number of different technical approaches, although experience to date indicates that we might have encountered severe difficulties in attempting to build the complete system to run on a single machine of the classes available to us.

It is worth noting that at our current stage of experiment enormous computing resources are required to implement our various mechanisms. Only when we have the full prototype completed will we start to address the issue of efficient implementation in a less specialized environment. Our ultimate

1 "Helping Users Find Help: Models of Online Help Organization"; Marjorie S. Horton, IBM Human Factors Center, in the ACM/SIGCHI bulletin for October 1986.

goal is to produce a system capable of running on an IBM PC/AT, or equivalent. Preliminary experiments in this direction indicate that this should be an achievable goal within a very few years, partly because of the appearance in this market of sophisticated AI tools, and partly because of the continuing trend in increasing hardware performance for a given price. It was only the comparatively recent availability of affordable, very powerful work-stations, epitomized by the Xerox Lisp machine, that made this project possible.

## Approach

We adopted a somewhat unusual approach as we embarked on the research project by consulting the records of Cognos' telesupport group for documentary evidence of problems encountered by QUIZ users. The important point was that these were end users of QUIZ, who were attempting to use it as the tool of choice to solve real-life data-processing problems. Their calls to the telesupport group could thus be construed to be typical of questions that any 'advisor' should be capable of answering.

We screened the questions for obvious irrelevancies, and then classified them according to a system which we developed. The majority of the questions were split between only two categories, the 'how do I ...' (HDI) and the 'why did ...' (WHY) or causal; other categories, including syntactic (SYN), explanation of obscure error messages (ERR), definitional (DEF), and hypothetical (HYP) accounting for the rest. The HDI's alone accounted for more than half the questions, and it was decided to make answering this type of question one major focus of the project. A second sub-project was initiated to tackle the next two most frequent categories, the WHY's and the SYN's. The SYN's (questions such as "can there be more than one ACCESS statement in a report?" or "what is the syntax of a FOOTING statement?") are the traditional meat of context sensitive help systems, but accounted for less than 15% of the total number of questions, indicating that what was needed did indeed lie outside the capabilities of contemporary help systems.

Once we had classified the questions, we made a detailed study of those falling into the categories of interest, further analysing them by topic. Perhaps the greatest beneficiaries of this exercise were our technical writers, who received a weighted list of six topics that clearly gave many users difficulties, and which therefore required new treatment in the reference manuals. Once we had classified the questions by topic, decided what problems appeared to characterize each, and what general characteristics could be inferred, we performed a final screening of the questions to eliminate those which could be classed as relating to very obscure issues or to technical tricks outside the mainstream use of QUIZ. This left us with a representative set of some 200 questions, and we defined our goal for the working prototype as the capability to answer any question which can clearly be shown to be directly related to one of these 200 questions.

Of course, in order to prove the correctness of an answer, one needs at least a model answer for each of the questions to act as a yardstick. We therefore circulated our 200-question set to members of various groups within the Cognos head office, including tele-support (where the questions originated), product development, and a variety of technical specialists. Interestingly enough, in many cases there proved to be no one correct answer, indicating the depth of the problem when using one of the sophisticated software tools that characterize a 4GL. Where more than one answer was produced for any given question, we had the experts weight each answer and recorded the set of answers, ordered by weight, with the question.

At this point we had the necessary information to start designing possible question answering mechanisms, and were able to embark on the real work of the project. We eventually broke the project down into four sub projects, and treated each as a relatively independent development up to the first (proof of concept) prototype stage (P1). These four sub-projects were:

- a parser capable of syntactically analysing any question asked by the user of the system, couched in a formalized English-like language, and integrated into a user interface;
- an HDI question-answering system;
- a causal (WHY and SYN) question-answering system; and
- an 'apprentice' system with the capability of learning from the user.

Each of these is briefly described in the following sections of the paper.

## The Parser and User Interface

The parser was developed as a Master's thesis under the guidance of a natural language specialist (Dr. Stan Szpakowicz) at the University of Ottawa. The parser is designed to accept four types of input, as follows:

- an assumption or descriptive text fragment providing background information which may help to clarify the context of a question. These are entered in a restricted form of English (see below), delimited by a full stop, and can be annotated with various special symbols to help the parser. Examples of such assumptions are:

```
'employees' is a keyed file .
'emp_num' is a key of 'employees' .
'surname' is a key of 'employees' .
```

- a question, which is also entered in restricted English, but delimited by a question mark. Questions can also be annotated with various special symbols to help the parser. An example of a typical question (in the context of the previous assumptions) is:

```
how do i access 'employees' using 'emp_num' and
'surname' ?
```

  another example (showing some annotation to help the parser correctly identify dependent clauses) is:

```
how do i print an item at [the beginning of a
detail line] ?
```

- a fragment of QUIZ code, presumed relevant to the questions being asked. Only minimal syntax checking is performed on the code fragment.
- the literal text of an actual QUIZ error message about which questions are being asked.

The subset of English that the parser is designed to accept is in fact a formalism specifically designed for knowledge representation called LESK: Language for Exactly Stating Knowledge. The LESK language has been developed over a number of years by Dr. Skuce of the University of Ottawa, and the work that had already gone into the specification of this language was one of the starting points for the Advisor project. In practice, LESK had to be considerably extended to make it fit our particular application, but its straightforward simplicity is easy to pick up, and once mastered, LESK can be used to express a wide variety of concepts. Any limitation in vocabulary is a relatively trivial problem to solve, requiring only additional entries in the lexicon; the grammar is sufficiently extensive to cope with the variety of constructions needed to state assumptions and frame questions about programming with QUIZ.

A question on its own, or a question accompanied by one or more assumptions, and/or a fragment of QUIZ code, and/or a QUIZ error message constitute a logical query, which is the unit of input to one of the question-answering systems. The parser outputs a successfully parsed fragment in the form of a parse tree, represented either as a Prolog list or as a Lisp S-expression according to a user specification. In the case of the parser failing to parse an input fragment, an interactive dialogue is initiated with the user to attempt to resolve the problem. The user is given the choice of respecifying the fragment (abort parse for this fragment), either ignoring or altering the specific word over which the parser has stumbled, or of undertaking a more technical dialogue via which a new word can be defined in the parser's lexicon.

The parser itself is written in Quintus Prolog, and runs under UNIX on a Sun/3 workstation. The design of the current implementation is based on some fairly standard natural language processing theories for syntactic analysis, driven by a sophisticated lexicon, and is capable of parsing any input fragment in between 0.1 and 0.5 seconds.

Building the lexicon[2] proved to be one of the most interesting exercises, in as much as we wrote a special concordance program through which we fed a large body of machine-readable documentation, including the complete text of the QUIZ reference manual. The output from this gave us some 1,500 words for inclusion in the lexicon, together with all the necessary references and annotations required to complete each lexical entry. This linguistic analysis of existing documentation raised a number of interesting side issues, chief among which was that, while natural language theory suggested that we might expect some 400-450 different verbs to have been used, in fact the total was only about 150. When we investigated this with the technical writing group, we discovered that the technicians in the development group who had vetted the documentation for technical accuracy had insisted on particular usage of a variety of technical terms, and it was this insistence that had led to the relative paucity of the vocabulary.

Much more interestingly, because each of these words was serving multiple duty, they tended to gain a variety of context-dependent meanings, which turned out to be the root cause of a number of the misunderstandings demonstrated in our sample questions. Since the QUIZ documentation has won awards as the best documentation of its kind, this is clearly not an isolated problem, and indicates not only the need for early involvement of technical writers in the product development cycle, but also that formal technical glossaries must be developed which assign unique and unambiguous meanings to each technical term. Interestingly enough, one possible application of the Advisor prototype is to the development and formalization of such technical glossaries for future products.

The parser development thus proved to be an interesting exercise, and the working prototype has proven to be fast and effective in use. It does constrain the user somewhat with respect to freedom of expression on input, but experiment and constant usage over several months have shown that it is easy to adapt to the parser's constraints, and that these do not interfere in any important way with the user's freedom of expression.

## The HDI Question-Answering System

The How-Do-I question-answering system (HDI) is the area into which the most effort has been directed, mainly because we wished to explore the capabilities of available large-scale AI tools. For a variety of reasons, we eventually settled on the Knowledge Engineering Environment (KEE™, from IntelliCorp™) running under Interlisp-D on the Xerox-1186 (Dove) Lisp machine. This has proved to be a nearly ideal prototyping environment, but its tremendous richness resulted in a long and relatively steep learning curve which we believe will take some 12 months to travel to the level of neo-expert.

Using these tools, we have developed a question-answering system that utilizes three fundamental mechanisms: frame-based representation of knowledge; a forward-chaining, rule-driven inferential system; and object-oriented methods written in InterLisp. These three mechanisms are combined in various proportions to answer a submitted query in three stages, as follows:

- Using an approach known as semantic interpretation, the parsed S-expression for a user query is received over the Ethernet, and translated into a formal representation using KEE frames, based on the system's *understanding* of the query. Understanding, in the context of semantic interpretation, means being able to classify all phrases of the input query in terms of concepts defined in the knowledge base, and correctly assigning all objects into rôles which fit the activities mentioned.

  The semantic interpretation phase can fail completely (I don't understand the question) or may start an interactive dialogue with the user to clarify and disambiguate phrases of the input query which are causing confusion.

2 A lexical entry is similar to the entry for a word in an English language dictionary, but with a variety of annotations describing the part of speech, and possible rôles that the word can play.

- Once the semantic interpreter has completely transformed the query into a conceptual representation as a KEE knowledge base, it invokes a mechanism that attempts to match the query against known knowledge. This matching mechanism may yield a perfect match (the query is completely answerable), a partial match (in which case the user will be solicited to supply additional information), or a mis-match (unable to answer the query). All queries must eventually reduce either to an acceptable match, or a rejection of the query as unanswerable in its submitted form.

  An acceptable match is comprised of a set of potentially useful KEE units, each of which is deemed to hold information relevant to answering the query.

- In the case of an acceptable match, a syntax generating mechanism is invoked, which assembles and displays an answer from information held in the set of potentially useful units. This mechanism has sufficient knowledge of QUIZ syntax to enable it to formulate an answer in the form of QUIZ statements, although the resultant code may have instantiation points representing generalities (e.g. `<report item>`) and unknown specifics (e.g. `<file name>`); these are represented in angle brackets.

  In addition, or as an alternative, to the QUIZ code generated (the definition of an HDI question is one for which an adequate answer is a piece of QUIZ [pseudo-]code), the answerer may generate a plain text message. Typically, this occurs in the cases where the query has an explicit negative answer, e.g. "It is not possible to report a heading to a subfile".

In fact there is no clear boundary between these three stages, and the decision as to which mechanism should be responsible for what has tended to be taken on the basis of implementational ease. However, it helps to think of the process broken down in this fashion, because the major consideration is the construction of generalized mechanisms driven from separate knowledge bases. The more domain-specific (QUIZ in our case) knowledge there is hard coded within the actual mechanisms, the less useful (read: extensible, portable, maintainable, etc.) the system becomes. A brief example may serve to highlight some of these ideas.

The previously discussed query:

```
'employees' is a keyed file .
'emp_num' is a key of 'employees' .
'surname' is a key of 'employees' .
how do i access 'employees' using 'emp_num' and
'surname ?
```

Is translated by the parser into the following set of Lisp S-expressions, which are combined into a single list before submission to the question-answering mechanisms:

**Assertion 1:** `(is_a yes (variable employees) (count_nounphrase 1 () keyed_file (variable employees) ()))`

**Assertion 2:** `(is_a yes (variable emp_num) (count_nounphrase 1 () key (variable emp_num) ( (of (count_nounphrase 1 () keyed_file (variable employees) ()))))))`

**Assertion 3:** `(is_a yes (variable surname) (count_nounphrase 1 () key (variable surname) ( (of (count_nounphrase 1 () keyed_file (variable employees) ()))))))`

**Question:** `(hdi yes () access ((nil sub (personal_pronoun i)) (nil d_o (count_nounphrase 1 () keyed_file (variable employees) ())) (nil using (and (count_nounphrase 1 () key (variable emp_num) ((of (count_nounphrase 1 () keyed_file (variable employees) ()))))) (count_nounphrase 1 () key (variable surname) ((of (count_nounphrase 1 () keyed_file (variable employees) ()))))))))`

Careful examination of the above S-expressions reveals that all the information represented in the assertions is duplicated in the question. In fact it would be possible to generate an identical question S-expression by posing the following question without any assertions:

```
how do i access a keyed file 'employees' using a
key 'emp_num' of 'employees' and a key 'surname'
of 'employees' ?
```

LESK allows the user considerable flexibility and freedom of expression when formulating queries. The Semantic Interpreter takes the S-expression list output by the parser, and creates a knowledge base containing the following units:

**01** an ACCESSING activity, with a target of 02, and an agent of 03;

**02** a FILE object, named 'employees', with an access-mechanism of 'keyed', a key-list of 04 and 05, and membership in the class of DATA-FILES;

**03** a set-of KEYS, pointing to 04 and 05;

**04** a KEY object, named 'surname', with membership in the class of DATA-ACCESS-KEYS;

**05** a KEY object, named 'emp_num', with membership in the class of DATA-ACCESS-KEYS.

A number of inferences have been made, including the fact that the keys are data access keys rather than sort keys, and that the file is a keyed *data* file. In this latter case, while the 'keyed' attribute can be readily inferred even when it is not explicitly stated, the 'data file' attribute is not obvious (there are a number of possible alternatives). Where disambiguation proves difficult, the Semantic Interpreter either asks for confirmation, e.g.

```
Please confirm that the referent: FILE
(employees) is a member of the class of DATA-
FILES (Y)/N ?
```

or leaves the problem to be tackled by a subsequent processing phase. At present the Semantic Interpreter draws the inference and requests user confirmation, but this has caused problems in more complex situations, and we may choose to let this type of problem stay unresolved until a subsequent phase.

The second (matching) phase extends this first representation, by altering existing units and generating new units, to generate the following set of potentially useful units:

- **01** an ACCESSING activity, with a target of 02, and a sub-activity of 06;
- **06** a LINKING activity, with a target of 07 and 08, and an agent of 03;
- **02** a FILE object, named 'employees', with membership in the class of PRIMARY-DATA-FILES;
- **07** a FILE object, named 'employees', with an access-mechanism of 'keyed', a key-list of 04, an alias of '<alias-for-surname>', and membership in the class of SECONDARY-DATA-FILES;
- **08** a FILE object, named 'employees', with an access-mechanism of 'keyed', a key-list of 05, an alias of '<alias-for-emp_num>', and membership in the class of SECONDARY-DATA-FILES;
- **03** a set-of KEYS, pointing to 04 and 05;
- **04** a KEY object, named 'surname', with membership in the class of DATA-ACCESS-KEYS;
- **05** a KEY object, named 'emp_num', with membership in the class of DATA-ACCESS-KEYS.

Notice that the name of an object is merely another attribute of a unit and not its identifier, thus allowing three FILE units to exist for the same file. At this point the query can be classed as fully matched, because all the concepts mentioned have been successfully matched with known concepts, with sufficient precision to generate an answer. The answer generating phase is invoked, and in fact increases the set of potentially useful units with a number of units to represent syntactic fragments. The new units are:

- **09** an ACCESS-CLAUSE, with a target of 02, and a link-list of 10 and 11;
- **10** a LINK_CLAUSE, with a target of 07, and a linkage-spec of 12;
- **11** a LINK_CLAUSE, with a target of 08, and a linkage-spec of 13;
- **12** a LINK-BY-KEY-SPEC, with a file-ref of 07, and a key-ref of 04;
- **13** a LINK-BY-KEY-SPEC, with a file-ref of 08, and a key-ref of 05.

These are much simplified representations, but show the type of cross-references that are constructed, together with a degree of redundancy that eases navigation between units in the knowledge base. From these syntactic fragments, the following QUIZ pseudo-code is generated and displayed in the answer window:

```
ACCESS EMPLOYEES &
LINK TO SURNAME OF EMPLOYEES ALIAS <alias-for-
surname> &
LINK TO EMP_NUM OF EMPLOYEES ALIAS <alias-for-
emp_num>
```

This is the correct answer to the query, but will require the user to code the actual alias names to be used before the statement becomes executable.

The P1 prototype is capable of answering a handful of queries on the topics of accessing (the previous example is a typical such query), reporting and sorting. Achieving this required knowledge representation for a total of some 85 concepts (both objects and activities) plus about a dozen syntactic fragment specifications, supported by some 70 rules and several dozen Lisp methods. However, the fundamental mechanisms appear readily extensible, and adding knowledge about new concepts should prove merely time-consuming.

## The Causal Question-Answering System

The causal question answering system, known as QAUZ, is the subject of a Ph.D. thesis by Branka Tauzovich, a member of Cognos' Research Division on scholarship at the University of Ottawa. The requirements for originality in a doctoral dissertation necessitated keeping this portion of the project independent, and it has thus been treated as a parallel development. The QAUZ system has been implemented entirely in Quintus Prolog on a Sun/3 workstation; its only direct interaction with the rest of the Advisor is that it invokes the same parser to process its raw input.

The QAUZ system typically accepts a fragment of QUIZ code as part of a query, because the usual WHY question requires the code in order to be able to establish the context in which the query is being asked. For ERR type questions (explaining error messages), should it prove impossible to answer the question without reference to the context, the user will be prompted to enter the code. In the prototypical implementation no provision was made to allow direct access to machine readable QUIZ code, but this would certainly be the approach adopted for any more serious implementation. The text of QUIZ error messages are also acceptable as input, because one major category of user question is the "why did such and such an error occur?".

The QAUZ system comprises a knowledge base ( implemented mainly in declarative rules, but augmented with a custom frame-based mechanism), which describes a mini-QUIZ model (a subset of QUIZ similar to that used in other parts of the Advisor project), a user interface with links to the parser, a forward-chaining inference mechanisms, a backward-chaining inference mechanisms, a rule base, and a variety of tools such as frame compilers. The system uses a combined forward- and backward-chaining strategy to reason forward from a question, and backward from actual QUIZ code, to generate an explanation which satisfactorily accounts for the reported facts.

While processing a problem, the QAUZ system constructs an internal representation of the problem state which shows all the various dependencies and causal mechanisms that may be contributary factors. Even if a focused answer cannot be reached by the system, a browsable display of the internal representation of the problem frequently yields sufficient information to enable the user to answer the query. The following example shows the QAUZ system reasoning about the circumstances under which a default heading should appear on a report; note that the system is capable of handling inconsistencies:

```
Enter your QUIZ code:
             >> REPORT X HEADING "Name"
             >>

> Why is the column heading set to the dictionary
  heading?

Your query is INCONSISTENT!

   Contradiction found regarding:
       the column_heading of a report_item is set
       to the dictionary_heading of an item

Either your QUIZ statements are incorrect
or you did not formulate your question properly!

Assuming the latter...

The NEGATION of your question is TRUE
because the following DOES NOT HOLD:
  all record_items are requested for reporting (f),
  CONSEQUENCE OF:
   using 'REPORT with the ALL_option' statement (f)
  AND
   a record_item is an accessible record_item (?),
   CONSEQUENCE OF:
    the record_item is a declared record_item (?),

    the record_item is a record_item of a file (?)
    AND
     the file is a declared file (?),
     CONSEQUENCE OF:
      using first_file_declaration in 'ACCESS'
      statement (?)
      OR
      using subsequent_file_declaration in
      LINK_TO_part in 'ACCESS with a
      LINK_TO_option' statement (?)
 OR
  a default for the column_heading of a
  report_item is used (f),
  CONSEQUENCE OF:
   using format_part without HEADING_part
   in report_item
    in report_items
     in report_group
      in 'REPORT with a report_group' statement (f)

> Why is not the column heading set to the
dictionary heading?

Because the following DOES NOT HOLD:
  all record_items are requested for reporting (f),
  CONSEQUENCE OF:
   using 'REPORT with the ALL_option' statement (f)
  AND
   a record_item is an accessible record_item (?),
   CONSEQUENCE OF:
    the record_item is a declared record_item (?),
    CONSEQUENCE OF:
     the record_item is a record_item of a file (?)
     AND
     the file is a declared file (?),
     CONSEQUENCE OF:
      using first_file_declaration in 'ACCESS'
      statement (?)
      OR
      using subsequent_file_declaration in
      LINK_TO_part

 OR
  a default for the column_heading of a
  report_item is used (f),
  CONSEQUENCE OF:
   using format_part without HEADING_part
   in report_item
    in report_items
     in report_group
      in 'REPORT with a report_group' statement (f)
```

The fundamental mechanisms of the QAUZ system are capable of dealing with all the various categories of question discussed in the Approach section of this paper. However, only the WHY, ERR, HYP, and SYN categories have actually been addressed in detail, and these only for the defined mini-QUIZ subset of QUIZ.

Obviously, the QAUZ and HDI systems overlap in capability, but the need to keep the QAUZ system independent of the rest of the project has allowed us the luxury of developing two totally separate approaches. The second prototype (P2) will benefit from the work done in both systems.

## The QUIZ Apprentice

The QUIZ Apprentice is a parallel project being undertaken at the University of Ottawa by a team led by Dr. Stan Matwin, and is not formally a part of the Advisor Project. However, it sprang from work started by Dr. Matwin in the early stages of the Advisor Project, and there are close ties between the two teams.

One of the chief reasons for the separation is that the Apprentice has been implemented entirely in Prolog, and as yet we have found no way to tightly couple Prolog with a Lisp-based knowledge representation mechanism. The Apprentice thus has its own internal model of QUIZ, complete with its own knowledge base constructed entirely in Prolog. In its earliest incarnation it used approximately the same subset of QUIZ as was chosen for the HDI question-answerer.

The prime characteristic of the apprentice approach is that it is capable of learning. For a rule-based system, this means that the operational system can create new rules as a standard part of its normal processing, as well as extending its knowledge with each new solution generated. In practice, the system designer seeds the knowledge base with a basic set of rules, using an interactive teaching mechanism to do this. This infant system can then be introduced into a user environment. Whenever it encounters a new problem for which it is unable to find a solution, or when it offers an incorrect solution, the user can enter the interactive teaching mode and help the system define a set of rules which, in

- Transferring the existing system into a smaller self-contained environment running on a computer such as the IBM-PC/AT.
- Replacing the current independent syntactic parser with extensions to the semantic interpreter.
- Development of deeper models of QUIZ, to enable greater understanding of queries by the system, and thus generation of more sophisticated answers.
- Focusing on a single topic, such as file access, and attempt to create an advisor with effectively complete understanding of this one topic.
- Development of more sophisticated reasoning strategies, capable of dealing with incomplete queries.
- Development of alternate answering mechanisms, to enable the user to get more out of the system than one blunt answer.

This last problem is really the heart of the design problem. It is very easy to develop a system that answers a question of the form 'Can you tell me the time?' with a blunt 'yes'; this is of little use, and serves only to irritate the user. It is essential that an advisory system be capable of interpreting the reason behind a question that is asked, and thus generating a meaningful answer, rather than blindly answering the literal interpretation of the question that was asked.

It used to be that the way to test whether a computer system was working was to pose the same problem repeatedly, and check that the answers were consistent, which usually meant identical. Perhaps *the* distinguishing feature of a knowledge-based system is its ability to learn. This means that the second time a question is posed the response should at the very least be faster, and that multiple repetitions should trigger some alternate action. In an ideal system this would include proposing alternate approaches in an attempt to help the user reach a solution to the real as opposed to the expressed problem.

Given the limited lifespan of the current project, it is unrealistic to expect that we will solve more than a fraction of these problems. However, it is to be hoped that we will make some forward progress, and perhaps come closer to the currently perceived ideals.

## Practical Implementations

Thus far this paper has been concerned mainly with reportage of what has been achieved, and a description of possible short-term research goals. Quite clearly, the P1 and P2 prototypes have little direct application in the resource-conscious practical data-processing world, relying as they do on costly and largely non-standard dedicated hardware and software environments. However, although the goal of research is knowledge, there are some clear pointers to what the future holds in store.

Firstly, these technologies are practical. That is to say, it *is* possible to build a knowledge-based system capable of doing real work in real environments. The problem is that at present it costs hundreds of thousands of dollars to implement such systems, which makes them cost-effective only in the most specialized of applications. Most of this cost is sucked up by the need for highly skilled development personnel, expensive tools, and the fact that very large machine resources are typically consumed by the operation of such systems.

Today, machine costs are falling, and the latest generation of chips (DEC MicroVax, Intel 80386, Motorola 68020, National Semiconductor 32032, TI Explorer, etc.) offer more than adequate raw power. Once this power has been packaged in suitable architectures, it will become an easy matter to boost a conventional system with an AI co-processor, much as floating-point co-processors have been integrated into conventional architectures. In addition, the cost of stand-alone micro-systems will continue to fall, at least in terms of their price/performance ratios. The machine resource problem will disappear as certainly as has the memory problem over the last few years.

Todays costly tools are rapidly being migrated from the esoteric Lisp machine environments of their engendering to the relatively conventional UNIX world, and will soon be available implemented in conventional programming languages such as C. At the same time, the primitive first generation PC-based AI tools are giving way to sophisticated second and third generation offerings that are beginning to rival the big tools. History tells us that competition will trigger falling prices until levels deemed acceptable by the user community are reached.

The problem of skilled personnel for development will also diminish as more and more sophisticated tools are developed. A logical extention of the QUIZ Apprentice is a system that can be taught by a domain expert who can be relatively ignorant of the internal mechanisms. This problem is analagous to the problem of data-base specialists, who are still required in sophisticated environments, but who can be ignored when simple applications are implemented using sophisticated DBMS systems.

Knowledge-based systems are a reality. Within three to five years they will become the norm. Should a QUIZ advisor be considered a reasonable product, a commercially attractive implementation capable of running on a dedicated IBM-PC/AT or equivalent could be brought to market within three years, i.e. first quarter 1989. It is by no means clear that such a product has a market, but the underlying technology has many applications, and a few of these are introduced in the following paragraphs.

Although true advisors are a possibility, they are unlikely to appear integrated into applications (in the same fashion as current help facilities) for several years yet. However, they are feasible packaged as stand-alone applications on dedicated hardware, which certainly introduces the possibility of a PC-based system. An alternative solution is to build a multi-user, server-type front end for a large-scale, central system dedicated to running one or more advisors, and allow users to query an advisor remotely via local network or telecommunications links.

A much more likely scenario is that, instead of an advisor, a teacher will be created. Computer-based training is a growing field, as the widespread adoption of computers, particularly at the desk-top level, exacerbates the problem of the relative scarcity of teachers. The AI community is currently focusing considerable attention on the problems of teaching, and AI is seen as the most likely technology to provide a quantum improvement over the

conjunction with its existing knowledge, will enable it to generate the correct solution.

The system's knowledge is based on problem-solution pairs, and all that is needed to teach the system is to supply the correct solution to a new problem. The apprentice not only stores the problem-solution pairs but is capable of reasoning, so that by decomposing problems into partial problems, and matching these against its knowledge base, it is capable of solving problems for which no explicit solution is stored. Although the underlying mechanisms are conceptually simple, even the initial prototype QUIZ Apprentice was capable of demonstrating remarkable performance.

Clearly the first user of an infant system needs to be fairly knowledgeable, in order to detect the situation where incorrect solutions are being offered by the apprentice. However, experiment has shown that the number of rules needed to adequately cover a given topic, while not fixed, can be quickly identified. In operation, the Apprentice rapidly builds up a few dozen rules (typically 2- or 3-) for a given topic; thereafter, new rules only need to be generated to cover exceptional cases. As soon as this point (where the rate of rule creation drops off sharply) is reached, the system can be turned over to a naïve user. Typically, from this point on there will be no incorrect solutions generated, and the system will only require teaching when new problems are encountered. When this occurs, the user can ask an expert to supply the correct solution.

The advantage of this approach is that it eliminates the problem of requiring a technical specialist to make changes in the system's stored knowledge, which is the major problem with most of today's operational expert systems. However, the more complex the fundamental knowledge representation and reasoning mechanisms, the harder it is to ensure that the rule construction mechanisms are correct: and the need to detect potentially incorrect solutions, at least in the early stages, introduces the much wider issue of the verification of knowledge bases. Not for nothing is this approach named 'apprentice', and it is important to remember that the step from apprentice to journeyman has always been a big one, requiring years of effort carefully directed by an experienced and capable master.

## The First Prototype (P1)

The first prototype (P1) was completed in October of 1986, and was successfully demonstrated to Cognos' Research management and to a committee from the National Research Council in mid November. The P1 system requires a Sun/3 workstation with at least 4 megabytes of real memory running under UNIX (System 4.2) with Quintus Prolog, linked via a 10 megabit Ethernet to a Xerox 1186 (Dove) Lisp machine with the maximum memory configuration (3.7 megabytes) running under Interlisp-D with IntelliCorp's KEE™ version 2.1. Both systems require access to at least 50 megabytes of individual disk space.

For P1, the various components of the system were linked together within a very basic user interface, written in a mixture of C and Quintus Prolog, and running on the Sun workstation. This user interface initializes all the communications mechanisms, controls the Lisp machine as a slave AI problem solver, and allows the user to enter, edit, debug, and submit (to the appropriate question-answering system) all the various components of a query. Although primitive in appearance and behaviour, this interface has proven to be a robust and indispensible tool for working with all the various components of the system, as well as acting as a prototype for the more ambitious interface planned for the second prototype (P2).

This system could be run entirely from the Xerox Lisp machine (the slave!) by opening an Interlisp window and running a terminal emulation via which the user interface on the Sun could be accessed. Queries could thus be entered in this window, and the rest of the screen used to display the question answering process in action.

For P1 we limited ourselves to a handful of questions on a few major QUIZ topics, namely: data access, reporting, sorting, and selecting (QAUZ only). A set of mechanisms that could be adapted to all these topics were developed, and based on the experience with P1 we expect little difficulty in extending the knowledge base to include new topics, as well as to cover these first topics more extensively. QUIZ has some 31 main verbs, and there is a rough equivalence between these and Advisor topics. The main purpose of the P1 prototype was to demonstrate the feasibility of answering questions, and to test a variety of implementation mechanisms. Both of these objectives were satisfied.

Although performance was not an issue for this prototype, it may be of interest to note that it took the KEE/Interlisp-based mechanisms between 4 and 20 seconds to answer each P1 question. This in a development environment with multiple levels of debugging active, and with extensive graphic traces being displayed. One of the P2 issues that we intend to address is the question of what sort of response might be considered reasonable for an advisory system.

The QAUZ system P1 prototype runs totally independently, and is self contained on a Sun/3.

## The Second Prototype (P2)

The second prototype (P2) is scheduled for July 1987, and will be based on the work done to date, once this has been extensively reviewed. At present it is too early to state with any certainty what will and will not be attempted, but a number of possibilities are open. These include:

- Integrating the entire system into a single machine environment. This will require the availability of new tools, notably KEE on the Sun/3.
- Coping with procedural or time-dependent problems.
- Extending the P1 prototype with lots of knowledge to give coverage of the majority of QUIZ rather than the limited subset used for P1.
- Developing a sophisticated user interface, taking full advantage of windows, interactive graphics, and other available techniques.
- Closely coupling Prolog with Lisp-based knowledge representation mechanisms.

rather pedestrian on-line tutorials available today. The key to a good teaching system is not its ability to detect correct answers, but its capacity for understanding *why* a wrong answer has been entered, and for developing a strategy to correct the user's misaprehension. The same knowledge required to implement an advisor serves to implement a teacher, although in the case of the latter this must be augmented with knowledge about the types of mistakes that can be made. Typically the ratio of information is about 8:1; for every correct fact there are roughly eight possible misunderstandings.

One direct application is to a tool to help software product designers keep their facts straight, and ensure that all the various components are functionally complete and correctly interconnected. This includes the development of technical glossaries in which each term is uniquely and unambiguously defined. Although such a system is a design tool, aimed at ensuring the correctness and completeness of a software product, it could also be extended to create outline technical documentation that conforms to a standard structure. By the same token, such a system could also ultimately be used to proof final documentation for completeness, accuracy, and consistency. The key to making any such system work will be ensuring that it is sufficiently easy to use, and directly beneficial, that development teams will be careful to keep its knowledge current and accurate. The major flaw in most paper-based design tools has been that the effort required to keep the paper up to date has tended to detract from the true development tasks.

Beyond such design aids, which ought to be equally useful to all builders of software applications, lie some very exciting possibilities. It is only a short step from an advisor which answers a 'how do I' question with a piece of pseudo-code to a system capable of generating executable code. This would allow users to program a system at a very high level, by expressing their needs in a fairly informal fashion. With an adequate control structure, the AI-based programming tool could help the users refine their ideas by posing questions until the application was completely defined. The major step forward needed to make this sort of tool a reality is the development of knowledge bases describing typical applications, but it transpires that a significant portion of the knowledge needed to implement an advisor is just this.

The key to all these systems lies in the user interface, which must allow the user to communicate with the system in a natural fashion that is as easy to use as the natural language and scribbled diagrams we use to communicate our ideas to other humans. The human tendency to make life as easy as possible means that we seldom use a long form when a shorter is available; typing true English at a keyboard is a non-starter, if quicker mechanisms are available. Even if affordable voice input technology makes spoken natural language a possibility, we will still need the ability to diagram, annotate, doodle, and so on. It is the availability of these latter mechanisms that will determine the time-scales for the widescale introduction of true AI-based tools.

Within two years, high resolution (1276 x 1024 minimum) colour graphic displays will be achieving affordable levels. We already have excellent interactive pointing devices. Three years will see the arrival of reliable voice input technology; it is already affordable, but has too low an accuracy rate to be really useful. In the same timescale very large data storage (optical disks), high quality hard copy devices (laser printers), and very fast desk-top architectures will all have achieved widescale commercial viability. All that is missing is the software, first its development and then its acceptance into the market place. Today, it exists only in fragmented prototypical form in research establishments around the globe.

A useful model that we can examine for what to expect is the introduction of the graphic desk-top metaphor pioneered by Xerox on its Star systems. Although demonstrably viable, it was far too expensive for all but a handful of users, and even the far more affordable Apple Lisa failed to achieve widespread acceptance. The introduction of the Apple Macintosh, however, offered the necessary price break-through that triggered both widespread acceptance of the technology, and a trend towards making an interactive, iconographic interface standard on all systems. The Macintosh is not yet four years old, at least in terms of its market availability. By analogy, we can expect today's high priced experimental interfaces to reach the mass market level in the same timescale as the necessary hardware becomes affordable. Another two to three years should serve for such products to gain credence similar to that now afforded the Macintosh. Thus, by 1992 it is very probable that systems based on current experimental advisor technology will be commonplace.

## Acknowledgements

To the National Research Council of Canada, the Natural Sciences and Engineering Research Council of Canada, and the Ontario Ministry of Colleges and Universities for various grants which have helped make this project possible.

To the University of Ottawa for its joint sponsorship of this project.

To Drs. Skuce, Matwin, and Szpakowicz of the University of Ottawa, and Dr. Oppacher of Carleton University for their participation in this project, and for advice freely given.

To Branka Tauzovich, Charles Truscott, Sylvain Delisle, Deborah Lazar, Patrick Constant, Zbigniew Koperczak, Yannick Toussaint, and Claude Queant for their technical contributions, both theoretical and practical.

To Bob Barr, Ian Craib, Al Slachta, and Phil Archdeacon, creators of the QUIZ on-line manual.

## References

[Cline et al.]
Cline, T.; Fong, W.; Rosenberg, S.: "An Expert Advisor for Photolithography". Procs., IJCAI-85, pp. 411-413, 1985.

[Constant et al.]
Constant, P.; Matwin, S.; Szpakowicz, S.: "A Question-Driven Approach to the Construction of Knowledge-Based Software Advisor Systems". Procs., Third IEEE Conference on AI Applications, Orlando, Florida, February 1987.

[Fargues and Adam]
Fargues, J.; Adam, J.P.: "KALIPSOS: A Text Processor for Knowledge Acquisition". Presented at the IBM Europe Linguistic Seminar, Davos, Switzerland, 1984.

[Fargues et al.]
Fargues, Jean; Landau, Marie-Claude; Dugourd, Anne; Catach, Laurent: "Conceptual Graphs for Semantics and Knowledge Processing". IBM Journal of Research and Development, Vol. 30, No. 1, January 1986.

[Fikes]
Fikes, R.E.: "ODYSSEY: A Knowledge-Based Assistant". Artificial Intelligence, Vol. 16, No. 3, 1981.

[Fikes and Kehler]
Fikes, R.; Kehler, T.: "The Role of Frame-Based Representation in Reasoning". Communications of the ACM, Vol. 28, No. 9, pp. 904-920, 1985.

[Gomez]
Gomez, F.: "A Model of Comprehension of Elementary Scientific Texts". Procs. Theoretical Approaches to Natural Language Understanding, Halifax, NS, 1985.

[Hiz]
Hiz, H. (Editor): "Questions". Dordrecht, 1978.

[KEE]
The KEE™ (Version 2.1) Technical Documentation Set for Xerox (Interlisp-D) machines, IntelliCorp™, Palo Alto, 1985:

[Kiefer]
Kiefer, F. (Editor): "Questions and Answers". Dordrecht, 1983.

[Lehnert]
Lehnert, W.: "The Process of Question Answering". Hillsdale, NJ, 1978.

[Matwin et al.]
Matwin, S.; Skuce, D.; Szpakowicz, S.: "Question-driven Approach to the Design of a Software Advisor System". Department of Computer Science, University of Ottawa, working paper, 1985.

[Matwin & Queant]
Matwin, S.; Queant, C.: "Knowledge Acquisition by Simple Learning in a QUIZ Programmer's Apprentice". To appear in the Procs. of the 2nd International Conference on Computers and Applications, IEEE Computer Society, Beijing, June 1987.

[Queant]
Queant, C.: "A QUIZ Apprentice". Department of Computer Science, University of Ottawa, Thesis TR-86-13, 1986.

[QUINTUS]
The Quintus Prolog Technical Documentation Set (Version 6), Copyright © 1986, Quintus Computer Systems Inc.:

[Schank]
Schank, R.: "Intelligent Advisory Systems". In: AI Business, P.H. Winston, K.A. Prendergast (editors), Cambridge, 1984.

[Skuce-83]
Skuce, D.: "The LESK Tutorial". TR-83-03, Department of Computer Science, University of Ottawa, 1983.

[Skuce-86]
Skuce, D.: "Natural Language Synthesis of a Database Reporting Language Using Commercial Expert System Technology". Working paper, University of Ottawa, December, 1986.

[Skuce et al]
Skuce, D.; Matwin, S.; Tauzovich, B.; Oppacher, F.; Szpakowicz, S.: "A Logic-Based Knowledge Source System for Natural Language Documents". Data and Knowledge Engineering 1, North Holland Publishing, 1985, pp. 201-231.

[Sowa]
Sowa, J.F.: "Conceptual Structures: Information Processing in Mind and Machine". Addison-Wesley Publishing Co., Reading, MA, 1984.

[Sowa and Way]
Sowa, John F.; Way, Eileen C.: "Implementing a Semantic Interpreter Using Conceptual Graphs". IBM Journal of Research and Development, Vol. 30, No. 1, January 1986.

[Szpakowicz et al]
Szpakowicz, S.; Matwin, S.; Skuce, D.: "QUIZ Advisor: A Consultant for a Fourth Generation Software Package". In procs. of the 2nd International Workshop on Expert Systems, Avignon, May 1986, pp. 155-168.

[Tauzovich]
Tauzovich, B.: "Representing Causal Relationships in an Expert Advisor for a Fourth Generation Language". Ph.D. thesis (in progress), Department of Electrical Engineering, University of Ottawa, 1986.

[Tauzovich-86]
Tauzovich, B.: "Representing Causal Relationships in an Expert Advisor for a Fourth Generation Language". Working paper, University of Ottawa, December, 1986.

[Tou et al]
Tou, F.N., Williams, M.D.; Fikes, R.E.; Henderson, D.A.; Malone, T.W.: "RABBIT: an Intelligent Database Assistant". Procs. AAAI-84, Pittsburgh, 1984.

[Wilensky et al]
Wilensky, R., Arens, Y.; Chin, D.: "Talking to UNIX in English: an Overview of UC". Communications of the ACM, Vol. 27, No. 6, pp.574-593, 1984.

[XEROX]
The Interlisp-D Reference Manual, Xerox Corporation, Palo Alto, 1985.

[Zarri]
Zarri, G.P.: "RESEDA: an Information Retrieval System Using Artificial Intelligence and Knowledge-Representation Techniques". Research and Development in Information Retrieval, Sixth Annual International SIGIR Conference, 1983.

# THE USE OF PHIGS IN AN ARTIFICIAL INTELLIGENCE ENVIRONMENT FOR MECHANICAL ENGINEERING

**Mike Thompson and Jim Roth**
CAD/CAM Technology Center
Digital Equipment Corporation
Chelmsford, Massachusetts

**Abstract**

The combination of artificial intelligence techniques with advanced computer graphics is an attractive approach to implementing mechanical engineering applications. This paper describes how PHIGS (Programmers Hierarchical Interface to Graphics) is relevant to our work in mechanical CAD. We describe our experience in implementing a subset of the standard and its use by Lisp programmers.

## MOTIVATION

We are a part of the CAD/CAM Technology Center (CTC) and are carrying out advanced development in CAD tools for special applications in mechanical engineering internal to Digital.

The Programmers Hierarchical Interface to Graphics System (PHIGS) will soon be an ANSI and ISO standard, see [1-4]. We have been investigating the suitability of PHIGS for our work by implementing a subset of the standard. Our experience leads us to conclude that the combination of PHIGS and Lisp suits our needs very well.

### Why Artificial Intelligence?

The application of AI tools in the area of mechanical engineer will emphasize richer data structures. Increasing the semantic content of the run time structures used by CAD applications is a precondition for finding solutions to problems such as routine design decisions, dimensioning and tolerance. CAD tools will need to deal explicitly with engineering objects rather than just geometry, see [5].

### Why graphics?

Mechanical parts are difficult to describe in words and the engineer must see three dimensional objects to understand them. The graphics software must be able to support interactivity such that modifications made by the engineer to the design will be displayed immediately. Ideally, the mechanical engineer should feel that the parts being displayed are 'really there'.

### Why PHIGS?

The CAD tool must support three dimensions for mechanical design (as distinct from mechanical drafting). PHIGS supports three dimensional modeling, viewing and user input.

A hierarchical structure is natural for mechanical CAD. Consider how parts move with assemblies and detailed features move with an engineering part. PHIGS models this structure very effectively. Changes in the hierarchy are reflected lower down. For instance, removal of a part also removes all the details of that part. Changing the spatial orientation of a part is achieved in PHIGS by making a single change and having this apply to the whole of the hierarchical structure below that point.

Although, in the past, mechanical engineering applications used mainly graphic information such as lines and text, more and more non-graphic information will be added in the future. Manufacturing requires information related to function, process and materials. It is difficult to reconcile high performance computer graphics with the use of data structures constantly being augmented by non-graphic information.

When considering the software design of data structures for storing engineering data and for driving the graphics, we may come to the conclusion that different requirements of access and performance necessitate different implementation. PHIGS implementations are able to optimize data structures concerned only with the graphics. However, there is an overhead in maintaining consistency between these two representations. This overhead may be acceptable if typical updates to the corresponding parts of the engineering and the graphics data structures are localized and small. To support this activity, PHIGS provides a rich mapping between the two.

Over the next few years, there will be increased availability to mechanical engineers of workstations designed with hierarchical graphics in mind. On such a workstation, a change to the PHIGS data will be immediately

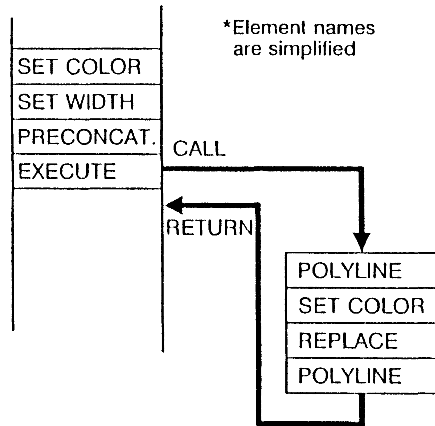



**Figure 1:** The execute structure element

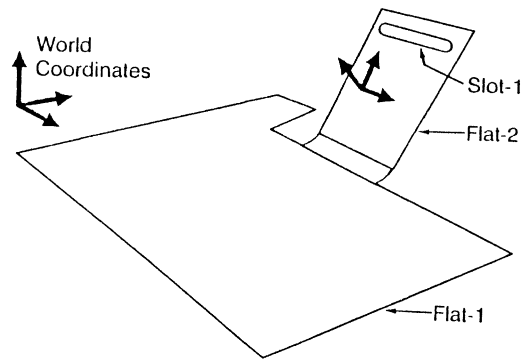



**Figure 2:** Sheet metal example

displayed. Such graphics hardware must be capable of completely traversing of the hierarchy many times a second.

### Why Lisp?

We chose Lisp because of its good runtime data structure capability. Lisp is suitable for fast prototyping because it can be run in interpretive mode. It also seems a good environment for other AI tools.

## EXAMPLES OF "HOW TO"

We shall now present some of the typical activities that a programmer will implement using PHIGS.

- Inheritance of attributes (color, location etc)
- Managing views
- Mapping between application and PHIGS
- Editing the display list
- Picking
- Dragging a structure
- Rubber banding

### Inheritance of Attributes

In order to display a mechanical CAD part, the programmer places in the PHIGS 'Structure Store', sufficient information to model the graphic appearance of that part. The programmer then 'posts' this model for display. The term 'PHIGS Structure Store' might give the reader the impression of a store containing objects that would presumably be coded in a declarative manner. This would immediately appeal to a Lisp programmer conversant with object oriented style. However this is not the case, and the contents of the structure store are more akin to assembly language code.

The term 'inheritance' and 'instance' used in the PHIGS standard are different from its meaning as used in object oriented programming, both references [1] and [7] have glossaries that clarify this.

Attributes of graphic entities include line style, color, and width. These attributes are like registers, the contents of which are pushed on occurrence of an **execute structure** command which is similar to a jump subroutine. They are accessible to the called routine and may be modified. They are restored on return from the structure.

In Figure 1, the programmer sets the color, line width and spatial orientation (by concatenating a transformation matrix), before making the **execute structure** command. Note that the attribute values that were set before the **execute structure**, will be restored after returning from the structure although they may no longer be needed.

PHIGS uses this 'assembly language style' to place as few restrictions as possible on the programmer who may be seeking maximum performance from the graphics hardware.

### Managing views

PHIGS places no restrictions on the layout of the views. Figure 3 shows an arrangement typical of mechanical CAD systems, which may be implemented using a 'top structure'.

In order to set up this structure, the programmer first prepares the view matrices. The views are then defined by references to the view indices from within structures. The top structure is created and then posted as a root and immediately PHIGS starts traversal of the hierarchy from this root and displays the views.

Panning and zooming in a view can then be achieved by repeatedly updating the corresponding view matrix.

### Mapping between application and PHIGS

Consider a CAD application that is displaying the sheet metal part shown in Figure 2. This simple part consists

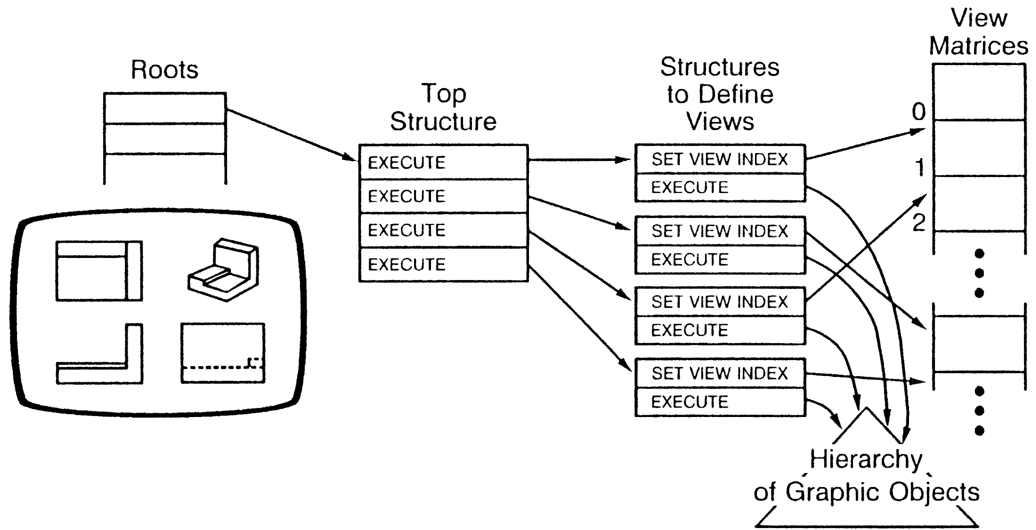



**Figure 3:** Managing four views with PHIGS

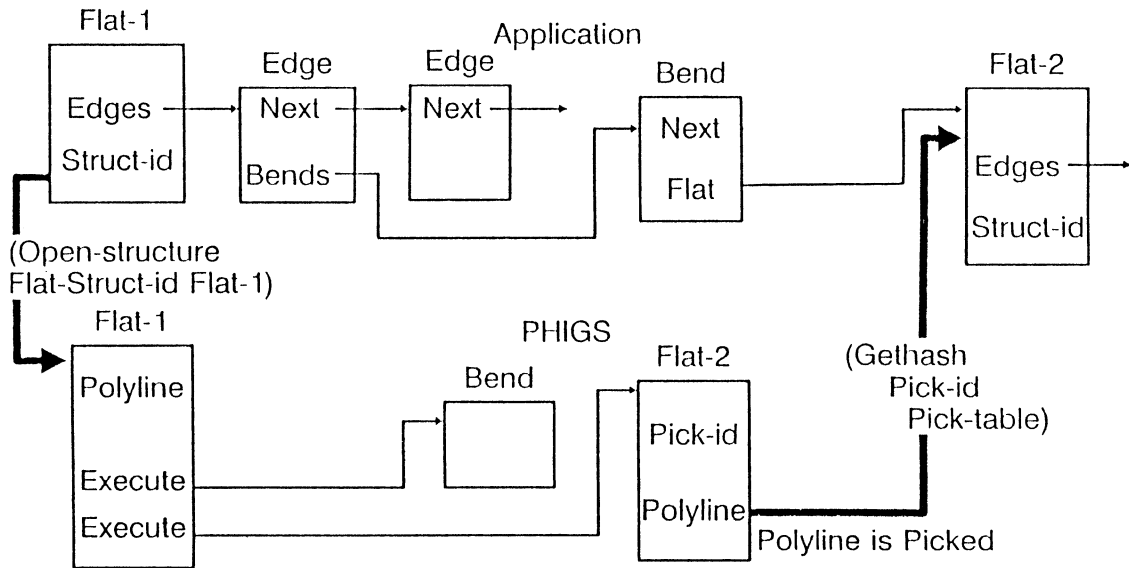



**Figure 4:** Mapping between Application and Graphics

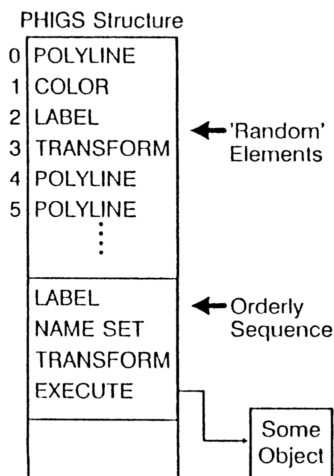



**Figure 5:** Orderly editing of the PHIGS structure store

of two 'flats' connected by a 'bend'. One of the flats has a slot stamped in it.

Figure 4 shows run time data structures that might be created for such a sheet metal part. The upper half of this Figure shows how the sheet metal part is modeled by Lisp structures for a flat, bend, and edge. Note how a flat points to a linked list of edges, and in turn the edges point to zero, one or more bends that are attached to them. In the lower half of the same Figure are shown run time structures created by PHIGS. This has a different structure with, for instance, edges represented by a single polyline. Also flats reference other flats directly using /bf execute structure. The data structures the programmer will create in the PHIGS structure will be optimized for display and user interaction.

PHIGS provides ample mapping capability between display list and engineering objects in the application achieved via structure id, labels, pick id and name sets. When going from application to graphics, (usually with the purpose of modifying the graphics) one uses structures identifiers and labels to get to a given location in the PHIGS structure store. This is done with the PHIGS command **open structure**.

When going from graphics to the application, (usually after the user has picked some graphic element on the display) one may use the pick identifier to map back into some application object. Figure 4 shows how this may be done using Lisp hash tables.

### Editing the PHIGS display list

The programmer may concentrate on the editing of the PHIGS structure store and let PHIGS deal with the display. To do this editing effectively, the programmer must keep track of what is in the structure store and devise some well organized way of modifying it without introducing faulty edits. Clearly it is difficult to keep track of random edits of structure elements. Figure 5 shows an orderly sequence of elements following a label. The transformation matrix is always the second element to follow a label.

Severe software maintenance problems are likely to result from having numerous software modules in the application carry out such edits. A better approach is to implement a data abstraction that does this editing and hides the offsets used from the higher levels of the application. Commonly used engineering objects may be represented by Lisp structures. For example, in an application to sheet metal design, we might have:

```
(defstruct bend
  first-flat
  second-flat
  angle
  radius
  length
  distance-from-origin-of-flat-1
...)
```

We write functions to create, delete, print and modify bends. These functions maintain the correspondence between Lisp and the PHIGS structures. It is our experience that such an approach works well when the data abstraction is built for a specific application.

### Picking

Consider the user picking the bottom edge of slot-1 with the mouse, in Figure 2. How the application deals with this probably depends on the context, for instance, it might decide that the user wished to pick the whole slot rather than just the bottom edge. The programmer can benefit from the PHIGS pick path capability in this situation.

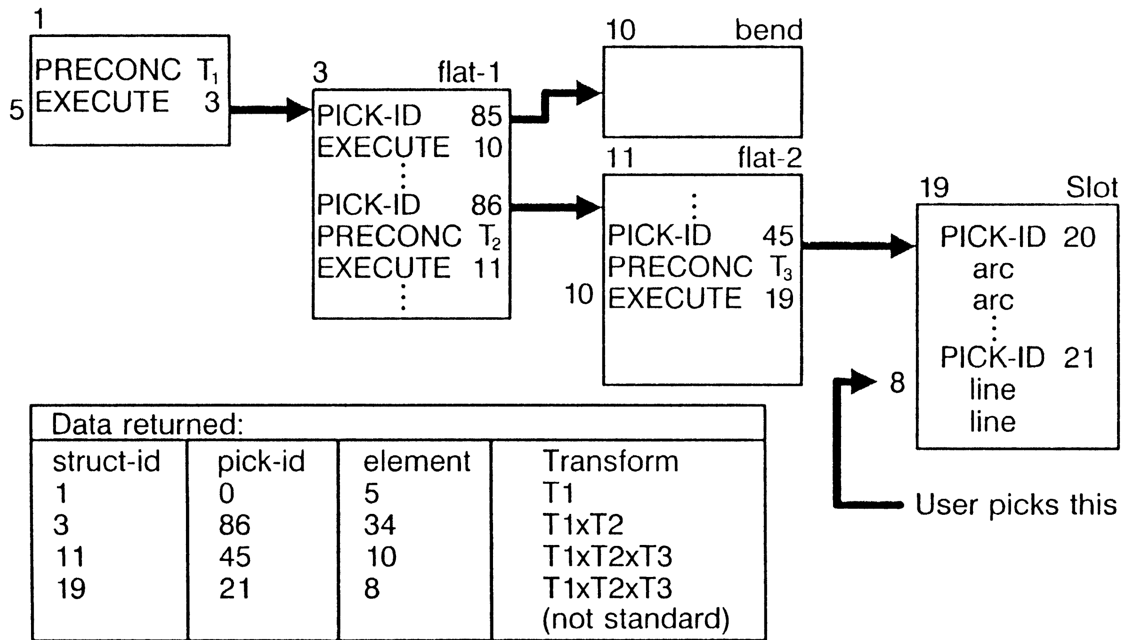
The pick path is a list of structures starting with the one picked and continuing up the hierarchy to the root. The PHIGS structures for the sheet metal part and the pick path returned are shown in Figure 6.

While debugging, the programmer can use the pick to check the structures that underlay the graphics being displayed. Figure 7 shows how the Lisp programmer could print out the pick path.

### Dragging a structure in a plane

Consider the following scenario. A CAD application displays the sheet metal part as shown in Figure 2. The engineer picks a line on slot-1. The application offers a menu of options that specifically apply to slots. The engineer selects the option to reposition the slot.

Figure 8 describes how we would do the dragging using a combination of PHIGS and fast workstation hardware. PHIGS make it relatively easy to implement activities such as this with application code that is independent of the view.




Data returned:

| struct-id | pick-id | element | Transform |
|---|---|---|---|
| 1 | 0 | 5 | T1 |
| 3 | 86 | 34 | T1xT2 |
| 11 | 45 | 10 | T1xT2xT3 |
| 19 | 21 | 8 | T1xT2xT3 (not standard) |

**Figure 6:** Structures and corresponding pick path

```
;;; A global buffer to receive the pick path. Avoids frequent
;;; space allocation but needs programmer discipline to ensure
;;; data is used before another pick is requested.
(defconstant *max-depth* 10)
(defvar *pick-xform*
(make-array
    (list *max-depth* 4 4)
    :element-type 'single-float))
(defvar *pick-path*
(make-array
(list *max-depth* 3)
:element-type '(signed-byte 32)))

;;;USER MAKES A PICK
(pprint (request-pick *ws* *max-depth* *pick-path* *pick-xform*))
```

**Figure 7:** Use of picking while debugging

```
Get the following:
    Transformation T mapping coordinates
        of flat-2 into world coordinates
        by concatenation of all matrices
        down path from root to flat-2.
    View matrix V computed from view
        parameters.
    Transformation M mapping coord
        of slot-2 into coords of
        flat-1.
    Current location of slot-1 on
        flat-2.

To drag, we repeat the following:
    Get locator's screen position L.
    Using T and V, map L to point P
        on flat-2.
    Calculate translation of P
        from previous P, and apply
to matrix M.
    Replace old matrix M with new
        matrix M.
```

**Figure 8:** Dragging in PHIGS

```
Get the following:
    Transformation T to map the
        coord system of flat-2
        into world coordinates
        (supplied by HGS pick path).
    View index.
    Transformation M mapping coord
        of slot-1 into coords of
        flat-2.
    Create temporary instantiation
        of slot-1 using a structure
        with transform M and execute
        structure.
    Change writing mode to complement.

To drag, we repeat the following:
    Redraw structure in current
        position (erase).
    Get locator's screen position L.
    Using T, map L into a point P
        on flat-2.
    Calculate translation of P
        from previous P, and apply
        to matrix M.
    Replace old matrix M with new
        matrix M.
    Redraw temporary structure.
```

**Figure 9:** Dragging in HGS

Figure 9 describes how we do the dragging on a device that does not have hierarchical graphics in hardware.

### Rubber banding

Although PHIGS implementations may support a standard 'rubber band' feature as a part of the locator function, we preferred to handle this ourselves so that the 'inner loop' passes through application code and the programmer determines the subtle detail of human interface.

As an example of this, consider how we might implement the dragging of the slot in Figure 2. At each pass through the loop, the application could check if the slot still lay within the boundary of flat-2. If this proved to be demanding on performance, then the check might be performed only when the user hesitated in the dragging action.

## HGS IMPLEMENTATION

The Hierarchical Graphics System (HGS) is a PHIGS-like system implemented by CADM Advanced Development. We are using it to prototype CAD tools for special applications (internal to Digital). We expect PHIGS may be used for future production versions of these tools.

HGS omits the following PHIGS capabilities which were less important from the point of view of a mechanical CAD evaluation: archiving, metafile, text output primitives, input widgets such as valuator and choice. HGS is also not device independent but is specific to the MicroVAX workstation and is layered on the MicroVMS workstation graphics software (also known as UIS), see [8].

HGS provides some extra control over dynamics that is not a part of the PHIGS standard. We expect that by the time the CAD tools we are developing come into production use, workstations will be available that will support full dynamics with PHIGS. In the meantime, we retain device specific control of the VAXstation II/GPX to implement smooth dynamics.

We have HGS bindings for FORTRAN, C, Pascal and Lisp, but the Lisp binding is a special case. We discuss here aspects that particularly concern the Lisp programmer.

- External calls
- Inline compilation
- On-line documentation
- Useful macros

### External Calls

Lisp contains a routine that interfaces Lisp functions to precompiled routines in shareable libraries. This routine **define-external-routine** takes care of mapping the shareable library and can set up calls to routines that conform to the VAX Procedure Calling Standard.

```
(defmacro with-open-structure ((struct init) &body body)
"WITH-OPEN-STRUCTURE (VAR INIT) BODY
   This function opens a structure for editing.
   If the structure does not already exist then a new
empty structure is created.
   BODY consists of one or more forms which might be used
   to make calls to PHIGS for instance to put elements into
   the open structure, however any Lisp forms may appear.
   These forms may use the symbol VAR. Typically VAR would
   be the symbol to be used to reference the id of the
   structure that has just been opened. VAR gets INIT as
   its initial value."
  '(let ((,struct (or ,init (hgs:alloc-struct-id))))
     (unwind-protect
 (progn (hgs:open-structure ,struct) . ,body)
       (hgs:close-structure))))
```

**Figure 10:** Part of HGS.LSP file

### Inline compilation

In our Lisp binding to HGS, we never have the user call the external routine by name:

```
(CALL-OUT HGS$COMPOSE_MATRIX_3 A B C)
```

Rather we always define a Lisp function that in turn calls the graphic function. This introduces another level of call only when the code is interpreted e.g. during development. The extra level is stripped out when the code is compiled with the proclamation of INLINE ensuring that the 'extra' call is removed.

### On-line documentation

We have made our HGS binding to Lisp rather more elaborate than our Pascal, C or FORTRAN bindings. In particular the on-line documentation is worth the effort.

With VAX Common Lisp, the **describe** function is available through the mouse by clicking on a symbol. The documentation string for that function is then displayed. In the case of **with-open-structure**, see Figure 10, this documentation string is the whole section enclosed in double quotes.

### Useful macros

We have implemented some macros that assist the Lisp programmer, for example: **with-open-workstation** and **with-open-structure**. These are based on the style of **with-open-stream**, see Steele [6].

The code for **with-open-structure** is shown in Figure 10 and an example of its use is shown in Figure 11. When debugging the programmer may invoke a lengthy activity with immediate graphics output. Use of Control C to get out is convenient with **with-open-structure** cleaning up before returning to the Lisp prompt.

```
;;; CREATE A NEW STRUCTURE
(defvar *my-structure*
    (alloc-struct-id))
(with-open-structure
    (s *my-structure*)
    (hgs:set-view-index *view-1*)
    (add-names-to-set *incl-set*)
    (store-object-with-pick-id
    small-xy-square)
    (list-structure s)
    (hgs:post-root *ws* s))

;;; Now see if that was ok.
(redraw-all-structures *ws*)

;;; It is worth writing a function to
;;; print the contents of structures.
(list-structure *my-structure*)

;;; This is very useful interactively
(delete-structure *my-structure*)
;;; Now we can try to again.
```

**Figure 11:** Use of with-open-structure

## SUMMARY

We find PHIGS offers suitable functionality for mechanical CAD. In order to arrive at this conclusion we implemented the Hierarchical Graphics System, a subset of PHIGS.

We are using Lisp to prototype CAD tools and have implemented a binding from Lisp to HGS. We feel that the Lisp binding for PHIGS can usefully serve a greater role than simply supporting external subroutine calls.

We look forward to using a full implementation of PHIGS. We expect PHIGS will come into its own over the next few years as workstation hardware with high performance and low cost occupy the desks of more and more engineers.

### References

[1] ANSI PHIGS X3.144-198x Functional Description.

[2] Heck, Michael, *Understanding PHIGS.* Megatek Corporation 1985.

[3] Plaeln, Martin. PHIGS: Programmers Hierarchical Interactive Graphics Standard. *Byte.* Nov 1987.

[4] Abi-Ezzi, Salim S., and Steven E. Kader. 'Phigs in CAD' *Computers in Mechanical Engineering.* July 1986.

[5] Wilson, Peter R. Information and/or Data? *IEEE Computer Graphics and Applications.* Nov 1987.

[6] Stefik. Mark, and Daniel. G. Bobrow, Object-Oriented Programming Themes and Variations. *The AI Magazine.*

[7] Steele. Guy L, *Common Lisp - The Language.* Digital Press 1984.

[8] *MicroVMS*
*Workstation Graphics Programming Guide.* Digital Equipment Corporation.

[9] Shuey. David, PHIGS: a graphic platform for CAD application development. *Computer-Aided Design* October 1987.

The following are trademarks of Digital Equipment Corporation: MicroVAX, VAX, MicroVMS, VAXstation.

# An Overview of the Common Lisp Object System

**Richard P. Gabriel** and **Linda G. DeMichiel**
Lucid, Inc.
Menlo Park, California

### Abstract

The Common Lisp Object System is an object-oriented system that is based on the concepts of generic functions, multiple inheritance, and method combination. All objects in the Object System are instances of classes that form an extension to the Common Lisp type system. The Common Lisp Object System is based on a meta-object protocol that renders it possible to alter the fundamental structure of the Object System itself. The Common Lisp Object System has been proposed as a standard for ANSI Common Lisp and has been tentatively endorsed by X3J13.

## History of the Common Lisp Object System

The Common Lisp Object System is an object-oriented programming paradigm designed for Common Lisp. The lack of a standardized object-oriented extension for Common Lisp has long been regarded as a shortcoming by the Common Lisp community. Two separate and independent groups began work on an object-oriented extension to Common Lisp several years ago. One group is Symbolics, Inc. with New Flavors, and the other is Xerox PARC with CommonLoops. During the summer of 1986, these two groups met to explore combining their designs for submission to X3J13, a technical working group charged with producing an ANSI standard for Common Lisp.

At the time of the exploratory meetings between Symbolics and Xerox, the authors of this paper became involved in the technical design work. The major participants in this effort were David Moon and Sonya Keene from Symbolics, Daniel Bobrow and Gregor Kiczales from Xerox, and Richard Gabriel and Linda DeMichiel from Lucid.

By March 1987 this three-way collaborative effort had produced a strong draft of a specification for the bulk of the Object System. X3J13 has voted an endorsement of that specification draft, stating that it would almost certainly be adopted as part of the standard and encouraging implementors to proceed with trial implementations. This paper is a report on the specification that was presented to X3J13.

## The Common Lisp Object System View of Object-Oriented Programming

Several aspects of the Object System stand out upon inspection: a) it is a layered system designed for flexibility; b) it is based on the concept of generic functions rather than on message-passing; c) it is a multiple inheritance system; d) it provides a powerful method combination facility; e) the primary entities of the system are all first-class objects.

## The Layered Approach

One of the design goals of the Object System is to provide a set of layers that separate different programming language concerns from one another.

The first level of the Object System provides a programmatic interface to object-oriented programming. This level is designed to meet the needs of most serious users and to provide a syntax that is crisp and understandable. The second level provides a functional interface into the heart of the Object System. This level is intended for the programmer who is writing very complex software or a programming environment. The first level is written in terms of this second level. The third level provides the tools for the programmer who is writing his own object-oriented language. It allows access to the primitive objects and operators of the Object System. It is this level on which the implementation of the Object System itself is based.

The layered design of the Object System is founded on the *meta-object protocol*, a protocol that is used to define the characteristics of an object-oriented system. By using the meta-object protocol, other functional or programmatic interfaces to the Object System, as well as other object systems, can be written.

### The Generic Function Approach

The Common Lisp Object System is based on *generic functions* rather than on message-passing. This choice is made for two reasons: 1) there are some problems with message-passing in operations of more than one argument; 2) the concept of generic functions is a generalization of the concept of ordinary Lisp functions.

A key concept in object-oriented systems is that given an operation and a tuple of objects on which to apply the operation, the code that is most appropriate to perform the operation is selected based on the classes of the objects.

In most message-passing systems, operations are essentially properties of classes, and this selection is made by packaging a message that specifies the operation and the objects to which it applies and sending that message to a suitable object. That object then takes responsibility for selecting the appropriate piece of code. These pieces of code are called *methods.*

With unary operations, the choice of a suitable object is clear. With multiary operations, however, message-passing systems run into problems. There are three general approaches to the problem of selecting a suitable object to which to send a message: currying, delegation, and distribution of methods.

Currying is a technique for turning a multiary operation into series of unary operations. In message-passing systems, this is accomplished by having the objects in question send messages among themselves to gather the contributions of each object to the final result of the operation. For example, adding a sequence of numbers can be done by asking each number to add itself to some accumulated total and then to send a message to the next object for it to do the same. Every object must know how to start and end this process. Currying may result in a complicated message-passing structure.

Delegation is a technique whereby an object is defined to handle an operation on a number of other objects. For example, to sum a sequence of numbers, there can be an object that will accept a message containing the identities of the numbers and then perform the addition. Thus, every object that can be involved in a multiary operations must know how to further delegate operations.

Distribution of methods is a technique whereby every object that can be involved in a multiary operation can be outfitted with enough information to directly carry out the operation.

In the generic function approach, objects and functions are autonomous entities, and neither is a property of the other. Generic functions decouple objects and operations upon objects; they serve to separate operations and classes. In the case of multiary operations, the operation is a generic function, and a method is defined that performs the operation on the objects (and on objects that are instances of the same classes as those objects).

Generic functions provide not only a more elegant solution to the problem of multiary operations but also a clean generalization of the concept of functions in Common Lisp. Each of the methods of the generic function provides a definition of how to perform an operation on arguments that are instances of particular classes or of subclasses of those classes. The generic function packages those methods and selects the right method or methods to invoke.

Furthermore, in Common Lisp, arithmetic operators are already generic in a certain sense. The expression

```
(+ x y)
```

does not imply that **x** and **y** are of any particular type, nor does it imply that they are of the same type. For example, **x** might be an integer and **y** a complex number, and the **+** operation is required to perform the correct coercions and produce an appropriate result.

Because the Object System is a system for Common Lisp, it is important that generic functions be first-class objects and that the concept of generic functions be an extension and a generalization of the concept of Common Lisp functions. In this way, the Object System is a natural and smooth extension of Common Lisp.

A further problem with message-passing systems is that there must be some way within a method of naming the object to which a message was sent. Usually there is a pseudovariable named something like "self." With generic functions, parameters are named exactly the same way that Common Lisp parameters are named—there is conceptual simplicity in using the fewest constructs in a programming language.

Some message-passing systems use a functional notation in which there is a privileged argument position that contains the object to which a message is to be sent. For example

```
(display x window-1)
```

might send a message to **x** requesting that **x** display itself on the window **window-1**. This expression might be paraphrased as

```
(send 'display x window-1)
```

With generic functions there are no such privileged argument positions.

In addition to these advantages over message-passing, there are three other fundamental strengths of generic functions:

- It is possible to abstract the definition of a generic function into parts that are conceptually independent. This is accomplished by splitting the definition of a generic function into separate parts where each part is the partial function definition for a particular set of classes. This leads to a new modularization technique.

- It is possible to spread the definition of a generic function among the places where the partial definitions make the most sense. This is accomplished by placing the appropriate **defmethod** forms where the relevant classes are defined.

- It is possible to separate the inheritance of behavior from the replacement of code. Generic functions select methods based on the structure of the class graph, but the generic functions are not constrained to be stored within that graph.

In a message-passing system, the class graph and the instances are truly central because the methods are associated with a particular class or instance. It makes sense in this setting to think of the methods as part of the structure of a class or an instance.

In a generic-function system, the generic functions provide a very different view of methods. Generic functions become the focus of abstraction; they are rarely associated unambiguously with a single class or instance; they sit above a substrate of the class graph, and the class graph provides control information for the generic functions.

Despite the advantages of generic functions, there may at times be reasons for preferring a message-passing style; such a style can be implemented by means of generic functions or by means of the Common Lisp Object System Meta-Object Protocol.

### The Multiple Inheritance Approach

Another key concept in object-oriented programming is the definition of structure and behavior on the basis of the *class* of an object. Classes thus impose a type system—the code that is used to execute operations on objects depends on the classes of the objects. The subclass mechanism allows classes to be defined that share the structure and the behavior of other classes. This subclassing is a tool for modularization of programs.

The Common Lisp Object System is a multiple-inheritance system, that is, it allows a class to directly inherit the structure and behavior of two or more otherwise unrelated classes. In a single inheritance system, if class $C_3$ inherits from classes $C_1$ and $C_2$, then either $C_1$ is a subclass of $C_2$ or $C_2$ is a subclass of $C_1$; in a multiple inheritance system, if $C_3$ inherits from $C_1$ and $C_2$, then $C_1$ and $C_2$ might be unrelated.

If no structure is duplicated and no operations are multiply-defined in the several superclasses of a class, multiple inheritance is straightforward. If a class inherits two different operation definitions or structure definitions, it is necessary to provide some means of selecting which ones to use or how to combine them. The Object System uses a linearized *class precedence list* for determining how structure and behavior are inherited among classes.

### The Method Combination Approach

The Common Lisp Object System supports a mechanism for method combination that is both more powerful than that provided by CommonLoops and simpler than that provided by Flavors.

Method combination is used to define how the methods that are applicable to a set of arguments can be combined to provide the values of a generic function. In many object-oriented systems, the most specific applicable method is invoked, and that method may invoke other, less specific methods. When this happens there is often a combination strategy at work, but that strategy is distributed throughout the methods as local control structure. Method combination brings the notion of a combination strategy to the surface and provides a mechanism for expressing that strategy.

A simple example of method combination is the common need to surround the activities of a method's invocation with a prologue and an epilogue: the prologue might cache some values in an accessible place for the method, and the epilogue will write back the changed values.

The Object System provides a default method combination type, *standard method combination*, that is designed to be simple, convenient, and powerful for most applications. Other types of method combination can easily be defined by using the **define-method-combination** macro.

### First-Class Objects

In the Common Lisp Object System, generic functions and classes are first-class objects with no intrinsic names. It is possible and useful to create and manipulate anonymous generic functions and classes.

The concept of "first-class" is important in Lisp-like languages. A first-class object is one that can be explicitly made and manipulated; it can be stored anywhere that can hold general objects.

Generic functions are first-class objects in the Object System. They can be used in the same ways that ordinary functions can be used in Common Lisp. A generic function is a true function that can be passed as an argument, used as the first argument to **funcall** and **apply**, and stored in the function cell of a symbol. Ordinary functions and generic functions are called with identical syntax.

### What the Common Lisp Object System Is Not

The Object System does not attempt to solve problems of encapsulation or protection. The inherited structure of a class depends on the names of internal parts of the classes from which it inherits. The Object System does not support subtractive inheritance. Within Common Lisp there is a primitive module system that can be used to help create separate internal namespaces.

## Classes

A *class* is an object that determines the structure and behavior of a set of other objects, which are called its *instances*. It is not necessary for a class to have any instances, but all objects are instances of some class. The class system defined by the Object System and the Common Lisp type system are tightly integrated, so that one effect of the Object System is to define a first-class type

system within Common Lisp. The class of an object determines the set of operations that can be performed on that object.

There are two fundamental sorts of relationships involving objects and classes: the subclass relationship and the instance relationship.

A class can inherit structure and behavior from other classes. A class whose definition refers to other classes for the purpose of inheriting from them is said to be a *subclass* of each of those classes. The classes that are designated for purposes of inheritance are said to be *superclasses* of the inheriting class. The inheritance relationship is transitive.

A typical situation is that one class, $C_1$, represents a possibly infinite set of objects, and a second class, $C_2$, represents a subset of that set; in this case $C_2$ is a subclass of $C_1$.

An object is an instance of a class if it is an example of a member of that class. If the class represents a set, then an instance is a member of that set.

Classes are organized into a *directed acyclic graph.* There is a distinguished class named **t**. The class **t** is a superclass of every other class.

Classes themselves are objects and are therefore instances of classes. The class of a class is called a **metaclass**. The existence of metaclasses indicates that the structure and behavior of the class system itself is controlled by classes. Generic functions are also objects and therefore also instances of classes.

The Object System maps the Common Lisp type space into the space of classes. Many but not all of the predefined Common Lisp type specifiers have a class associated with them that has the same name as the type. For example, an array is of type **array** and of class **array**. Every class has a corresponding type with the same name as the class.

A class that corresponds to a predefined Common Lisp type is called a *standard type class*. Each standard type class has the class **standard-type-class** as a metaclass. Users can write methods that discriminate on any primitive Common Lisp type that has a corresponding class. However, it is not allowed to make an instance of a standard type class with **make-instance** or to include a standard type class as a superclass of a class.

All programmer-defined classes are instances of the class named **standard-class**, which is an instance of the class named **class**; the class named **class** is an instance of itself, which is the fundamental circularity in the Object System.

Instances whose metaclass is **standard-class** are like Common Lisp structures: they have named slots, which contain values. When we say that the structure of an instance is determined by its class and that that class is an instance of **standard-class**, we mean that the number and names of the slots are determined by the class, and we also mean that the means of accessing and altering the contents of those slots are controlled by the class.

**Defining Classes**

The macro **defclass** is used to define a new class.

The definition of a class consists of the following: its name, a list of its direct superclasses, a set of slot specifiers, and a set of class options.

The direct superclasses of a class are those classes from which the new class inherits structure and behavior. When a class is defined, the order in which its direct superclasses are mentioned in the **defclass** form defines a *local precedence order* on the class and those superclasses. The local precedence order is represented as a list consisting of the class followed by its direct superclasses in the order mentioned in the **defclass** form.

A slot specifier includes the name of the slot and zero or more slot options that pertain to that particular slot. The name of a slot is a symbol that could be used as a Common Lisp variable name. The slot options of the **defclass** form allow for the following: providing a default initial value form for the slot; requesting that methods for appropriately named generic functions be automatically generated for reading or writing the slot; controlling whether one copy of a given slot is shared by all instances or whether each instance is to have its own copy of that slot; and specifying the type of the slot contents.

There are two kinds of slots: slots that are local to an individual instance and slots that are shared by all instances of the given class. The **:allocation** slot option to **defclass** controls the kind of slot that is defined.

In general, slots are inherited by subclasses. That is, a slot defined by a class is also a slot implicitly defined by any subclass of that class unless the subclass explicitly shadows the slot definition. A class can also shadow some of the slot options declared in the **defclass** form of one of its superclasses by providing its own description for that slot.

Slots can be accessed in two ways: by use of generic functions defined by the **defclass** form and by use of the primitive function **slot-value**.

The syntax of **defclass** provides several means for generating methods to read and write slots. Methods can be requested for all slots or for particular slots only. Methods can be requested to read and write slots or to read slots only. If a slot *accessor* is requested, a method is automatically generated for reading the value of the slot, and a **setf** method is also generated to write the value of the slot. If a slot *reader* is requested, a method is automatically generated for reading the value of the slot, but no **setf** method for it is generated. Readers and accessors can be requested for individual slots or for all slots. Reader and accessor methods are added to the appropriate generic functions. It is possible to modify the behavior of these generic functions by writing methods for them.

The function **slot-value** can be used with any of the slot names specified in the **defclass** form to access a specific slot in an object of the given class. Readers and accessors are implemented by using **slot-value**.

Sometimes it is convenient to access slots from within

the body of a method or a function. The macro **with-slots** is provided for use in setting up a lexical environment in which certain slots are lexically available as variables. It is also possible to specify whether the macro **with-slots** is to use the accessors or the function **slot-value** to access slots.

A class option pertains to the class as a whole. The available class options allow for the following: requesting that methods for appropriately named generic functions be automatically generated for reading or writing all slots defined by the new class; requesting that a constructor function be automatically generated for making instances of the class; and specifying that the instances of the class are to have a metaclass other than the default.

For example, the following two classes define a representation of a point in space. The class **x-y-position** is a subclass of the class **position**:

```
(defclass position () ())

(defclass x-y-position (position)
    ((x :initform 0 :accessor position-x)
     (y :initform 0 :accessor position-y)))
```

The class **position** is useful if we want to create other sorts of representations for spatial positions. The x- and y-coordinates are initialized to 0 in all instances unless explicit values are supplied for them. To refer to the x-coordinate of an instance of the class **x-y-position**, **position**, one writes

```
(position-x position)
```

To alter the x-coordinate of that instance, one writes

```
(setf (position-x position) new-x)
```

The macro **defclass** is part of the Object System programmatic interface and, as such, is on the first of the three levels of the Object System. When applied to an appropriate metaclass, the function **make-instance** provides the same functionality on the second level.

### Class Precedence

Each class has a *class precedence list*. The class precedence list is a total ordering on the set of the given class and its superclasses for purposes of inheritance. The total ordering is expressed as a list ordered from most specific to least specific.

The class precedence list is used in several ways. In general, more specific classes can *shadow*, or override, features that would otherwise be inherited from less specific classes. The method selection and combination process uses the class precedence list to order methods from most specific to least specific.

The class precedence list is always consistent with the local precedence order of each class in the list. The classes in each local precedence order appear within the class precedence list in the same order. If the local precedence orders are inconsistent with each other, no class precedence list can be constructed, and an error will be signaled.

## Generic Functions

The class-specific operations of the Common Lisp Object System are provided by generic functions and methods.

A *generic function* is a function whose behavior depends on the classes or identities of the arguments supplied to it. The *methods* associated with the generic function define the class-specific operations of the generic function.

Like an ordinary Lisp function, a generic function takes arguments, performs a series of operations, and returns values. An ordinary function has a single body of code that is always executed when the function is called. A generic function is able to perform different series of operations and to combine the results of the operations in different ways, depending on the class or identity of one or more of its arguments.

The operations of a generic function are defined by its methods. Thus, generic functions are objects that can be *specialized* by the definition of methods to provide class-specific operations. The behavior of the generic function results from which methods are selected for execution, the order in which the selected methods are called, and how their values are combined to produce the value or values of the generic function.

Thus, unlike an ordinary function, a generic function has a distributed definition corresponding to the definition of its methods. The definition of a generic function is found in a set of **defmethod** forms, possibly along with a **defgeneric-options** form that provides information about the properties of the generic function as a whole. Evaluating these forms produces a generic function object.

In addition to a set of methods, a generic function object comprises a lambda-list, a method combination type, and other information. In Common Lisp a lambda-list is a specification of the parameters that will be passed to a function. The syntax of Common Lisp lambda-lists is complex, and the Object System extends it further.

The lambda-list specifies the arguments to the generic function. It is an ordinary function lambda-list with these exceptions: no **&aux** variables are allowed and optional and keyword arguments may not have default initial value forms nor use supplied-p parameters. The generic function passes to its methods all the argument values passed to it, and only these; default values are not supported.

The method combination type determines the form of method combination that is used with the generic function. The *method combination* facility controls the selection of methods, the order in which they are run, and the values that are returned by the generic function. The Object System offers a default method combination type that is appropriate for most user programs. The Object System

also provides a facility for declaring new types of method combination for programs that require them.

The generic function object also contains information about the argument precedence order (the order in which arguments to the generic function are tested for specificity when selecting executable methods), the class of the generic function, and the class of the methods of the generic function. While the Object System provides default classes for all generic function, method, and class objects, the programmer may choose to implement any or all of these by using classes of his own definition.

Generic functions in the Object System are nearly indistinguishable from ordinary functions: they can be applied, stored, and manipulated exactly as ordinary functions are. In this way, the Object System is smoothly integrated into the Common Lisp framework, and there is no sense in which Common Lisp programs are partitionable into the functional parts and the object-oriented parts.

### Defining Generic Functions

Generic functions are defined by means of the **defgeneric-options** and **defmethod** macros.

The **defgeneric-options** macro is designed to allow for the specification of properties that pertain to the generic function as a whole, and not just to individual methods.

If a **defgeneric-options** form is evaluated and a generic function of the given name does not already exist, a new generic function object is created. This generic function object is a generic function with no methods. The **defgeneric-options** macro may be used to specify properties of the generic function as a whole—this is sometimes referred to as the "contract" of the generic function. These properties include the following: the lambda-list of the generic function; a specification of the order in which the required arguments in a call to the generic function are to be tested for specificity when selecting a particular method; declarations that pertain to the generic function as a whole; the class of the generic function; the class of all the methods of the generic function; and the method combination type to be used with this generic function. The Object System provides a set of default values for these properties, so that use of the **defgeneric-options** macro is not essential.

When a new **defgeneric-options** form is evaluated and a generic function of the given name already exists, the existing generic function object is modified. This does not modify any of the methods associated with the generic function.

The **defmethod** form is used to define a method. If there is no generic function of the given name, however, it automatically creates a generic function with default values for the argument precedence order (left-to-right, as defined by the lambda-list), the generic function class (the class **standard-generic-function**), the method class (the class **standard-method**), and the method combination type (standard method combination). The lambda-list of the generic function is congruent with the lambda-list of the new method. In general, two lambda-lists are congruent if they have the same number of required parameters, the same number of optional parameters, and the same treatment of **&allow-other-keys**.

When a **defmethod** form is evaluated and a generic function of the given name already exists, the existing generic function object is modified to contain the new method. The lambda-list of the new method must be congruent with the lambda-list of the generic function.

## Methods

The class-specific operations provided by generic functions are themselves defined and implemented by *methods.* The class or identity of each argument to the generic function indicates which method or methods are eligible to be invoked.

A method object contains a *method function*, an ordered set of *parameter specializers* that specify when the given method is applicable, and an ordered set of *qualifiers* that are used by the method combination facility to distinguish among methods.

Each required formal parameter of each method has an associated parameter specializer, and the method is expected to be invoked only on arguments that satisfy its parameter specializers. A parameter specializer is either a class or a list of the form **(quote** *object***)**.

A method can be selected for a set of arguments when each required argument satisfies its corresponding parameter specializer. An argument satisfies a parameter specializer if either of the following conditions holds:

- The parameter specializer is a class, and the argument is an instance of that class or an instance of any subclass of that class.
- The parameter specializer is **(quote** *object***)** and the argument is **eql** to *object.*

A method all of whose parameter specializers are **t** is a *default method*; it is always part of the generic function but often shadowed by a more specific method.

Method qualifiers give the method combination procedure a further means of distinguishing between methods. A method that has one or more qualifiers is called a *qualified* method. A method with no qualifiers is called an *unqualified method.*

In standard method combination, unqualified methods are also termed *primary* methods, and qualified methods have a single qualifier that is either **:around**, **:before**, or **:after**.

### Defining Methods

The macro **defmethod** is used to create a method object. A **defmethod** form contains the code that is to be run when the arguments to the generic function cause the method that it defines to be selected. If a **defmethod**

form is evaluated and a method object corresponding to the given generic function name, parameter specializers, and qualifiers already exists, the new definition replaces the old.

Each method definition contains a *specialized lambda-list*, which specifies when that method can be selected. A specialized lambda-list is like an ordinary lambda-list except that a *parameter specifier* may occur instead of the name of a parameter. A parameter specifier is a list consisting of a variable name and a parameter specializer name. Every parameter specializer name is a Common Lisp type specifier, but the only Common Lisp type specifiers that are parameter specializers names are type specifier symbols with corresponding classes and type specifier lists of the form (**quote** *object*). The form (**quote** *object*) is equivalent to the type specifier (**member** *object*).

Only required parameters can be specialized, and each required parameter must be a parameter specifier. For notational simplicity, if some required parameter in a specialized lambda-list is simply a variable name, the corresponding parameter specifier is taken to be (*variable-name* **t**).

A future extension to the Object System might allow optional and keyword parameters to be specialized.

A method definition may optionally specify one or more method qualifiers. A method qualifier is a non-**nil** atom that is used to identify the role of the method to the method combination type used by the generic function of which it is part. By convention, qualifiers are usually keyword symbols.

Generic functions can be used to implement a layer of abstraction on top of a set of classes. For example, the class **x-y-position** can be viewed as containing information in polar coordinates.

Two methods are defined, called **position-rho** and **position-theta**, that calculate the $\rho$ and $\theta$ coordinates given an instance of the class **x-y-position**.

```
(defmethod position-rho ((pos x-y-position))
   (let ((x (position-x pos))
         (y (position-y pos)))
     (sqrt (+ (* x x) (* y y)))))

(defmethod position-theta
           ((pos x-y-position))
  (atan (position-y pos) (position-x pos)))
```

It is also possible to write methods that update the 'virtual slots' **position-rho** and **position-theta**:

```
(defmethod (setf position-rho)
   (rho (pos x-y-position))
  (let* ((r (position-rho pos))
         (ratio (/ rho r)))
   (setf (position-x pos)
         (* ratio (position-x pos)))
   (setf (position-y pos)
         (* ratio (position-y pos)))))

(defmethod (setf position-theta)
   (theta (pos x-y-position))
  (let ((rho (position-rho pos)))
   (setf (position-x pos)
         (* rho (cos theta)))
   (setf (position-y pos)
         (* rho (sin theta)))))
```

To update the $\rho$-coordinate one writes

```
(setf (position-rho pos) new-rho)
```

This is precisely the same syntax that would be used if the positions were explicitly stored as polar coordinates.

### Class Redefinition

The Common Lisp Object System provides a powerful class-redefinition facility.

When a **defclass** form is evaluated and a class with the given name already exists, the existing class is redefined. Redefining a class modifies the existing class object to reflect the new class definition.

When a class is redefined, changes are propagated to instances of it and to instances of any of its subclasses. The updating of an instance whose class has been redefined (or that has a superclass that has been redefined) occurs at an implementation-dependent time; this will usually be upon the next access to that instance or the next time that a generic function is applied to that instance. Updating an instance does not change its identity. The updating process may change the slots of that particular instance, but it does not create a new instance.

A similar task is to change the class of an instance. The generic function **class-changed** is invoked automatically by the system after **change-class** has been used to restructure an instance to conform to the new class. Users may define methods on the generic function **class-changed** to control the change-class process.

For example, suppose it becomes apparent that the application that requires representing positions uses polar coordinates more than it uses rectangular coordinates. It might make sense to define a subclass of **position** that uses polar coordinates:

```
(defclass rho-theta-position (position)
    ((rho :initform 0
       :accessor position-rho)
     (theta :initform 0
       :accessor position-theta)))
```

The instances of **x-y-position** can be automatically updated by defining a **class-changed** method:

```
(defmethod class-changed
  ((old x-y-position)
   (new rho-theta-position))
  ;; Copy the position information
  ;; from old to new to make new
```

```
;; be a rho-theta-position at the
;; same position as old.
(let ((x (position-x old))
      (y (position-y old)))
  (setf (position-rho new)
        (sqrt (+ (* x x) (* y y)))
        (position-theta new)
        (atan y x))))
```

At this point we can change an instance of the class **x-y-position**, **p1**, to be an instance of **rho-theta-position** by using **change-class**:

```
(change-class p1 'rho-theta-position)
```

## Inheritance

Inheritance is the key to program modularity within the Object System. A typical object-oriented program consists of several classes, each of which defines some aspect of behavior. New classes are defined by including the appropriate classes as superclasses, thus gathering desired aspects of behavior into one class.

### Inheritance of Slots and Slot Description

In general, slot descriptions are inherited by subclasses. That is, slots defined by a class are usually slots implicitly defined by any subclass of that class unless the subclass explicitly shadows the slot definition. A class can also shadow some of the slot options declared in the **defclass** form of one of its superclasses by providing its own description for that slot.

In the simplest case, only one class in the class precedence list provides a slot description with a given slot name. If it is a local slot, then each instance of the class and all of its subclasses allocate storage for it. If it is a shared slot, the storage for the slot is allocated by the class that provided the slot description, and the single slot is accessible in instances of that class and all of its subclasses.

More than one class in the class precedence list can provide a slot description with a given slot name. In such cases, at most one slot with a given name is accessible in any instance, and the characteristics of that slot involve some combination of the several slot descriptions.

Methods that access slots know only the name of the slot and the type of the slot's value. Suppose a superclass provides a method that expects to access a shared slot of a given name, and a subclass provides a local description of a local slot with the same name. If the method provided by the superclass is used on an instance of the subclass, the method accesses the local slot.

### Inheritance of Methods

A subclass inherits methods in the sense that any method applicable to an instance of a class is also applicable to instances of any subclass of that class (all other arguments to the method being the same).

The inheritance of methods acts the same way regardless of whether the method was created by using **defmethod** or by using one of the **defclass** options that cause methods to be generated automatically.

## Class Precedence List

The class precedence list is a linearization of the subgraph consisting of a class, $C$, and its superclasses. The **defclass** form for a class provides a total ordering on that class and its direct superclasses. This ordering is called the *local precedence order*. It is an ordered list of the class and its direct superclasses. A class precedes its direct superclasses, and a direct superclass precedes all other direct superclasses specified to its right in the superclasses list of the **defclass** form. For every class in the set of $C$ and its superclasses, we can gather the specific relations of this form into a set, called $R$.

$R$ may or may not generate a partial ordering, depending on whether the relations are consistent; we assume they are consistent and that $R$ generates a partial ordering. This partial ordering is the transitive closure of $R$.

To compute the class precedence list at $C$, we topologically sort $C$ and its superclasses with respect to the partial ordering generated by $R$. When the topological sort algorithm must select a class from a set of two or more classes, none of which is preceded by other classes with respect to $R$, the class selected is chosen deterministically. The rule that was chosen for this selection process is designed to keep chains of superclasses together in the class precedence list. That is, if $C_1$ is the unique superclass of $C_2$, $C_2$ will immediately precede $C_1$ in the class precedence list.

It is required that an implementation of the Object System signal an error if $R$ is inconsistent, that is, if the class precedence list cannot be computed.

## Method Combination

When a generic function is called with particular arguments, it must determine what code to execute. This code is termed the *effective method* for those arguments. The effective method can be one of the methods of the generic function or a combination of several of them.

Choosing the effective method involves the following decisions: which method or methods to call; the order in which to call these methods; which method to call when **call-next-method** is invoked; what value or values to return.

The effective method is determined by the following steps: 1) selecting the set of applicable methods; 2) sorting the applicable methods by precedence order, putting the most specific method first; 3) applying method combination to the sorted list of applicable methods, producing the effective method.

When the effective method has been determined, it is called with the same arguments that were passed to the

generic function. Whatever values it returns are returned as the values of the generic function.

The Object System provides a default method combination type, *standard method combination*. The programmer can define other forms of method combination by using the **define-method-combination** macro.

### Standard Method Combination

Standard method combination is the default method combination type. Standard method combination recognizes four roles for methods, as determined by method qualifiers.

*Primary methods* define the main action of the effective method; *auxiliary methods* modify that action in one of three ways. A primary method has no method qualifiers. The auxiliary methods are **:before**, **:after**, and **:around** methods.

The semantics of standard method combination are given as follows:

If there are any **:around** methods, the most specific **:around** method is called. Inside the body of an **:around** method, **call-next-method** can be used to immediately call the next method. When the next method returns, the **:around** method can execute more code. By convention, **:around** methods almost always use **call-next-method**.

If an **:around** method invokes **call-next-method**, the next most specific **:around** method is called, if one is applicable. If there are no **:around** methods or if **call-next-method** is called by the least specific **:around** method, the other methods are called as follows:

- All the **:before** methods are called, in most specific first order. Their values are ignored.

- The most specific primary method is called. Inside the body of a primary method, **call-next-method** may be used to pass control to the next most specific primary method. When that method returns, the first primary method can execute more code. If **call-next-method** is not used, only the most specific primary method is called.

- All the **:after** methods are called in most specific last order. Their values are ignored.

If no **:around** methods were invoked, the most specific primary method supplies the value or values returned by the generic function. Otherwise, the value or values returned by the most specific primary method are those returned by the invocation of **call-next-method** in the least specific **:around** method.

If only primary methods are used, standard method combination behaves like CommonLoops. If **call-next-method** is not used, only the most specific method is invoked, that is, more general methods are shadowed by more specific ones. If **call-next-method** is used, the effect is the same as **run-super** in CommonLoops.

If **call-next-method** is not used, standard method combination behaves like **:daemon** method combination of New Flavors, with **:around** methods playing the role of whoppers, except that the ability to reverse the order of the primary methods has been removed.

The use of method combination can be illustrated by the following example. Suppose we have a class called **general-window**, which is made up of a bitmap and a set of viewports:

```
(defclass general-window ()
   ((initialized :initform nil
     :accessor general-window-initialized)
    (bitmap :type bitmap
     :accessor general-window-bitmap)
    (viewports :type list
     :accessor general-window-viewports)))
```

The viewports are stored as a list. We presume that it is desirable to make instances of general windows but not to create their bitmaps until they are actually needed. Thus there is a flag, called **initialized**, that states whether the bitmap has been created. The **bitmap** and **viewport** slots are not initialized by default.

We now wish to create an announcement window to be used for messages that must be brought to the user's attention. When a message is to be announced to the user, the announcement window is exposed, the message is moved into the bitmap for the announcement window, and finally the viewports are redisplayed:

```
(defclass announcement-window
 (general-window)
 ((contents :initform "" :type string
   :accessor announcement-window-contents)))

(defmethod display :around
           (message (w general-window))
 (unless (general-window-initialized w)
  (setf (general-window-bitmap w)
        (make-bitmap))
  (setf (general-window-viewports w)
        (list
         (make-viewport
          (general-window-bitmap w))))
  (setf (general-window-initialized w) t))
  (call-next-method))

(defmethod display :before
           (message (w announcement-window))
 (expose-window w))

(defmethod display :after
           (message (w announcement-window))
  (redisplay-viewports w))

(defmethod display
           ((message string)
            (w announcement-window))
 (move-string-to-window message w))
```

To make an announcement, the generic function **display** is invoked on a string and an annoucement window. The **:around** method is always run first; if the bitmap has not been set up, this method takes care of it. The primary method for **display** simply moves the string (the announcement) to the window, the **:before method** exposes the window, and the **:after** method redisplays the viewports. When the window's bitmap is initialized, the sole viewport is made to be the entire bitmap. The order in which these methods are invoked is the following: 1) the **:around** method, 2) the **:before** method, 3) the primary method, and 4) the **:after** method.

### Meta-Object Protocol

The Common Lisp Object System is implemented in terms of a set of objects that correspond to predefined classes of the system. These objects are termed *meta-objects*. Because meta-objects underlie the rest of the object system, they may be used to define other objects, other ways of manipulating objects, and hence other object-oriented systems. The use of meta-objects is specified by in the Common Lisp Object System meta-object protocol.

The meta-object protocol is designed for use by implementors who need to tailor the Object System for particular applications and by researchers who wish to use the Object System as a prototyping tool and delivery environment for other object-oriented paradigms. It is also designed for the implementation of the Object System itself.

The Object System provides a number of predefined meta-objects.

### Evaluation of the Design Goals for the Common Lisp Object System

Even when designers start with a clean slate, it is difficult to achieve the design goals of a language, but the task of satisfying those goals is very much more difficult when two different languages with existing user communities are being merged. The final design of a language—its shape and clarity—depends on the ancestor languages. If the ancestor languages do not have a clean design, the designers must weigh the desirability of a clean design against the impact of changes on their existing user communities.

In this case there were two such ancestor languages, each with a young but growing user community. CommonLoops has a relatively clean design. Its experimental implementation, "Portable CommonLoops," is freely distributed to a small user community. New Flavors is the second generation of a basic system called "Flavors." Flavors is an older, commercially supported message-passing system. New Flavors is a generic-function-based descendant of Flavors, but a descendant that supports compatibility with its ancestor. The design of New Flavors is not as clean as that of CommonLoops, but its user community is larger and is accustomed to commercially motivated stability.

With these facts in mind, let us examine the design goals for the Common Lisp Object System.

- Use a set of levels to separate programming language concerns from each other.

This goal was achieved quite well. The layering is well defined and has been shown to be useful both in terms of implementing the Object System as well as in terms of implementing another object-oriented system (Hewlett-Packard's CommonObjects). CommonLoops already has a layered design.

- Make as many things as possible within the Object System first-class.

This goal was also achieved quite well. Both classes and generic functions are first-class objects. Generic functions were not first-class objects in either CommonLoops or New Flavors, but fortunately the user-community disruption is minimal for these changes.

- Provide a unified language and syntax.

It is tempting when designing a programming language to invent additional languages within the primary programming language, where each additional language is suitable for some particular aspect of the overall language. In Common Lisp, for example, the **format** function defines a language for displaying text, but this language is not Lisp nor is it like Lisp.

There are two additional languages in the Object System: the language of method combination keywords and a pattern language for selecting methods, which is used when defining new method combination types. Method combination was considered important enough to tolerate these additional languages.

CommonLoops has no additional languages. New Flavors has a rich method combination facility with more complex versions of the two additional languages just mentioned; these two languages were simplified for inclusion in the Object System.

- Be willing to trade off complex behavior for conceptual and expository simplicity.

This was largely achieved, although it required a considerable amount of additional technical work. For example, the class precedence list algorithm was altered from its original form largely because the original was not easily explainable.

- Make the specification of the language as precise as possible.

The specification of Common Lisp is fairly ambiguous because it represents a technical compromise between several existing Lisp dialects. The ground rules of the compromise were explicitly established so that with a small

effort each of the existing Lisp dialects could be made into a Common Lisp.

The Object System design group was in a similar position to the Common Lisp design group, but on a smaller scale. The language of the specification is considerably more precise than the language of the Common Lisp specification.

## Conclusion

The Common Lisp Object System is an object-oriented paradigm with several novel features; these features allow it to be smoothly integrated into Common Lisp.

## References


[DGB1] Daniel G. Bobrow, Linda G. DeMichiel, Richard P. Gabriel, Sonya Keene, Gregor Kiczales, and David A. Moon, *Common Lisp Object System Specification*, X3J13 Document 87-002.

[DGB2] Daniel G. Bobrow, Kenneth Kahn, Gregor Kiczales, Larry Masinter,Mark Stefik, and Frank Zdybel, "CommonLoops: Merging Lisp and Object-Oriented Programming," ACM OOPSLA Conference, 1986.

[AG] Adelle Goldberg and David Robson, *Smalltalk-80: The Language and its Implementation*, Addison-Wesley, Reading, Massachusetts, 1983.

[GLS] Guy L. Steele Jr., *Common Lisp: The Language*, Digital Press, 1984.

[ES] *Reference Guide to Symbolics Common Lisp: Language Concepts*, Symbolics Release 7 Document Set, 1986.

# BUSINESS APPLICATIONS SIG

# POSTAGE & MAILING COST SAVING COMPUTER PROGRAMS

Richard L. Fleischer

RICHARD L. FLEISCHER & ASSOCIATES, INC.
Roslyn Heights, New York

**Abstract**

The U.S. POSTAL SERVICE work sharing discount schemes and acceptance requirements offer volume mailers opportunities to save substantial costs by properly preparing mailings utilizing computer software.

This presentation describes the postal requirements, identifies volume mailers, and suggests their potential return on investment from four computer mailing systems. It describes the functions performed by (A) Carrier Route coding and mailing, (B) ZIP+4 coding and mailing, (C) Five Digit ZIP code correction and insertion, and (D) Merge/ Purge-duplicate elimination programs written in ANSI COBOL 74 and operational on DEC and other medium to large-scale computers.

## I. Postal Regulations and Discounts

The U.S. Postal Service offers discounts to volume mailers that presort 1st, 2nd and 3rd class for profit and non-profit mail. A sample of this rate scheme is shown herein.

The Post Office offers discounts of 1.8c per piece of qualifying Carrier Route Coded 3rd class mail and 1.0 for 1st class letters and postcards (in addition to the ZIP Presort discounts of 2.4c and 4.0c respectively).

The Post Office offers discounts of 0.5c and .9c each on machine-readable First Class mail bearing a 9-digit ZIP Code (ZIP+4).

Carrier Route coded mail moves through the Post Office at least one day faster than uncoded mail. In peak season it may save three days or more.

On all three classes of mail a discount is offered if a specific number of pieces of mail all having the same 5-digit ZIP code are presorted and bundled together. If the mailer further identified, marks and bundles pieces being delivered by the same letter carrier (known as Carrier Route code) an additional discount is offered. This requires 10 pieces in a carrier route in 1st and 3rd class mail and 6 pieces in 2nd class mail. On first class mail in addition a discount is offered for mail if it contains the ZIP+4 code.

For all classes of mail the Postal Services specifies how many pieces of mail are required in a tray, bag or pallet and how these must be labeled.

For all classes of mail the Postal Services specifies how many pieces of mail are required in a tray, bag or pallet and how these must be labeled.

2nd and 3rd class mail without 5 digit ZIP codes or with incorrect ZIP codes are not delivered and 1st class mail so addressed is frequently delayed. Computer programs can easily correct wrong five digit ZIP codes, assign ZIP+4 and Carrier Route codes and perform the mail qualification and preparation functions.

## II. Volume Mailers and their Potential Return on Investment

For purposes of this analysis 'volume mailers' are described as those organizations mailing two million or more pieces of mail per year. Such mailers can normally obtain a return on investment on the installation of Carrier Route

| | 1st Class | | 2nd Class | 3rd Class | |
|---|---|---|---|---|---|
| | 5-Digit | ZIP+4 | per piece | Regular | Nonprofit |
| Basic Rate | 22.0c | 21.2c | 12.3c | 12.5c | 6.0c |
| ZIP Presort | 18.0c | 17.5c | 9.6c | 10.1c | 4.9c |
| Carrier Presort | 17.0c | 17.0c | 7.8c | 8.3c | 3.4c |

coding or ZIP+4 coding software in less than one year. The larger second and third class carrier route discount results in more dollar savings than in 1st class. Obviously the larger and more dense mailings offer greater potential benefits than smaller more geographically spread mailings.

For example, an organization that mails two million pieces of third class mail per year but does not presently carrier route code its mailings could save $28,800 of postage per year, if it achieved 80% carrier qualification.

Generally, organizations that can benefit most mail to the consumer rather than to other businesses and have either a main frame computer or a large mini. Industries included are: Direct Mail Marketers, Department Stores, Specialty Stores, Commercial Banks, Savings Banks, Insurance Companies, State, Local and Federal Agencies, Associations, Charities, Religious Organizations, Fraternal Groups, Labor Unions, Credit Bureaus, Letter Shops, Direct Mail Computer Service Bureaus, Credit Card Companies, Utilities, Printers, Publishers, Magazines and other large mailers.

Organizations selling products and services or soliciting funds normally mail to individuals not on their own list. Typically they rent multiple lists from other organizations whose customers or members hopefully will be interested in responding to the organization's offering. Since many individuals' names and addresses are frequently on more than one list or even duplicated in slightly different form on the same list, the prospective recipients' names and addresses are run through a merge/purge duplicate elimination system so that a given individual will receive only one piece.

Typically this process could reduce the number of pieces mailed by as much as 20 percent. This results in elimination of unnecessary paper printing, assembling, postage and mail preparation costs.

Also direct marketers need to identify good versus poor prospects to improve the response rates achieved. Four tools assist the mailer to achieve this goal.

a Merge/purge - duplicate elimination packages identify and remove duplicate names from both internal and rented lists.

b ZIP code correction systems prevent many undeliverable addresses (because of bad ZIP codes) from being mailed.

c Census encoding system allows for affixing demographic data to lists.

d Profiling, targeting and measuring systems identify customers with buying histories that would be probably respondents and then measure the response by sub group.

When a prospect responds to a promotion it is critical to efficiently and promptly provide the product or service requested. This function is fulfilled by mail and phone order processing systems.

Some organizations purchase one or more mail related computer services from Direct Mail Service bureaus, letter shops or printers.

However, the availability or programs written in ANSI COBOL offer the volume mailer with a DEC VAX computer an opportunity to perform such activities in-house frequently (1) improving operational control, (2) reducing lead times, (3) saving costs, (4) enhancing the security of its list (a very valuable asset), and (5) providing greater flexibility in selecting an outside vendor to perform the physical mail preparation function.

## III. Carrier Route Coding and ZIP+4 Coding and Mail Systems

The functioning of a carrier route coding and a ZIP+4 coding system are very similar in nature. The differences are that data file utilized in the first is the Post Office supplied CRIS and in the second is the Post Office supplied ZIP+4 file. From CRIS the carrier route code can be obtained but from ZIP+4 the ZIP+4 code, carrier route code and residence type can be derived. The ZIP+4 file consists of over 22 million records while the CRIS file only contains about 4 million records. In 1987 ZIP+4 discounts apply only to first class mail and as a result most 2nd and 3rd class only mailers find carrier coding more economic.

Some differences exist in default rules in matching to each file and the mail qualification rules vary from class to class. However, the major functions, logic and flow of the two systems are almost identical. Therefore, this presentation will describe only one, the ZIP+4 system.

An overview of the functions performed by ZIP+4 system follows:

### Conditioning and Maintaining ZIP+4 File

1. EDIT
   Provides ability to correct or modify Post Office file of Carrier Route Codes, ZIP+4 Codes and street addresses. Reads the Post Office file and applies internal logic to correct and reorganize records on the Post Office file. Reads 131 character records and reduces these to 110 characters. Another option allows compacting to 35 characters.

2. SORT
   Provides ability to resequence Post Office tape. Sorts the corrected and edited Post Office file into sequence for matching with the user's name and address file. The sort sequence from high to low order is as follows: ZIP Code, street name, street suffix, street direction, primary house number range.

3. PRINT
   Prints Post Office File. Prints the Post Office tape or ranges of ZIP Codes the user selects. The printout is a valuable tool for diagnosing and correcting conditions preventing a proper matching of the user's addresses with the Post Office file.

   This program can generate a tape copy of only those ZIP Code ranges specified by the user. Processes the ZIP+4 tape provided by the Post Office containing 131 character records.

4. UPDATE
   Applies manual addition or deletion of records to the reduced size (110 characters) and standardized Post Office file. Reads the edited file and performs the same selective printing and tape copy functions as described above. Also it allows the user to (A) delete bad records; (B) create additional records in which the data located in the secondary name field has been transferred to the primary name field; and (C) to generate records with changed street names and other data.

   After initial match has taken place and uncodable addresses have been identified, can be used to extract selected ZIP Codes and correct misspellings and other errors on the Post Office file so that more addresses can be coded.

**Editing Mailer's File**

5. EDITS AND PARSE ADDRESS
   Reads user name and address file (and/or change file) and establishes a match address field which conforms to the requirements and conventions prescribed by the Post Office. Expands abbreviations, identifies the separate components of the address (street name, suffix, direction and house number, apartment number) and properly organizes this information. Post Office box and rural routes receive special handling.

   The program contains a series of tables of valid directions, special street name contractions, valid suffixed and alternate abbreviations for them and of frequently used abbreviations appearing in street names. Additions or deletions to these tables can be made to customize the program to best handle special local conditions. Other special rules have been inserted to solve significant problems relating to some street names. The output file from this program contains all the information on the user input file plus a match field of 42 characters used by a subsequent program. It designates additional fields for it to place the Carrier Route Code and ZIP+4 add-on digits if locations were not specified in the user's input field.

6. SORTS FILE
   Sorts the user file into the same sequence as the sorted Post office file.

**Matching Edited Mailer's File with Edited Post Office file**

7. CODES FILE
   Matches the Post Office ZIP+4 file with the reorganized user name and address file. It assigns ZIP+4 and Carrier Route Codes where a match exists and for those addresses that cannot be looked up automatically prints an error listing designating the reason for the failure. CN030 produces two output tapes (a) all matched addresses containing ZIP+4, Carrier Route Codes and residence codes, and (b) all unmatched addresses. The format and record sizes of both output tapes can be specified by the user. When insufficient information exists on the mailer's file to code an addition to the most accurate code the program applied various default rules approved by the postal service. For example, if no apartment number is present the building or block face code can be applied.

**Mail Qualification and Preparation**

8. SORTS
   Sorts mailing file into Carrier Route Code within ZIP Code (5 digit) sequence.

9. PRINTS CARRIER QUALIFICATION REPORT
   Prepares a control summary report for the Post office and a control file containing counts of the number of qualified and unqualified addresses by Carrier Route Code for each ZIP Code. To prepare the report for the Post Office only the qualified tape should be run. To prepare an internal document for analysis both the qualified and unqualified tapes should be run. This program, the next two steps would only be run if the mailer was doing Carrier Route presorting in addition to ZIP+4 coding.

10. SORTS
    Sorts control file. Resequences the control file placing ZIP code count records in front of Carrier Route Code count records.

11. PRINTS BAG/TRAY LABELS for Carrier Route Code qualified mail.

12. ORGANIZES FILES
    Splits the user file into separate Carrier Route qualified and unqualified address files. The non Carrier Route qualified addresses are run through the user's existing label printing program and codified to print the ZIP+4 code. If Carrier Route Coded qualified mail is not part of the job there will be no qualification file as input and no Carrier Route Code address file out. Produces a ZIP Code qualification/residual report and a control file that is used to (a) print bag/tray labels for ZIP presorted mail.

If comingling is not allowed, performs a critical function of determining which ZIP+4 coded mail should be qualified and which should be commingled with 5 digits coded mail so as to optimize the discount actually obtained. To obtain the presort discount (of 3 cents) the mailer must have 10 or more pieces in a 5-digit ZIP Code or 50 or more in a 3-digit Sectional Center. Therefore this program determines when to forego the ZIP+4 discount (of .5c) in order to achieve the presort discount.

13. SORTS the control file generated in the last step.

14. PRINTS bag/tray labels for ZIP qualified mail.
15. SPLITS the 1st Class mailing into four (4) separate groups based on the postage rate applicable. These are:

    A. ZIP+4 coded presorted mail
    B. ZIP+4 coded residual mail
    C. 5-digit presorted mail
    D. 5-digit residual mail or the residual ZIP+4 and residual 5-digit may be commingled. if the mailing contains a sufficient percent of ZIP+4 coded mail the ZIP+4 and 5-digit presorted may be commingled.

## IV. ZIP Code Correction

ZIP Code Correction and Insertion software utilizes two U.S. Postal Service files, the city-state file and the CRIS file. Both files are edited and sequenced similarly to steps 1 though 4 of the ZIP+4 coding process. The mailer's file is also edited using step five of that process.

In addition, the system performs three major functions, (1) separates addresses located in single ZIP-coded areas from those located in multi-zoned cities and towns, (2) matches the mutli-zoned address with the CRIS file and (3) matches single zoned addresses with the city state file.

1. SEPARATES MULTI AND SINGLE ZONED CITIES
   Identifies and separates addresses with ZIP Codes where the first three digits represent a unique ZIP Coded office. These approximately 300 offices are defined in the Domestic Mail Manual (Exhibit 122,634) and comprise most major cities and highly populated areas such as 100 New York, NY; 108 New Rochelle, NY; 122 Albany, NY; 441 Cleveland, OH; 981 Seattle, WA, etc. Edits city, state field.

2. CORRECTS MULTI-ZONED ADDRESSES
   Edits, sorts these addresses and then matches them against an edited, sorted Post Office CRIS file. The matching program compares the mailer's street address and house number against the corresponding information on the Post Office file and if the ZIP Code does not match, corrects it. The program can assign Carrier Route Codes in the same pass, if desired.

3. CORRECTS SINGLE ZIP CODED ADDRESSES
   Compares city and state of mailers' addresses that are not part of a multi-ZIP coded city or town with a Post Office file of office names and their ZIP Codes. These ZIP Codes are mainly suburban and rural areas, such as 11576 Roslyn, NY or 11577 Roslyn Heights, NY. If the ZIP Code on the mailer's file does not match the ZIP on the Post Office file for the same Post Office name, then the Post Office ZIP Code for that Post Office name is substituted.

## V. Merge/Purge - Duplicate Elimination

The system enables mailers to identify duplicate names and addresses within a file and between multiple files. Also, it can be run to suppress names existing in one file from being selected when occurring on another file.

The system is written in ANSI COBOL and can run in conjunction with the Carrier Route Coding; the Merge/Purge system uses the Carrier Routing System Edit Program to standardize address fields.

The system provides mailers with a wide variety of optional criteria for deciding if a name and address are unique such as (a) address only; (b) address and last name; (c) address, last name and first initial; or (d) address and last name for single family and duplex residents and address, first name and initial for building with four or more apartments.

Merge/Purge-Duplicate Elimination systems consist of (A) edits that standardize the address (step 5 of the ZIP+4 system), (B) edits that manipulate the individual's name and construct a key using some or all of the letters, (C) sort and merge utilities (D) purge and suppress programs that compare the edited name and addresses of successive records to identify records to be eliminated from the mailing file and (E) report programs that provide management with an ability to examine the use made of each list.

1. FORMAT: converts outside lists to the house list file size, record format and blocking factor. if the Merge/Purge question is answered yes, the program generates a six-character name code which is used in the matching logic. Format calls a sub-routine called SOUND which manipulates certair frequently confused letter combinations.

2. SOUND is a subroutine that can be used by FORMAT which utilizes phonics in name standardization.

3. EDIT parses and standardizes the street address.

4. A SORT utility is used to arrange each file to be merge/purged into into the same sequence based on a 52-character key consisting of the standardized name and address and a purge priority code. The use of a multi-tape input SORT-MERGE utility where multiple files are involved is also possible and will reduce the number of merges required where many files are involved.

5. MERGE Utility is used to merge together multiple sorted files.

6. PURGE takes the edited standardized sorted merged file, identifies duplicates based on the parameters entered into the FORMAT and PURGE programs. It creates two output files, one containing unique records and one containing duplicates. Where duplicates are encountered the name and address used for mailing is from the list assigned the highest priority from 00 to 68. The lowest number is the highest priority.

7. SUPPRESS takes in two edited, standardized and sorted files. File A is used to suppress mailing to any duplicate address on file B. The outputs are two files: (a) the B File without any names and addresses contained on the File A and (b) the names and addresses on file B that duplicated names on File A and were, therefore, suppressed.

   The unique names and addresses out of PURGE AND SUPPRESS go into the next step in the mailer's process.

   The Duplicate and File ID files from Purge and Suppress go into one or more of five report programs that print reports:

8. BROKER REPORT — By list report indicates level of dupes with each other list. Shows following data for each other list relative to a list: total records in, invalid addresses, addresses into merge/ purge, total out, matched in merge/ purge, percent matched, percent to total, percent dropped, percent of total, subtotal, intra file dupes, suppression results by file and total.

9. DUPE REPORT — Multi-buyer report by list and in total; identifies number of two-time buyers, three- time buyers and four or more.

10 DROP REPORT — Listing of dropped (duplicate) names and address (sample only).

11. SUMMARY REPORT — Summary report of one line per list input indicates total records in, records dropped, multi-buyers, records retained, percent retained, percent of total.

12. SUPPRESS REPORT — Dropped address report (by list) indicates number of duplicates and number suppressed.

### VI. Other Mailing Applications

The coding systems previously described standardized and code addresses so that mail can qualify for the lowest postage rates available and can move through the postal system with minimal delays. ZIP correction and merge-purge software eliminate undeliverable and duplicate pieces from being mailed thereby doing away with substantial otherwise unproductive costs.

Other opportunities exist to eliminate costs by not mailing to individuals with low propensity to respond to a given promotional mailing. Computer programs that assist in accomplishing this goal perform two functions:

1. Direct marketing management, profiling, targeting and measuring systems create detailed past buying history records that can be used to predict the customer's future actions.

2. Census coding systems by attaching census tract and block codes can allow further profiling of customers and non-customers with demographic data that can help to predict the behavior of non-customers based on their similarity to customers.

A future presentation will discuss these complex applications.

### VII. Summary

Carrier Route coding, ZIP+4 coding, ZIP code correction and insertion and merge/purge-duplicate elimination software can save volume mailers substantial postage and other mailing costs. Organizations doing sufficient mail volume using DEC computers can achieve excellent return on investment by acquiring software written in COBOL so these functions can be performed in-house.

Suggestions for improvements, requests for additional information including upates on changing postal discounts and questions can be directed to

Richard L. Fleischer
Richard L. Fleischer & Associates, Inc.
135 Village Road
Roslyn Heights, New York 11577-1522
(516) 621-2826

# DATA ACQUISITION, ANALYSIS, RESEARCH, AND CONTROL SIG

# DEVELOPING A CIM ARCHITECTURE

Nigel P. Weymont
Jeffrey S. Honeyager

Texas Instruments
Industrial Systems Division
Hunt Valley
Maryland 21030

## ABSTRACT

From an analysis of the requiremnts for Computer Integrated Manufacturing, or CIM, systems, a generic system model has been developed using structured design techniques. Basic elements common to all systems have been identified including business cells, production cells, area controllers, a common data manager and CIM users. The model has been validated using a pharamaceutical packaging line as a practical example. Comments concerning the design and implementation of CIM systems are also included.

## 1.0 INTRODUCTION

Currently, Computer Integrated Manufacturing, or CIM, is widely viewed as the solution to automation. Unfortunately this has led to a great deal of confusion about what constitutes CIM, how it should be developed, implemented and managed. When we were faced with the problem of developing answers to these questions, it became apparent that some sort of CIM model was needed to allow us to understand the functional requirements for CIM systems. It was also recognized that CIM is highly customized for each application and possibly each system is unique. Nevertheless, we wished to analyze CIM systems so that we could identify some basic elements, and hopefully, some elements common to all CIM systems.

A common representation of CIM architecture uses a triangle, shown in Figure 1, which is divided into horizontal slices(1). The lowest level, along the base of the triangle represents the factory floor and successive upward layers represent a work cell, a department, a plant and finally, at the apex, a corporation. However, such an abstraction does not provide a particularly useful model, and its basis as a triangle is unclear. Such models are not useful because they do not represent a fundamental requirement of CIM as we see it, which is the need to move data around an organization in such a way that each part of the organization has the data necessary to contribute to the operation of the organization. Furthermore, such pyramid models do not give any insight into how CIM would be implemented in an

organization. Obviously another methodology for modelling is necessary in order for the model to meet the needs outlined above.

We found that there has not been a systematic method for defining a generalized CIM model, and so it was decided to develop a model from scratch which would enable us to do the following,

- o understand the structural elements of CIM
- o analyze the fundamental processes within CIM
- o generate a reference model which could be used to identify how our own products should be incorporated into CIM systems

Because we view data and information movement as a fundamental property of CIM systems it is appropriate to use Structured Design Techniques which offer the ability to include real time properties, data flows and data transformations. Such models are similar to energy, mass and material flow models widely used in engineering. To date structured design techniques have played a key role in understanding the automation needs of our clients. This has led to a two-way benefit. We gain credibility with our clients by demonstrating an understanding of their business and we gain confidence in being able to address our clients needs. Therefore, the use of Structured Design techniques is very appropriate.

This paper will first discuss what is to be modelled, and after a brief review of structured design techniques the model for CIM will be described using a context diagram at the highest level followed by decomposition of the processes identified in the context diagram. Following the model description, we will discuss how we checked the applicability of the model to some CIM type systems, namely a pharmaceutical packaging facility. Some implementations of the CIM processes will then be presented, followed by a brief discussion of some of the more salient points of the model and some of the concepts which we have drawn from the model.

## 2.0 REQUIREMENTS FOR A CIM MODEL

The subject being modelled is a completely integrated business system. In setting out to develop the CIM model, a list of features to be described by the model was first developed.

- o There must be a real time exchange of data and information between different business or work areas and that real time would have different meaning to different users.
- o There will probably be a need to integrate a number of local databases which already exist throughout the business

enterprise.

- There will be a need to integrate all aspects of the business environment which means a diverse use of data, information and resources.
- There will be existing resources as well as a need to share resources such as data.
- Areas of automation will most likely already exist and be the nucleus of the CIM system.
- Future expansion of the automation system will certainly occur.
- A wide range of packaged application programs will be used by the CIM system including such applications as statistical analysis, material and resource planning (MRP) and payroll systems.
- The system must provide for user friendliness and easy accessibility.
- The system must provide useful and accurate data in a form appropriate for all users.
- The model must be understandable and easily adapted to specific applications.
- The model must not be specific to a particular hardware environment.
- The model must be sufficiently rigorous and complete to be useful.

The model was then developed by identifying data flows around a typical business system and the processes that act upon the data to transform the data to useful information. The data flows identified do not provide a comprehensive picture of a business, but are sufficiently characteristic for the design purposes.

## 3.0 STRUCTURED DESIGN TECHNIQUES

Before developing the model it is necessary briefly outline the techniques and tools of structured design. A more detailed discussion is available elsewhere(2). The elements of structured analysis that we used are,

- Context Diagrams
- Data Flow Diagrams
- State Transition Diagrams
- Mini-Specifications
- Data Dictionaries
- Walk-Throughs

### 3.1 Context Diagrams.

Context diagrams are used to model the boundary of a system. It is effectively a statement of scope in which the system is bounded by processes over which there is no control. Within the boundaries of the context diagram individual processes transform data according to rules of the process.

### 3.2 Data Flow Diagrams.

Data Flow Diagrams (DFD) are used to model the flow of data and show the processes which transform data in the system. Data Flow Diagrams are layered to decompose the system into functional parts.

3.3 State Transition Diagrams.

State Transition Diagrams (STD) are used to model the dynamics of the system and show how the state of the system can change and the paths between these states.

3.4 Mini-Specifications.

Mini-Specifications are pseudo-code descriptions of what is happening inside a process.

3.5 Data Dictionary.

The data dictionary contains the definitions of all the terms used in the context diagrams, DFDs and STDs and mini-specs.

3.6 Walk-Through.

A walk-through is a procedure used for reviewing the designs, specifications, programs or other technical materials and checking them for errors and completeness. Walk-through allow de-bugging to take place before the implementation phase of the project.

Most of these tools and techniques were used during development of the model,although Data Flow Diagrams and Walk-Through were the most helpful for developing a coherent CIM model. In fact the model presented here is the result of several iterations as a result of walkthroughs.

## 4.0 THE STRUCTURED MODEL FOR CIM

4.1 Model Description.

The environment of a generic CIM system is shown in Figure 2, which is the system context diagram. The generic CIM system is structured around a Common Data Manager which is consistent with the concept of transferring information and data around the business enterprise.

Around the Common Data Manager are User, Business and Production processes. The user is primarily concerned with requesting and receiving data from the system. We have made the distinction that the user represents a manual, human interface to the CIM system, in contrast to the business and production processes which are more automated in nature and may include applications packages which take data from the system depending upon the application.

4.2 Business processes.

Business processes are the administrative and management units of a plant or facility. The processes provide support to production processes. Examples of business processes are the functions provided by purchasing, sales, marketing, finance, accounting, and human resources. They are secondary to production in the sense that their functions do

not directly influence manufacturing on an hourly basis.

Data used by the business processes is production data which is analyzed for production management information. Additionally, the data is likely to be used to for developing the goals and objectives for the production process. Both data flows from the business processes are shown passing into the common data manger where the data is available to others.

4.3 Production processes.

The concept of the production process is similar. Production schedules and supervisory control data is passed into the production process. This data is transformed by the particular manufacturing process into actual production data for the business. An extra data flow into the production process, an Interface Configuration, is also shown. This data flow contains information about how data is to be gathered from the production cells. An example would be a request for total hourly production from each cell.

Each process is now decomposed into the next lowest level to describe each processes in more detail

4.4 Area Controllers.

It was found that business and production processes could both be decomposed into the same generic structure, which has been called an Area Controller, as shown in Figure 3.

The elements of an area controller are a communications interface to the data manager, operating procedures, data stills, a communications interface to the physical devices in the cell, and a local area control database.

4.5 Communication Interfaces.

The communications interface accepts arriving data streams and transforms them into the appropriate format for upstream or downstream users.

The communications interface performs limited data manipulation. Its main function is to provide a logical connection to other CIM functions and it is the lowest level of interconnection between devices and the system.

Examples of communications interfaces are,

- o Gateways (includes hardware, software and specific processes).
- o Database distributors.
- o Mail Routing.
- o Interfaces to other network protocols.

The communications interfaces may be levels 2 through 5 of the OSI model(3) which are,

Level 2 - Congestion Control (Node to Node)

Level 3 - Traffic Flow (End to End)

Level 4 - Network Access and Security (Access)

Level 5 - Interprocess Communications (User)

Note that the communications interface assumes some form of networking methodology. Consequently, all devices on the network must have a physical connection to the communications media. It is the responsibility of these devices to provide a minimum of OSI Level 1 (Link Control) connectivity.

4.6 Operating Procedures.

Business operating goals and objectives are created by analyzing information such as current sales, market trends, current inventory levels, and production capacity. The objectives are then communicated to the manufacturing arm of the organization which determines production schedules and receives production feedback. This very general activity is extremely complex and can be broken down into sub-processes.

Within a production cell, an example of operating procedures would be area optimization, energy management and statistical process/quality control.

4.7 Data Still.

The main function of the Data Still is to manipulate raw data into valid meaningful data. An example of data distillation in a production cell is the conversion of a bottling line count, received from a packaging machine, into total bottles per lot. The lot quantity is subsequently archived and made available to an inventory update procedure. Other examples of production cell data still functions are data validation, summations, calculation of moving averages, shift summaries and efficiency calculations. Examples of a data still in a business cell are sales summaries and project cost summaries.

4.8 Local Area Control Database.

During development of the model it was recognized that any real CIM system would contain a number of local databases. For example, a production cell may contain a control system database which would not be available to other users within the overall system. In fact it may be a specific requirement to preclude global access to certain local databases for security reasons. It was also recognized that local databases are necessary for local data storage prior to processing through the still. Additionally, operation and system security considerations usually mean that individual cells should continue to operate independently in the event of problems elsewhere in the system. This is commonly referred to as distributed processing.

4.9 Common Data Manager.

The common data manager is

shown in Figure 4 and consists of communications interfaces, data templates, a database manager and a database.

4.10 Database Manager.

The database manager is a software 'engine' whose sole purpose is to maintain the CIM data. Ideally, the Database Management System (DBMS) is consistent between the different business and production cells such as accounting, marketing functions, the engineering workstations, and numerically controlled machines. In reality, however, this is rarely the case.

A central theme throughout the factory automation literature is that the database must be logically central but physically dispersed. The rationale for locating data locally is based upon desired local throughput, frequency of use and as mentioned above, security. If all database applications use the same database management system, integration of the local databases becomes trivial. However, because this is not usually the case, the best alternative is to convert the local data to global data as soon as possible and to use a standard global database when data must be shared between users. This also allows preservation of the local databases together with the benefits discussed above.

A number of advantages are realized by converting data to a standard form as close to the data source as possible. These are,

- o Maximum use of local computing resources
- o Data validation at the origin
- o The data is available to the maximum number of nodes or devices

Since databases and database manipulation are not new concepts, there are a large number of database management systems available on the market.

4.11 Data Template.

The data template provides an interface between the database management system and the area controllers and users. It includes a mask that transforms data between local and global formats. The database manager includes this functionality to convert data into formats for insertion into the database or for converting data extracted from the database by other users and cells. Additionally, the data template handles the procurement of data for the database and the shipment of data to area controllers. Examples of this are,

- o Sending data such as production data to the Database Management System
- o Requesting data from the Data Base Management System such as scheduling data for production cells,

- Handling and queuing transactions to and from the Data Base Management System

- Logging events during system downtime

- Formatting data to and from the communication interface and the Data Base Managemnet System

Ideally an integral part of the data template is to provide common interfacing to the DBMS. This is the same functionality as DIGITAL Standard Relational Interface (DSRI) which is beginning to emerge as a de facto standard.

4.12 User Interface.

The final process shown in the CIM context diagram (Figure 2) is the user. This is decomposed in Figure 5. This distinct functionality is included in the context diagram (Figure 2) because there will always be a need for casual human interface with the CIM system. Part of this functionality will be the creation of ad hoc reports from the database, manual manipulation of the database, and system utilities such as configuration.

Note that it is tacitly assumed that each area will contain a man-machine interface as one of the physical devices in the cell. Consequently, the user interface is included to represent system management functions and the ability for users to initiate sessions on demand rather than by schedule. The latter case of scheduled system use is more representative of the business cells and in particular the production cells.

## 5.0 VALIDATION OF THE MODEL

As in any model development procedure it is necessary to validate the model by comparing its behavior with actual behavior. For validation of the model we applied it to some systems with which we were familiar and checked to see that a CIM system described by the model would be realistic, implementable and provide the appropriate automation solution. Application of the model to a pharmaceutical packaging facility will be described here although other examples were also used.

### 5.1 Pharmaceutical Packaging Line.

The motivation for introducing automation in this example is the improvement of the management and control of a bottle filling and packaging facility. These improvements are to be achieved by,

- o keeping an historical database of the bottled product on a per-lot basis.
- o Maintaining an operator database including batch/lot, shift and personnel utilization data.
- o Inventory and operator data is to be collected directly from the production floor as production occurs.

Using these desired objectives the required data flows were succesfully mapped onto the CIM model. The system context diagram is shown in Figure 6, and a data flow diagram for the process is shown in Figure 7. It may be seen that the model developed in the previous sections can be used to identify the required functionality of the desired automation system.

## 6.0 MODEL IMPLEMENTATION

So far, hardware implementation of a CIM system has not been mentioned. This is completely consistent with our approach for developing the system functionality in terms of data flows using structured techniques. The consequence of this procedure is that the hardware is selected to fit the functionality rather than having a functionality dictated by a hardware solution.

In actuality there are usually many possible hardware implementations for a CIM system once the functionality has been identified. However, the CIM architecture should have a consistent approach in terms of information analysis, data exchange and software development. An advantage of structured design techniques is that they enforce consistency and validation. Therefore, once the design is available any number of implementation methodologies may be used.

After the initial hardware and system solution is reached, there will be many forces

acting to mandate the decision. Again, this reinforces the need for a rigorous and complete functional design so that implementation can be made with care and forethought. The CIM solution must be consistent at as many levels as possible. The networking strategy must remain the same and the global data manager must be consistent. Even the development tools should not deviate from a norm.

6.1 Typical business configurations.

The business areas may reside on a mainframe, a mini-computer or a micro-computer. Typically, many of the business cell applications will already have been at least partially implemented. Characteristically, these will fall under the auspices of material control, finance and accounting, production planning and scheduling, sales and marketing and human resources. When implementing CIM, one of the primary issues is often integration of the existing business cells with production cells.

During integration of the business cells some of the following issues are likely to emerge.

- o Data retrieval -Is existing data in a simple format or in a proprietary database?

  -Can the data be accessed using common techniques?

  -If the data is in a proprietary database expect expensive interface development.

  -If the data is easily accessible integration will be somewhat easier using standard tools.

  -If the data management is accomplished using the same mechanism as the CIM global data manager, then integration will be much easier.

- o Data Archival

  -If there is an interface with an integrated application, all writes to the data must be accomplished using the constraints established by the integrated application.

  -Only as a last resort should a new application write directly to a file that is being maintained by an existing application.

  -Accounting systems must be secure from unauthorized access.

  -Once data is forced into an integrated application, the integrity of the system becomes suspect.

6.2 The Communications Interface.

This process is one of the critical elements of a CIm implementation. The interface must be able to handle multiple devices,

multiple protocols and multiple communications media.

The type of communications interfaces needed is heavily dependent upon the devices that must be supported. Current possibilities are Ethernet and MAP (Manufacturing Automation Protocol).

6.3 Operational Procedures and Data Stills.

The procedures and the stills depend upon the nature of the business, the production cells and the business cells. It is anticipated that in a process plant, for example, a still may be implemented which takes point data collected at one minute intervals, and after data validation, calculates ten minute, hourly and shift averages. These averages are then transferred to the global data base.

An example of a procedure is a boiler dispatch which optimize load distribution between multi-unit multi-fuel steam boilers. Typically such an application would reside on a small host computer with a communications link to a realtime process control system. Inputs to the procedure include fuel costs, which are available from a business cell, current demand filtered or averaged by a still, and boiler efficiencies available from a local database. The procedure outputs the operating points for each boiler to produce an optimal steam cost.

6.4 The Database Manager.

The Database Manager is the heart of the system. It provides the necessary data for other parts of the system to enable them to function in a way that is optimal to the complete enterprise

A common database manager should be chosen early in the CIM implementation process. However, over the past few years, factory automation has frequently been implemented as point solutions or as islands of automation. The approach works well during the prototype phase, but because data compatibility was often not an initial requirement, problems arise when the interfacing phase arrives. By selecting a common data manager at the outset, the effort needed to later interface different systems will be less.

6.5 Data Template.

In practice the data template will often be part of the data base manager. For example, many data base packages include a feature for distributing sections of the database around different users. Utilities are also included which will periodically update the local databases to incorporate changes made in the main database. Other features that may be incorporated in the database manager are transparent access to the database, provided that applications conform to some standard. This is therefore a data template function because it provides format conversion for data.

## 7.0 DISCUSSION

The model and its development demonstrate a number of concepts which are worth mentioning.

Firstly, it was apparent to us that CIM has been around in the continuous process industries such as steel and petroleum for a long time. In these sectors of industry, computers have been directly involved in manufacturing for at least twenty years, although they have tended to be islands of automation as isolated plant process control and automation systems and business systems. Nevertheless, they represent a CIM system.

It has also become apparent to us that each user can be represented in the same way. It can be seen that a generic area controller can be used to describe either a production process or a business process The differentiation between each depends upon the cell product.

During development of the model we found it necessary to include a database in each area to contain cell specific information which is not useful to other users.

Another enhancement made during model development was the inclusion of direct cell to cell communications. After initial attempts at model validation, it was realized that channeling all information through the database manger was not realizable and unnecessary.

After identifying the functional elements of CIM systems it can be seen that the elements are actually distributed around the hardware and software implementation of the CIM system. However they must exist somewhere in the system, and unless they can be identified within the architecture it is unlikely that the system will perform satisfactorily.

The actual characteristics of system must be determined with the implementation team and the plant engineers. Network traffic analysis, system sizing, disk requirements, speed constraints all must be collated into the CIM system. Our conceptual model does not preclude many types of implementations but is does facilitate understanding the requirements.

## 8.0 CONCLUSION

Early in the study we realised that the desire for CIM has been around for a long time and consequently most businesses today will already include islands of automation. Frequently, the integration of these islands of automation will be a principal objective of CIM projects.

Although each CIM system is likely to have a set of unique characteristics and applications, the model which has been developed demonstrates that all CIM systems will share key functional processes. Once these processes are recognised, it is much easier to plan and implement in a methodical and logical manner the complete CIM system. Moreover, by dividing these systems into key elements it is possible to maximise the transportability of these elements both around and between systems.

Because of the nature of the problem, the key to successful implementation of CIM is understanding the functional requirements of the system. Unless these requirements are established at the outset of a CIM implementation, the eventual result is unlikely to match or even resemble the original intent.

# 9.0 REFERENCES

1. Stern Jr, D.E. Tying Islands of Automation into CIM Systems, DEC Professional, 6(11)44-52, November 1987.

2. Ward, P.T. Structured Development for Real Time Systems, Yourdon Press, New York, 1985.

3. Weik, M.H. Communications Standard Dictionary, Van Nostrand Reinhold Company, New York, 1983.

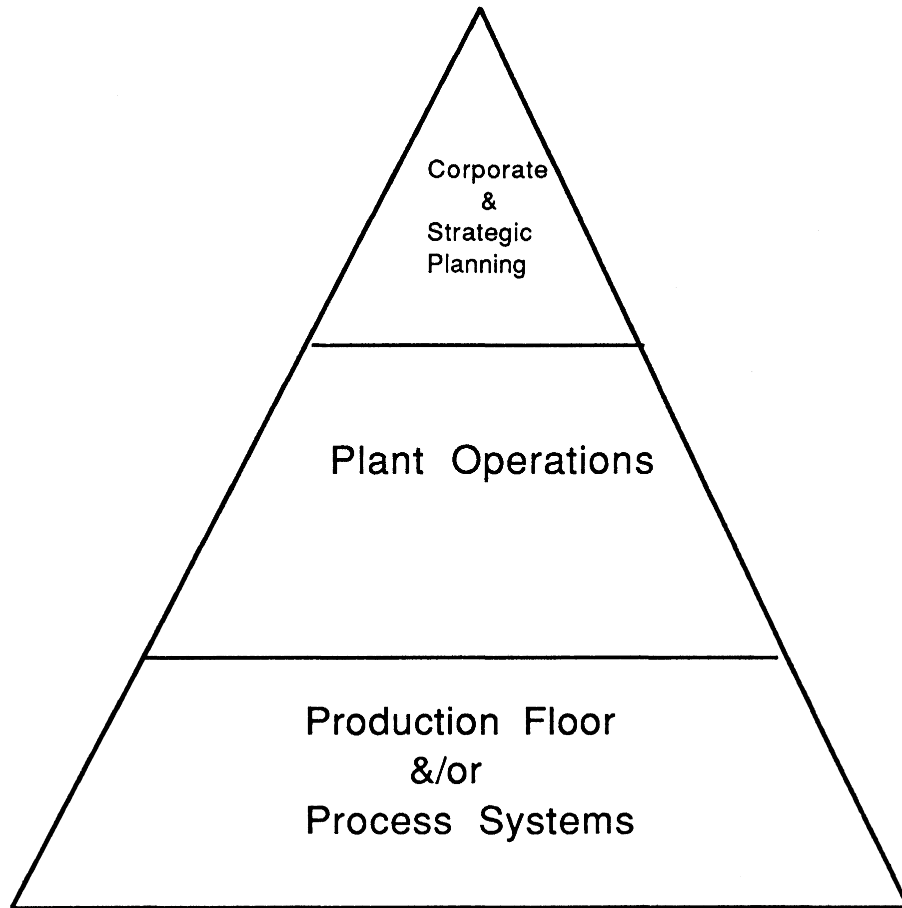



FIGURE 1. The classic CIM model showing a hierarchy of business functions.

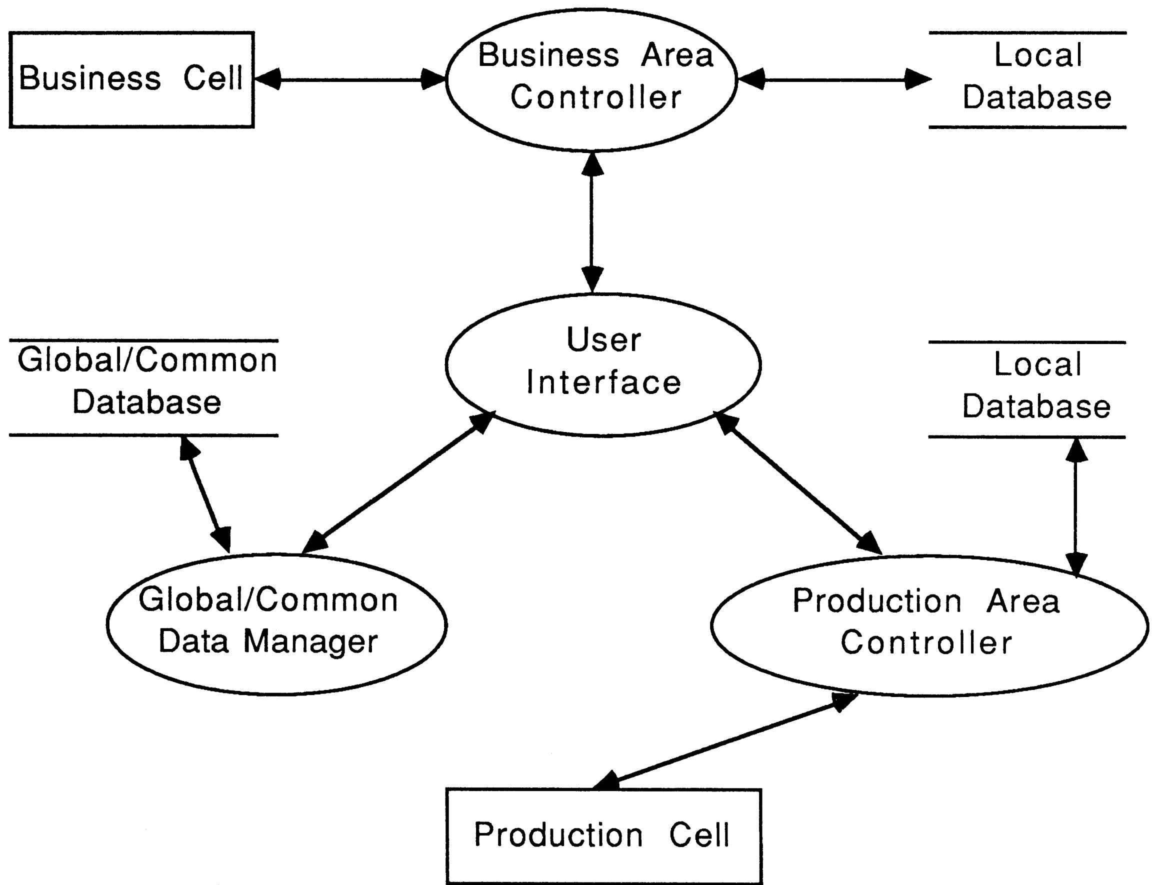



FIGURE 2. Context diagram for a CIM system.

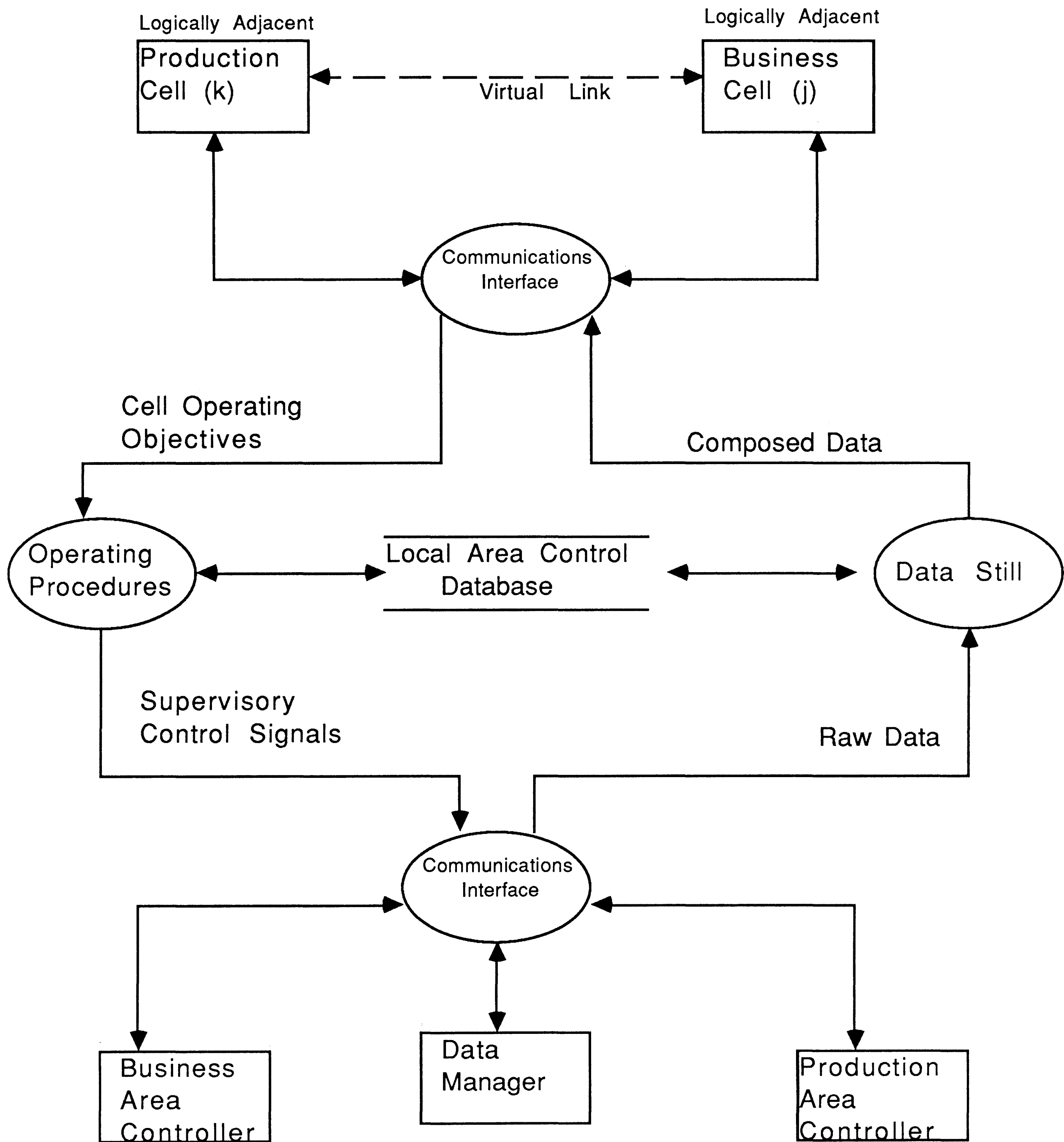



FIGURE 3. Data flow diagram for a business or production area controller.

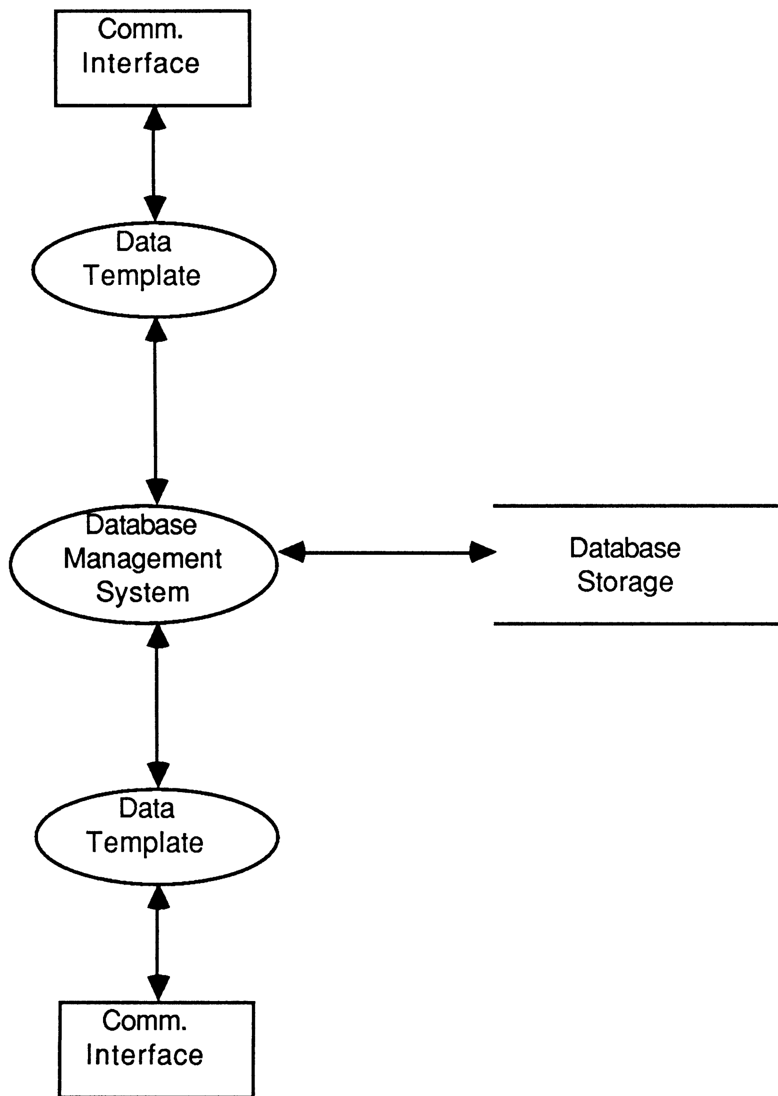



FIGURE 4. Data flow diagram for the Common Data Manager.

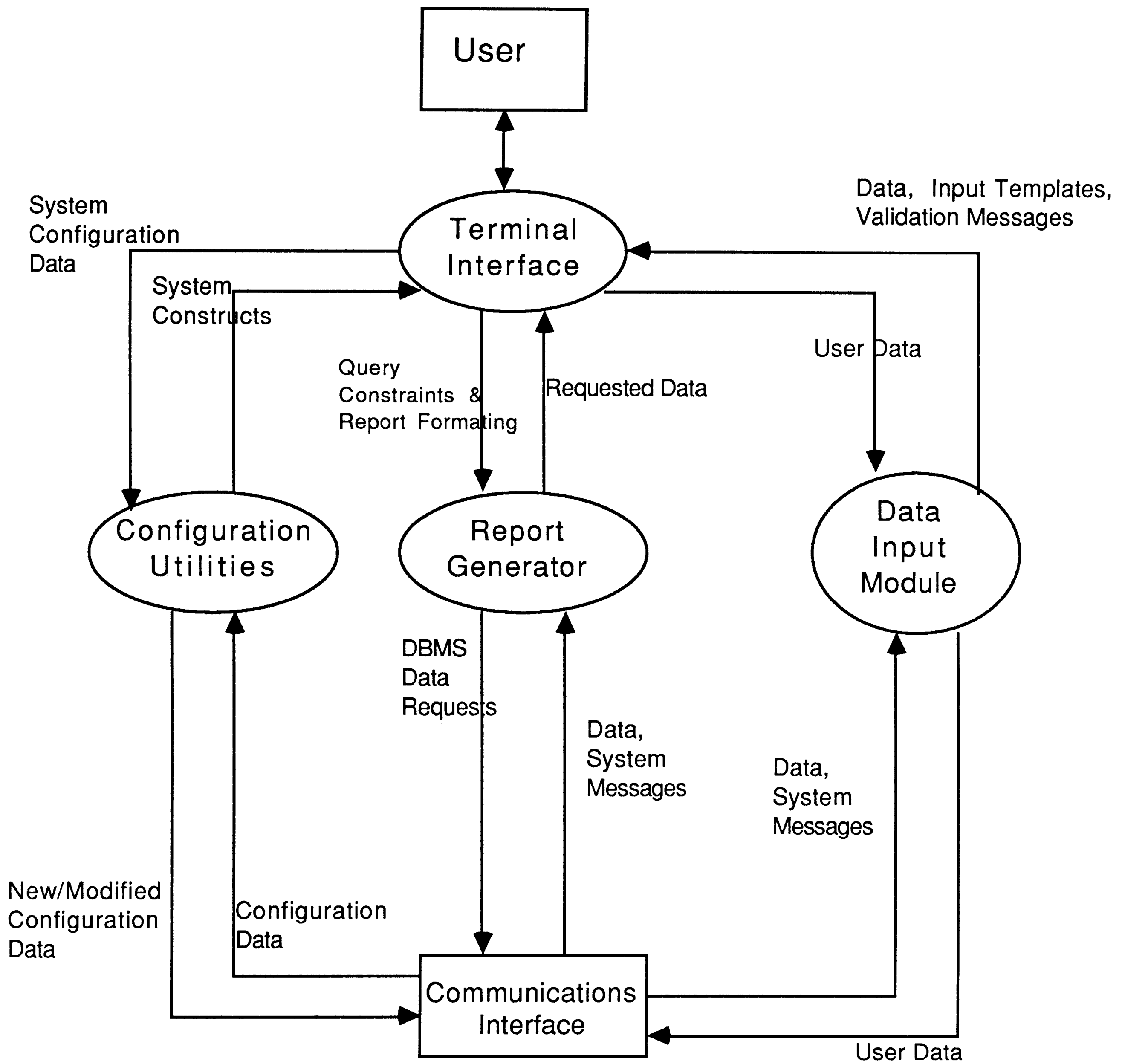



FIGURE 5. Data flow diagram for the User Interface.

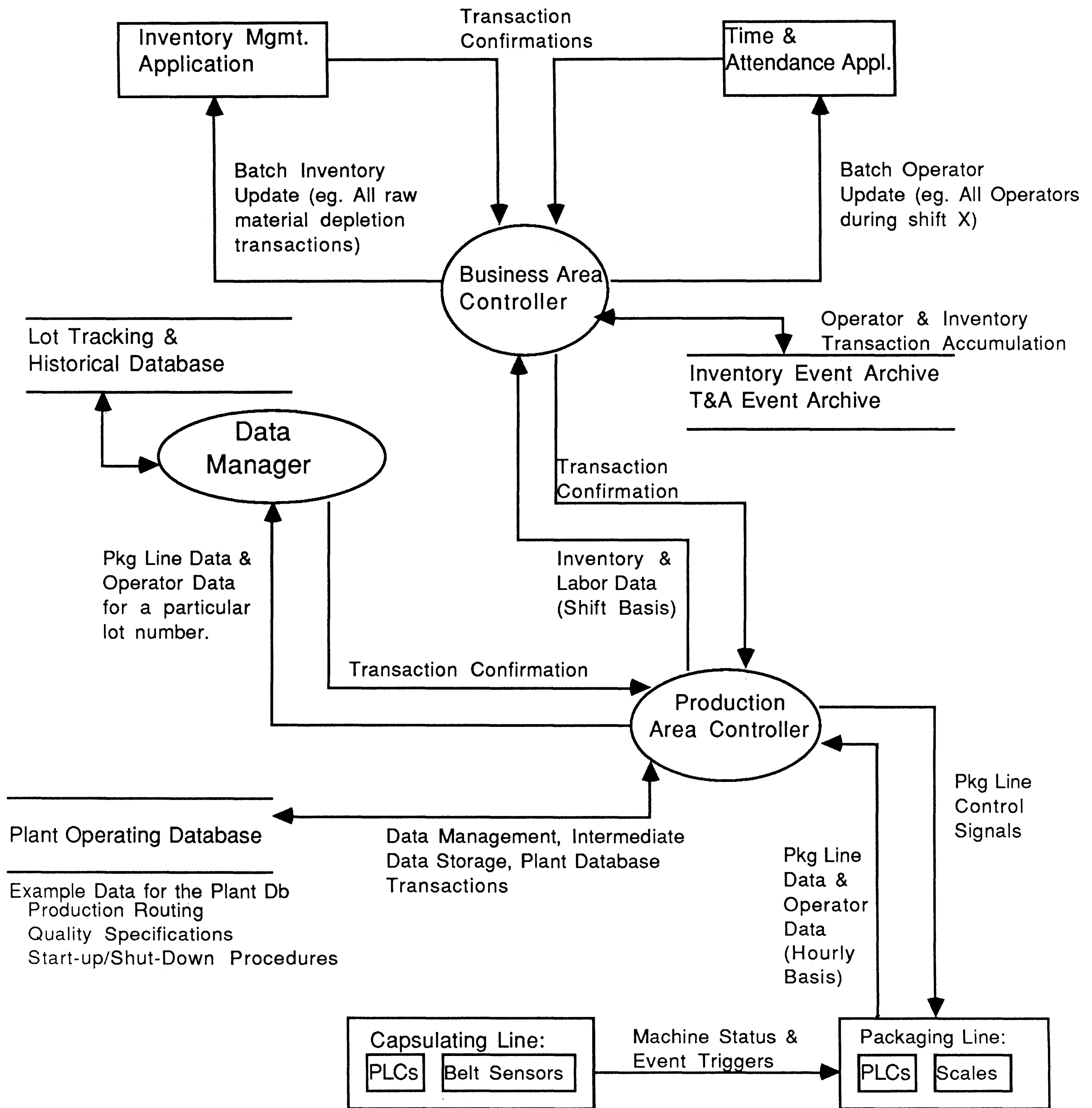



FIGURE 6. Example context diagram for a pharmaceutical packaging facility

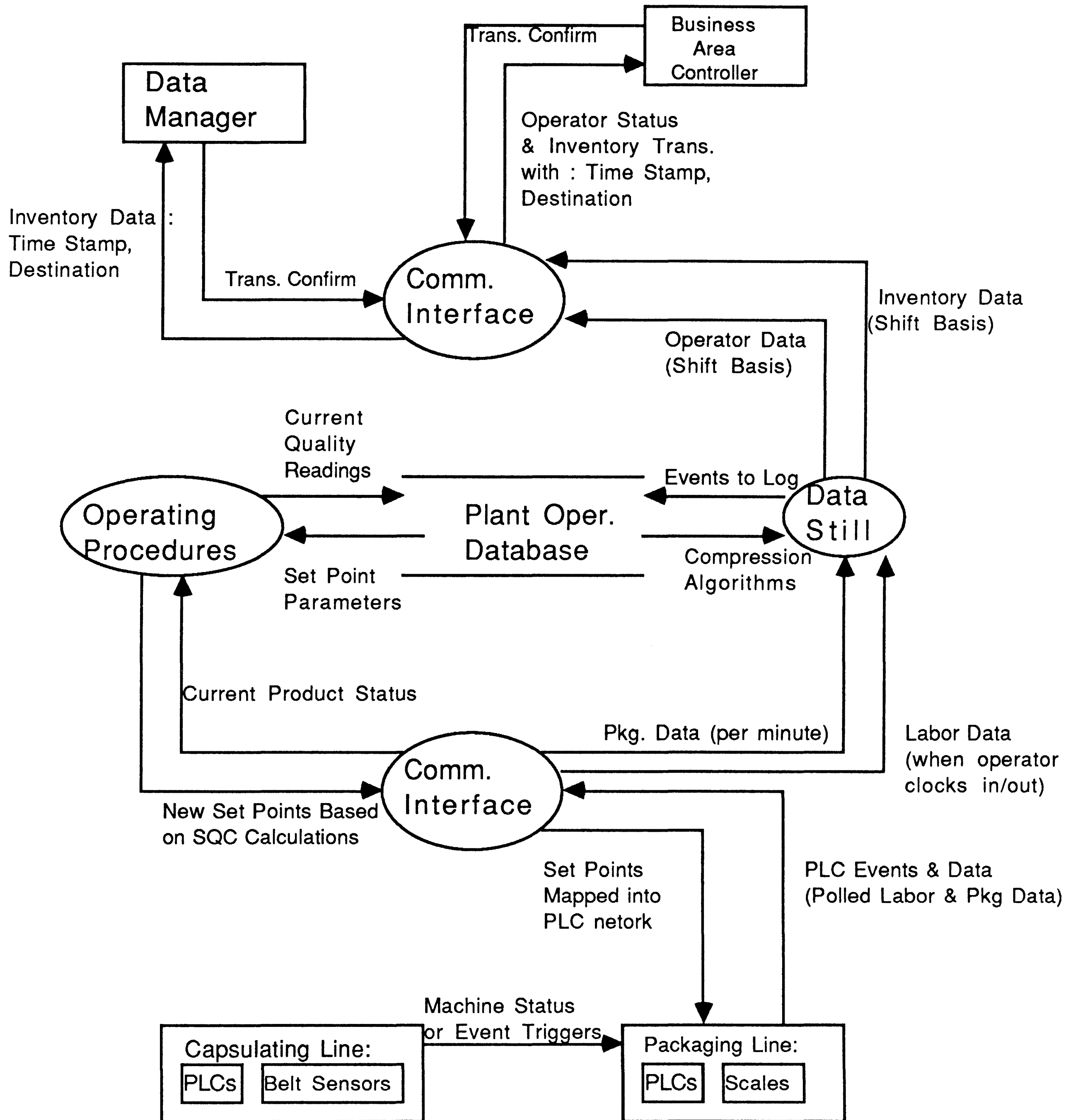



FIGURE 7. Example data flow diagram for a pharmaceutical packaging plant.

# INTERACTIVE CONTROL ENGINEERING COMPUTER ANALYSIS PROGRAM

ROBERT L. EWING
SAM C. HUGHES
KRIS L. LARSEN
GARY B. LAMONT

Department of Electrical and Computer Engineering

School of Engineering
Air Force Institute of Technology
Wright-Patterson AFB, OH 45433

## Abstract

The interfacing and modification of the software program "Interactive Control Engineering Computer Analysis Package (ICECAP)" is presented. This enhanced version of ICECAP uses double precision (D & G Formats) along with new plotting capability by utilizing the GWCORE standard graphics package interface with drivers developed for the VT240, Tektronix 4014 and LN03 laser printer. Methodology, description and problems are included for a detailed analysis of the modification of the package, giving insight into future possible modifications. This package originated at the Air Force Institute of Technology through the efforts of several thesis students.

## Introduction

ICECAP is a computer aided design package, written in both the Fortran and Pascal languages under VMS, and utilizing interactive graphics for control theory design, analysis and performance evaluation. It can plot root locus, frequency response, or time response for either the S or Z domains (Continuous or Discrete). Transfer functions or matrix can be updated or recovered from previous sessions. User friendly transfer function manipulation is provided such as the TUSTIN transform, along with on-line help capability.

## Objectives

Design is largely a trial and error process. A designer often employs heuristics or "rules of thumb" in design, but they rarely result in a final acceptable solution without interactive feedback. CAD tools typically provide most of the manipulative functions required to synthesize and analyze a design based on objects and requirements, but, contain very little knowledge of the design "process" itself. CAD software is a tool, and as much, the user must direct the process and interpret its output. In essence, the designer defines objects (entities) whose attributes are to be instantiated (given values) in an iterative manner. For example, the development of a second-order compensator (object) requires the instantiation of its coefficients (attributes) based upon desired performance requirements (overshoot, damping,...). ICECAP provides input/output dialogue for the coefficients (or poles and zeros) as well as manipulating the attributes to determine performance.

ICECAP attempts to provide a very "friendly" user interface with on-line help for the novice as well as the expert control engineer.

The following list reflects the general CAD tool considerations:

CAD Objectives

Automatic/Interactive Design Process
Improved Effectiveness/Efficiency
Friendly User Interface

Design is a Trial and Error Process

Complex/Not Completely Understood
Iterative Feedback Process

Design Model

Determine Design Goals
Conceptual Design Phase
Design Procedure/Design Knowledge
Design Evaluation

CAD Tool Interface

User Directs Process
Tool Package Guides User at Some Level
Requires Feedback
Partial Synthesis Algorithms
Various Data Presentation Methods
Storage and Timing Considerations

## Structure of ICECAP

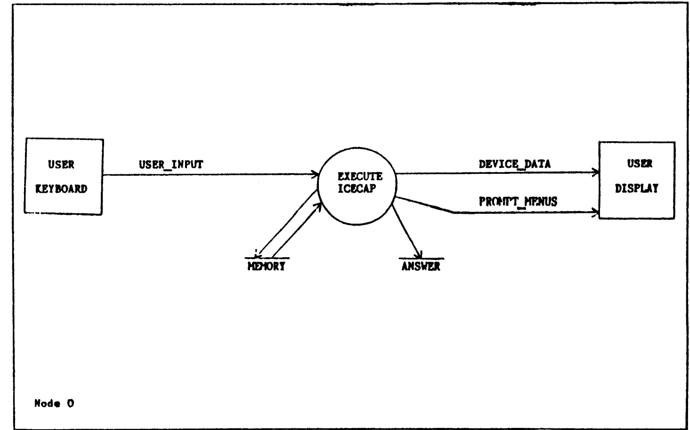
System design data flow diagrams and associated data dictionary entries were used as the software development tools for ICECAP. (See Figure 1 and 2) The data dictionary completely documents every process entry, data flow entry, and data file represented in the data flow diagrams. The data flow diagrams are layered or paritioned into hierarchical levels. The top-level diagram, called the context diagram, defines the interface between the system and its environment. The lower-level diagrams expand into greater and greater detail. This type of documentation provides for ease of maintenance and modification.

## Capabilities

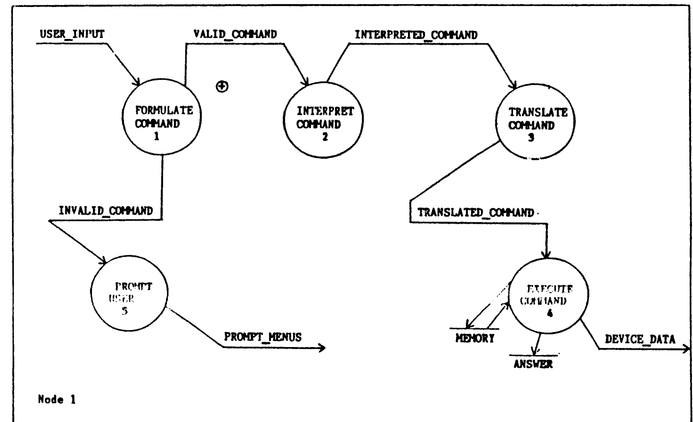
ICECAP (Interactive Control Engineering Computer Analysis Package) is a highly structured modular program (375 subroutines) that provides the tools for a Control Designer's Workbench in the general areas of:

1) Conventional Control Design (Discrete/Continous)
2) Quantitative Feedback Theory Design
3) Modern and Optimal Design Theory
4) Graphics Environment for Performance Analysis

Figure 1




Data Flow Diagram for ICECAP System Diagram




Data Flow Diagram for ICECAP Context Diagram

Figure 2

PROCESS ENTRY

PROCESS NUMBER: 1.0

DATE OF LAST REVISION: 15 AUG 83

PROCESS NAME: EXECUTE_ICECAP

PROCESS DESCRIPTION: EXECUTE_ICECAP represents the execution phase of the computer program ICECAP, a modern interactive computer-aided design and analysis package for control systems.

INPUT DATA FLOW(S): USER_INPUT

OUTPUT DATA FLOW(S): DEVICE_DATA
PROMPT_MENUS

SOURCE(S): USER_KEYBOARD

SINK(S): USER_DISPLAY

FILES READ: MEMORY

FILES WRITTEN: ANSWER
MEMORY

DATA FLOW ENTRY

DATA FLOW NAME: ABORT_COMMAND

DATE OF LAST REVISION: 15 AUG 83

DESCRIPTION: ABORT_COMMAND is a user-supplied character in the command buffer that informs the system that the user wishes to delete the current contents of the command buffer and have subsequent characters processed as a new command.

SOURCES(S): DETERMINE_INVALID_COMMAND_TYPE

DESTINATION(S): PROVIDE_COMMAND_ABORT_MESSAGE

ICECAP has its origins in TOTAL, an interactive control system design program developed at the Air Force Institute of Technology as a research project in 1978.(8) TOTAL was hosted on a Control Data Corporation's Cyber computer. In 1982, it was rehosted on the Vax-11/780 computer, and renamed ICECAP.(6) A front-end Pascal program was added to make the software more user friendly. In 1984, descrete and matrix functions were added. (1, 10, 14) GWCORE (George Washington University's Core) was added in 1985 (3), providing root-locus and time response plots. Quantitative Feedback Theory (QFT) was added in 1986, with double precision and advanced graphics just being completed. A PC version (ICECAP-PC) has been implemented for continuous systems design. A expert system called "TOTAL-EASE" has been developed using ICECAP-PC for designing lead compensators.

MATLAB is also an integral component of ICECAP. MATLAB was originally developed by C. Moler at the University of New Mexico. It was written as a convient tool for computations involving matrices. MATLAB provides access to the LINPACK and EISPACK software; these two packages represent the state of the art in matrix computational methods. EISPACK contains routines for matrix eignevalue computations while LINPACK provides subroutines for solving and analyzing simutaneous linear equations. The MATLAB program has been enhanced with control design functions to form a complete interactive computer-aided control system design package.

General numerical analysis primitives, in ICECAP, perform the solution of simultaneous linear equations, matrix inversion, eigensystem analysis, singular value decomposition, and other matrix decompositions. Other specialized primitives are provided for conventional modern control design. These include root locus design, state feedback design, optimal control design (via the algebraic Riccate equation) and quantitative feedback theory design (QFT) . Both continuous and discrete systems are supported.

### Double Precision Conversion

The conversion of ICECAP from single to double precision deals with the changing of variables and intrinsic functions from single precision to double precision. ICECAP presently has over 98 common blocks that are used in 375-plus modules. These common blocks have from one element up to 50 elements contained in them.

The conversion started with analyzing the local, global and common block variables and deciding which variables needed to be double precision. After the analysis, the .MAP file, which is created when the modules are LINKed together, was an aid in converting the variables to double precision. All the common, local and global variables were changed to double precision and recompiled. After the variables were changed to double precision, the intrinsic functions were ready to be changed. The LANGUAGE SENSITIVE EDITOR and its split screen capability was extensively used with the .LIS files to correct errors in the FORTRAN77 code, during this time.

After all the routines compiled, the recompiled modules were LINKed to all the modules that were not changed. A command file was created using VAX/VMS DCL, so that when a user logs into the system, the command files creates a separate directory for the user and allows the user to run either the single or double precision version of ICECAP.

The present format is D-FLOATING, due to a limitation of a ploynomial root function subroutine, but it is planned by AFIT to code its own polynomial root function so that ICECAP will run in G-FLOATING format. The D-FLOATING format has a 16-digit mantissa and a 2-digit exponent and a range of approximately $0.29E-38 <= x <= 1.7E+38$. The preferred double precision format is G-FLOATING format which has a 15-digit mantissa and a 3-digit exponent and a range of approximately $0.56E-308 <= x <= 0.9E+308$. G-FLOATING has greater precision with the size of the exponent which is more desirable in control engineering than the extra digit in the mantissa. Later development plans also include changing from double precision to quadruple precision. (See Table 1 for sample of Single and Double Precision)

### ICECAP GRAPHICS

The graphics offered by ICECAP consist of printer-type plots, graphics plots and hardcopy plots. Data plots are created by using the DISPLAY or PRINT commands. The printer plots can be run on any terminal are are stored in ANSWER DAT when the print command is used. Drivers exist that support GWCORE graphics for VT and Tektonix terminals. The screen dump subroutine can be used with the VT241 terminal to procure a hard copy of the plot to be printed on an LN03 laser printer. Three types of data plots are available from ICECAP: 1) Time response, 2) Frequency response (including phase, magnitude, Bode and Nichols charts) and 3) Root Locus plots.

Table 1

```
**************** SINGLE PRECISION ICECAP *********************
 ICECAP¿ DEF CLTF FAC
  FACTORED   INPUT OF CLTF
  ENTER NUM & DENOM DEGREES (OR SOURCE): ¿ 0 6
  ENTER NUMERATOR CONSTANT: ¿ 1
  CLTF NUMERATOR (CLNPOLY)          CLTF ZEROS (CLZERO)
(  1.000    )          POLYNOMIAL CONSTANT=    1.000
  ENTER DENOMINATOR CONSTANT: ¿ 1
 ENTER EACH ROOT RE.IM
    HPOLE( 1)= ¿ -3.5 0
    HPOLE( 2)= ¿ -3.5 0
    HPOLE( 3)= ¿ -3.5 0
    HPOLE( 4)= ¿ -3.5 0
    HPOLE( 5)= ¿ -3.5 0
    HPOLE( 6)= ¿ -3.5 0
  CLTF DENOMINATOR (CLDPOLY)        CLTF POLES (CLPOLE)
(  1.000    )S** 6    (   -3.500    ) + J(   0.0000E+00)
(  21.00    )S** 5    (   -3.500    ) + J(   0.0000E+00)
(  183.8    )S** 4    (   -3.500    ) + J(   0.0000E+00)
(  857.5    )S** 3    (   -3.500    ) + J(   0.0000E+00)
(  2251.    )S** 2    (   -3.500    ) + J(   0.0000E+00)
(  3151.    )S** 1    (   -3.500    ) + J(   0.0000E+00)
(  1838.    )          POLYNOMIAL CONSTANT=    1.000
 ICECAP¿ DEF GTF POL
  POLYNOMIAL INPUT OF GTF
  ENTER NUM & DENOM DEGREES (OR SOURCE): ¿ 0 6
  ENTER  1 NUMER COEFF HI TO LO: ¿ 1
  GTF NUMERATOR (GNPOLY)          GTF ZEROS (GZERO)
(  1.000    )          POLYNOMIAL CONSTANT=    1.000
  ENTER  7 DENOM COEFF HI TO LO: ¿ 1 21 183.8 857.5 2251 3151 1838
  GTF DENOMINATOR (GDPOLY)        GTF POLES (GPOLE)
(  1.000    )S** 6    (   -2.422    ) + J(   0.4782    )
(  21.00    )S** 5    (   -2.422    ) + J(  -0.4782    )
(  183.8    )S** 4    (   -3.155    ) + J(    1.371    )
(  857.5    )S** 3    (   -3.155    ) + J(   -1.371    )
(  2251.    )S** 2    (   -4.923    ) + J(    1.117    )
(  3151.    )S** 1    (   -4.923    ) + J(   -1.117    )
(  1838.    )          POLYNOMIAL CONSTANT=    1.000
```

```
********** DOUBLE PRECISION ICECAP **********
 ICECAP¿ DEF CLTF FAC
 FACTORED INPUT OF CLTF
  ENTER NUM & DENOM DEGREES (OR SOURCE): ¿ 0 6
  ENTER NUMERATOR CONSTANT: ¿ 1
  CLTF NUMERATOR (CLNPOLY)       CLTF ZEROS (CLZERO)
(0.100000000000D+01)        POLYNOMIAL CONSTANT=0.100000000000D+01
  ENTER DENOMINATOR CONSTANT: ¿ 1
  ENTER EACH ROOT RE,IM
   CLPOLE( 1)= ¿ -3.5 0
   CLPOLE( 2)= ¿ -3.5 0
   CLPOLE( 3)= ¿ -3.5 0
   CLPOLE( 4)= ¿ -3.5 0
   CLPOLE( 5)= ¿ -3.5 0
   CLPOLE( 6)= ¿ -3.5 0
  CLTF DENOMINATOR (CLDPOLY)     CLTF POLES (CLPOLE)
(0.100000000000D+01)S** 6    (-.350000000000D+01) + J(0.000000000000D+00)
(0.210000000000D+02)S** 5    (-.350000000000D+01) + J(0.000000000000D+00)
(0.183750000000D+03)S** 4    (-.350000000000D+01) + J(0.000000000000D+00)
(0.857500000000D+03)S** 3    (-.350000000000D+01) + J(0.000000000000D+00)
(0.225093750000D+04)S** 2    (-.350000000000D+01) + J(0.000000000000D+00)
(0.315131250000D+04)S** 1    (-.350000000000D+01) + J(0.000000000000D+00)
(0.183826562500D+04)        POLYNOMIAL CONSTANT=0.100000000000D+01
 ICECAP¿ DEF GTF POL
  POLYNOMIAL INPUT OF GTF
  ENTER NUM & DENOM DEGREES (OR SOURCE): ¿ 0 6
  ENTER  1 NUMER COEFF HI TO LO: ¿ 1
  GTF NUMERATOR (GNPOLY)        GTF ZEROS (GZERO)
(0.100000000000D+01)        POLYNOMIAL CONSTANT=0.100000000000D+01
  ENTER  7 DENOM COEFF HI TO LO: ¿ 1.0 21.0 183.75 857.5 2250.9375
                        3151.3125 1838.265625
  GTF DENOMINATOR (GDPOLY)        GTF POLES (GPOLE)
(0.100000000000D+01)S** 6    (-.350000000000D+01) + J(0.000000000000D+00)
(0.210000000000D+02)S** 5    (-.350000000000D+01) + J(0.000000000000D+00)
(0.183750000000D+03)S** 4    (-.350000000000D+01) + J(0.000000000000D+00)
(0.857500000000D+03)S** 3    (-.350000000000D+01) + J(0.000000000000D+00)
(0.225093750000D+04)S** 2    (-.350000000000D+01) + J(0.000000000000D+00)
(0.315131250000D+04)S** 1    (-.350000000000D+01) + J(0.000000000000D+00)
(0.183826562500D+04)        POLYNOMIAL CONSTANT=0.100000000000D+01
```

The printer type plots are the default when ICECAP used with the VT100 terminal. These plots are composed of a number of 80 character arrays which are filled with asterisks for data points, dashes for grid lines, etc. These arrays are written to the screen to form the plots. The resolution is composed of '*'s representing data but will do as an approximation. The printer plots were available prior to this new version of ICECAP.

The graphics plots use the GWCORE graphics package. This is the George Washington University implementation of the SIGGRAPH CORE graphics standard. Drivers for the VT125, VT241, Tektronix 4010 or Tektronix 4014 are used. The VT241 supports 3 colors while the other terminals are monochrome. The VT driver was derived from an existing Tektronix driver. (3) The time response and frequency plots automatically scale the plots. The root locus and Nichols chart allows for both autoscaling and zoom. A grid may be displayed by turning the grid switch on. The Nichols chart allows up to ten constant magnitude and ten constant angle curves to be super imposed on the plot.

Each graphics plot may be saved into a file by the operator. This is done with a bit map screen dump subroutine and is available only on the VT241 terminals. The screen dump subroutine stores the plot screen bit map in SIXEL format and attaches the appropriate parameters to the beginning of the file to give it the correct aspect ratio when it is printed. The plot may be sent to the printer using a utility command procedure which displays the file names in the users directory and sends the SIXEL file of the desired plot to the LN03 in landscape mode. (See Figure 3)

## QFT Capabilities

Quantitative Feedback Theory is a frequency domain design technique that provides robust performance and incorporates disturbance attenuatin and output decoupling despite plant uncertainties. This method may be applied to the following types of systems: linear time-invariant single-input-single-output (SISO), nonlinear SISO, linear time-invariant multiple input-multiple output (MIMO) and nonlinear MIMO. (4)

ICECAP provides the following QFT operations:

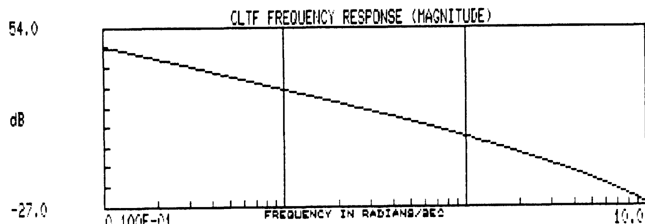
Plant Transfer Function Definition
- Definition of a maximum of 50 plants.

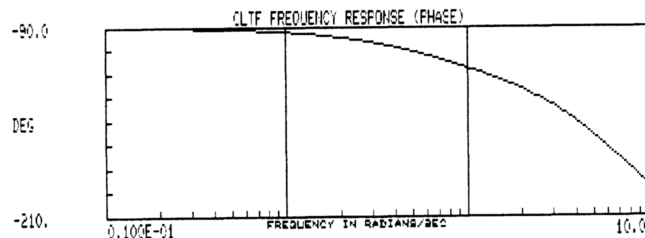
Interactive Tracking Model Generation
- Define upper and lower tracking control ratios(TUTF and TLTF, respectively).
- Display or print the time and frequency response of TLTF and TUTF.

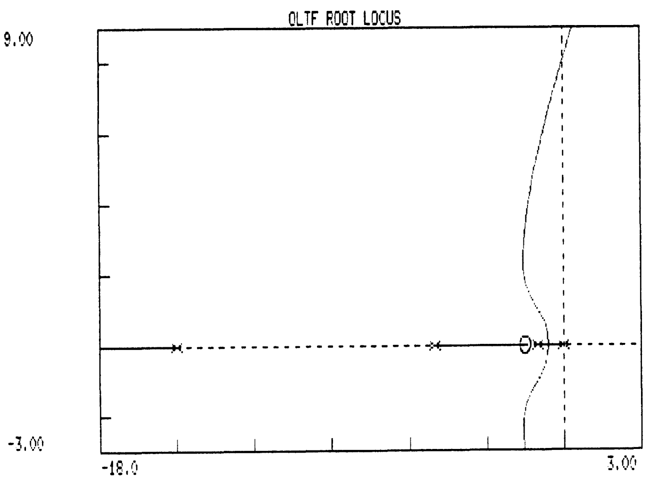
Interactive Distrubance Model Generation
- Define the disturbance model control ratio, TDTF.
- Display or print the time or frequency response of TDFT.

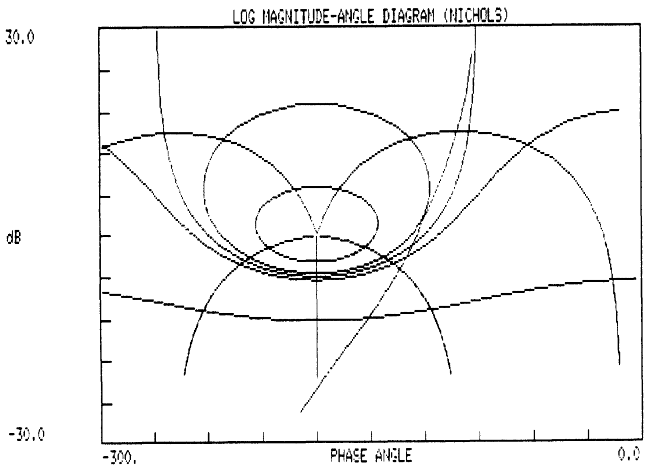
Figure 3

**ICECAP GRAPHICS**




Plant Template Determination

- Template defined as a set of points on the Nichol's chart whose coordinates correspond to the magnitude and phase at a patricular frequency. A curve is not drawn thru the points. For better accuracy, input more plants.

Nominal Plant Selection

Tracking Bound Determination

- Tracking bounds determined, based upon template points described above. Bound determined when where is the difference in the frequency response of the upper and lower models at a particular frequency at M is the maximum difference in M contours.

Disturbance Bound Determination

- Determines the upper disturbance bound.

Composite Bound Determination

- Define this transfer function (LOFT)
- Display or print its time or frequency response.

Prefilter Definition (FTF)

Cascade Compensator Transfer Function (GFT) Formation

Loop Transmission Transfer Function (LFT) Formation

- Determine the disturbance closed-loop transfer function (CLTF) for each plant and display or print its time response.
- Determine the tracking closed-loop transfer function (FCTF) for each plant and display or print its time response.

## Summary

ICECAP runs on versions 4.X of the VMS Operating System or on any Micro Vax or higher class machine. It requires one of the following terminals:

1. VT - 100 (non-graphics)
2. VT - 125
3. VT - 240
4. TEK 4010 / 4014 (graphics only)

Future enhancements by graduate students at the Air Force Institute of Technology will focus on;

Possibility of Quad-Precison (Multi-)

IBM-PC (1985) Version Extended (9, 11, 12)

Interface Module Design

Kahlman Filter Design Additions

Stochastic Performance Additions

3-Dimensional Graphics

Simulation Extended (Users Inputs)

Block-Diagram Manipulation and Display

The LANGUAGE SENSITIVE EDITOR (LSE) is designed to be used in tandem with the SOURCE CODE ANALYZER (SCA), the COMMON DATA DICTIONARY (CDD) and the CODE MANAGEMENT SYSTEM (CMS). They will be used in the future to create a full development environment for ICECAP. The SCA will aid in the analysis of the code and the LSE will allow the ICECAP system to be run and tested while writing the code. The CDD will keep track of the variables in the ICECAP system and their definitions in one location for future developers of the system. The CMS will be used to keep track of all development of the system especially if it is occurring concurrently.

## Biblography


1. Armold, Abraham T., "Further Development of an Interactive Control Engineering Computer Analysis Package (ICECAP) for Discrete and Continuous Systems", M.S. Thesis, AF Institute of Technology, WPAFB, 1984.

2. Birdwell, J.D., et al, "Cascade," Proceedings of 2nd IEEE Symposium on Computer-Aided Control System Design, 1985.

3. Bullard, John R. Interactive Computer Graphics for Analysis and Design of Control Systems. MS Thesis, Wright-Patterson Air Force Base, Ohio: Air Force Institute of Technology, Dec 1985.

4. Cole, Sandra R.H. A Computer-Aided Design Package for Quantitative Feedback Theory. M.S. Thesis. Wright-Patterson Air Force Base, Ohio: Air Force Institute of Technology, Dec 1986.

5. D'Azzo, John J. and Constantine H. Houpis, Linear Control Systems Analysis and Design. McGraw-Hill, 1981.

6. Gembarowski, Charles J., Development of an Interactive Control Engineering Computer Analysis Package (ICECAP) for Discrete and Continuous Systems. M.S. Thesis. Wright-Patterson Air Force Base, Ohio: Air Force Institute of Technology.

7. Houpis, C. and Lamont, G.B., Digital Control Systems: Theory, Hardware, Software, McGraw-Hill, 1985.

8. Larimer, Stanley J., An Interactive Computer-Aided Design Program for Digital and Continuous Control System Analysis and Synthesis. M.S. Thesis. Wright-Patterson Air Force Base Ohio: Air Force Institute of Technology, Mar 1978.

9. Mashiko, Susan K. and Gary C. Tarczynski, "Development of a Computer Aided Design Package for Control System Design and Analysis for a Personal Computer." AFIT/GE/EEG/885D/ WPAFB, Ohio, Dec, 1985.

10. Narathong, Chiewcharn, A Modern Control Theory Enhancement Tool to an Interactive Control Engineering Computer Analysis Package (ICECAP). M.S. Thesis. Wright-Patterson Air Force Base Ohio: Air Force Institute of Technology, Dec 1986.

11. Parisi, Vincent M., Development of a Computer Aided Design Package for Control System Design and Analysis for use on a Personal Computer," AFIT/GE/EE/83D-53, WPAFB, Ohio, Dec 1983.

12. Schiller, Mark W., "TOTAL EASE": A Personal Computer Based Expert System for Control System Engineering", M.S. Thesis, Air Force Institute of Technology, WPAFB, Ohio, Dec 1986.

13. Travis, Mark A., Interactive Computer Graphics for System Analysis. M.S. Thesis. Wright-Patterson Air Force Base, Ohio: Air Force Institute of Technology, Dec 1983.

14. Wilson, Robert E., Continued Development of an Interactive Control Engineering Computer Analysis Package (ICECAP) for Discrete and Continuous Systems. M.S. Thesis. Wright-Patterson Air Force Base, Ohio: Air Force Institute of Technology, Dec 1983.

**RT-11/VMS Networking for Real-Time Applications**

**Jonathan D. Melvin, Ph.D.**
**California Institute of Technology**
**Mail Stop 104-44, Pasadena, CA 91125, (818)356-4126**
**and**
**Interfield Research Associates**
**2314 La Mesa Drive, Santa Monica, CA 90402, (213)395-5841**

**ABSTRACT**

Very fast (2.5 to 4 MBits/sec) Ethernet software allows LSI-11s and PDP-11s to serve as powerful front-end real-time processors for VAXes and microVAXes in a number of industrial and academic settings. This IRANET software is easily configured for any kind of data acquisition or process control application. Operating applications include nuclear data collection, plasma physics measurements, semiconductor device testing, and medical imaging. With this software, multi-processor 16-bit plus 32-bit systems are assembled which outperform 16-bit- and 32-bit-only systems.

## INTRODUCTION

It is important to recognize that 16-bit PDP-11 and LSI-11 computers with 1970s architecture can help solve today's most demanding real-time problems. Real-time operations requiring fast interrupt response and efficient hardware control are easily implemented on 16-bit computers running simple real-time operating systems (RT-11). Complex data analysis and process control are easily programmed on 32-bit computers running full-function operating systems (VMS). By linking 16-bit and 32-bit computers with high-speed Ethernet communication software, multiprocessor systems have been developed which incorporate these 16-bit and 32-bit advantages.

This paper describes IRANET, a real-time network software system developed for such high-speed communication between 16-bit and 32-bit computers by Interfield Research Associates in Santa Monica, California. Using IRANET, networked systems are solving problems ranging from nuclear data acquisition to medical image processing at Caltech, Kodak, General Dynamics, radiology facilities at a number of large hospitals, and other locations.

## OVERVIEW OF IRANET

### Hardware Configuration

IRANET connects one or more RT-11 computers, called "satellite computers", to a VAX or microVAX computer, called a "host computer", with standard Ethernet hardware. Connections through DELNIs, thin- or thick-wire Ethernet cables, and fiberoptic repeaters are all supported. Multiple hosts can communicate on one cable, and IRANET can communicate on a single cable concurrently with DECnet, TCP/IP, and other network software systems. Figure 1 illustrates the typical hardware configuration.

### Unique Features

IRANET differs from other network systems in three ways:

1. It is designed specifically for real-time communication, i.e., it implements
   a. fast data transfer;
   b. fast communication of unexpected events over the network.

2. It uses no network-specific protocols:
   a. satellite computers communicate over the network as if with disks (called "pseudodisks");
   b. host computers access pseudodisk data in shared-memory regions (VMS global sections). Pseudodisk data are automatically saved in host

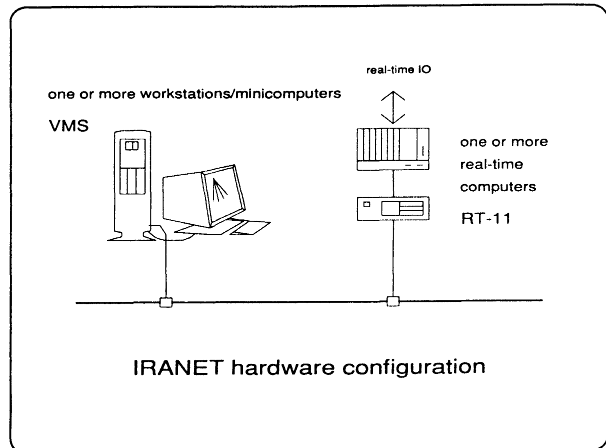



Figure 1: Hardware Configuration for IRANET

disk files by the VMS operating system. Figure 2 illustrates the "protocol-free" communication.

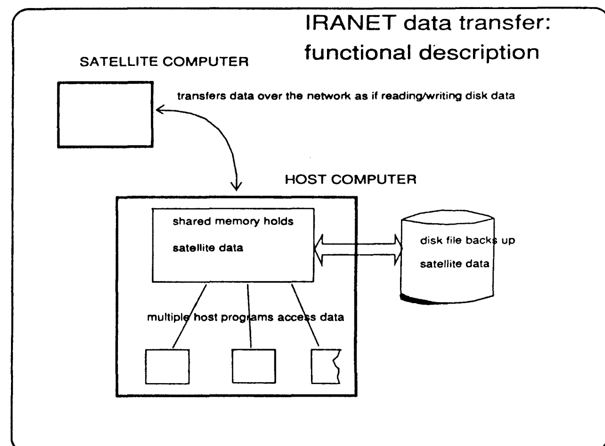



Figure 2: Diagram of IRANET Communication

3. User tasks can be easily added to the network: Any user-programmed subroutine can be added to the host software so that it executes when a satellite reads or writes specific files on a pseudodisk. Typical user-programmed tasks implement
   a. analysis by the host of data acquired by a

satellite;

b. feedback control by the host of computer-controlled equipment interfaced to the satellite;

c. access to host-based data bases in response to satellite requests.

## Applications

The software structure of IRANET lends itself to implementation of networked RT-11/VMS applications. A typical RT-11/VMS application which includes data acquisition, process control, graphics, and an operator interface is diagrammed in Figure 3. IRANET carries data and control information between satellite and host computers. User-programmed tasks in the host software provide menus for system control and live graphical data display in two separate windows on a VAXstation-II screen.

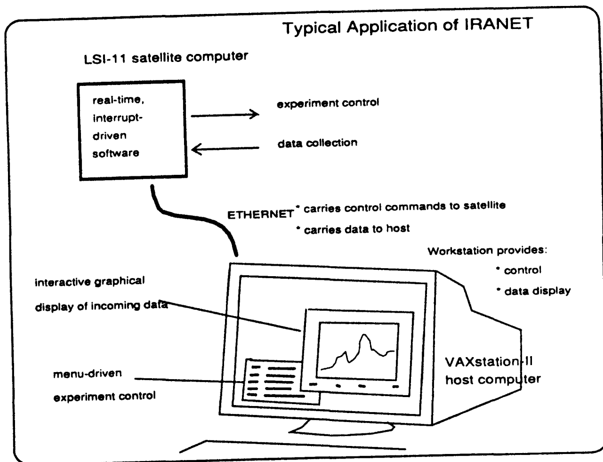



Figure 3: Diagram of a Typical Application

Code in the appendix illustrates programming of IRANET's "protocol-free" communication. Operating applications include:

1. Medical image processing and archiving;
2. Nuclear data acquisition;
3. Plasma physics experiments;
4. Laser-doppler velocimetry measurements;
5. Magnetic media testing;
6. GaAs device testing;
7. Molecular beam epitaxy system control.

## PERFORMANCE

Figure 4 compares performance of IRANET with other network software executing on the same computers and Ethernet interfaces. The simple, real-time design of IRANET give it two-to-ten times greater speed.

Notes follow on IRANET data transfer speed measurements: (1) 4 MBits/sec is measured for host-to-satellite communication. Satellite-to-host communication speed is 2.6 MBits/sec, limited by microVAX-II processor speed. The new microVAX-3000 series processor should be considerably faster. (2) VAXelan speeds reflect one-way communication (no guaranteed delivery of data). DECnet speed is measured during file copy where delivery is guaranteed. IRANET speeds include guaranteed delivery.

IRANET speed for response to unexpected events is determined by measuring the time for: (1) a satellite computer to request data over the network; (2) a host to execute a user-supplied task associated with the satellite's

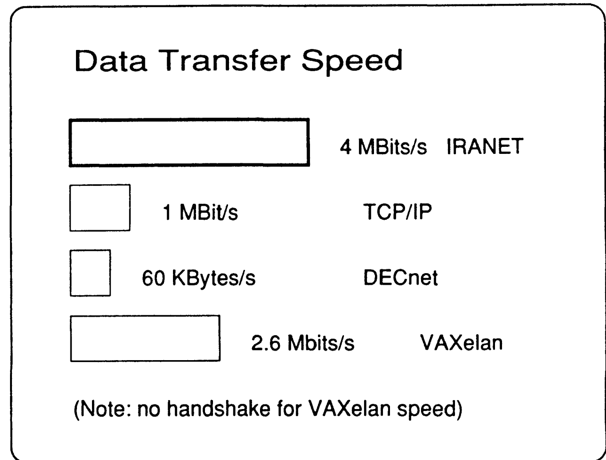



Figure 4: Real-time Performance

data request; and (3) subsequent delivery of the data to the satellite (see Figure 8 below). The average for this time is 5.7 milliseconds on a LSI-11/23 or LSI-11/73 communicating with a VAXstation-II over DEQNA interfaces. This time compares favorably with the absolute minimum communication time measured by DEC personnel for VMS Ethernet device drivers and DEQNA interfaces on microVAX-IIs (4 to 5 milliseconds — DECUS talk DA051, December 7, 1987, Anaheim, California).

## TECHNICAL DESCRIPTION OF IRANET

### IRANET-Operated Ethernet Interfaces

On the satellite computer, IRANET includes disk-like device drivers with which RT-11 programs can access Ethernet though DEQNA or Interlan NI1010 or NI2010 interfaces. IRANET host programs communicate through the VMS QIO interface with standard DEC Ethernet device drivers. Any VMS-supported Ethernet interface can be used on any VAX, microVAX, or VAXstation. Furthermore, IRANET operates on VMS systems through an Ethernet interface concurrently with other network software such as DECnet, TCP/IP, LAT, and LAVC.

### Efficient Communication Protocol

IRANET communicates with protocols diagrammed in Figure 5. Three features of this protocol provide optimal real-time performance:

1. Minimum memory-to-memory copy. While many network software systems copy network data several times before delivery to a user program, IRANET avoids data copy entirely for host-to-satellite communication. It copies data only once for satellite-to-host communication. Consequently, network communication occurs near maximum Ethernet interface hardware speed.

2. Many packets are sent before acknowledgement is required -- IRANET assumes that communicating computers have sufficient speed and memory to receive a sequence of packets without error. This assumption minimizes the communication delays associated with transmission of acknowledgements. However, all communication is checked to guarantee accurate data delivery. In the rare case of error (due to excessive network traffic or computer load) transfers are retried as necessary.

3. Large memory buffers for receipt of data are assumed. IRANET does not notify a remote

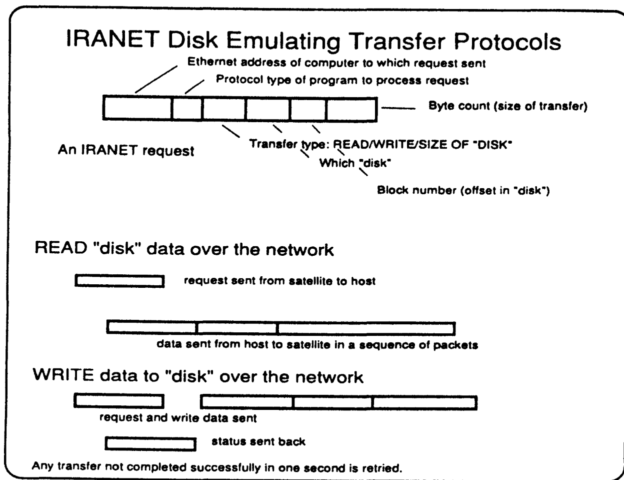



Figure 5: Network Communication Protocol

computer in advance when a large block of data will be sent. Thus, advance notification delays are avoided.

## Simple Network Addressing

IRANET uses simple network addressing in order to optimize communication speed. The satellite computer directs communication to a host by using the host's physical Ethernet address. Further, the satellite selects a single program on the host by using Ethernet protocol types assigned to the satellite. (When IRANET is installed, the user assigns two protocol types to each satellite. These protocol types must not be in use by other active network software such as DECnet or TCP/IP. A host program is associated with a particular satellite by assignment of the same pair of protocol types. Up to 32768 satellites can address separate programs on a single host.)

## Configuration of Basic IRANET Applications

Figures 6, 7, and 8 illustrate configuration of IRANET for three basic applications: data collection, process control, and network-activated execution of user-supplied tasks. Complex applications consist of a combination of these basic applications.

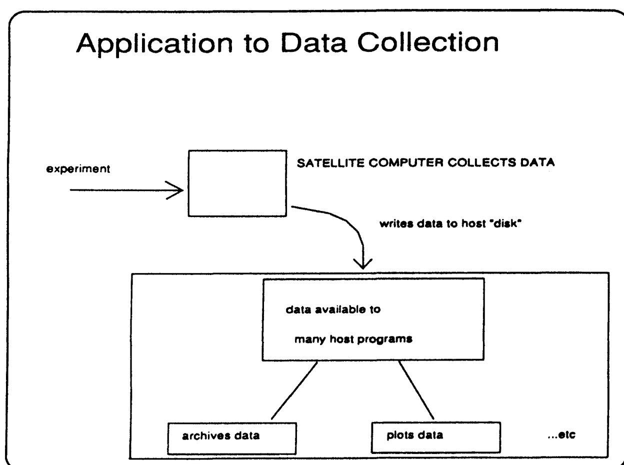



Figure 6: Data Collection

**APPENDIX**

Sample Data Acquisition/Analysis Program Using IRANET

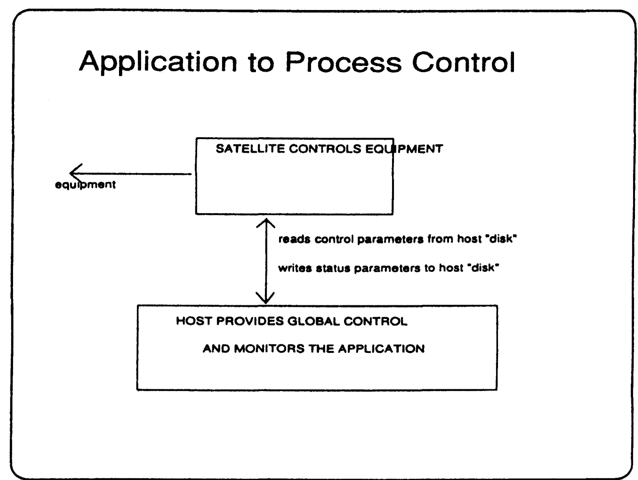



Figure 7: Process Control

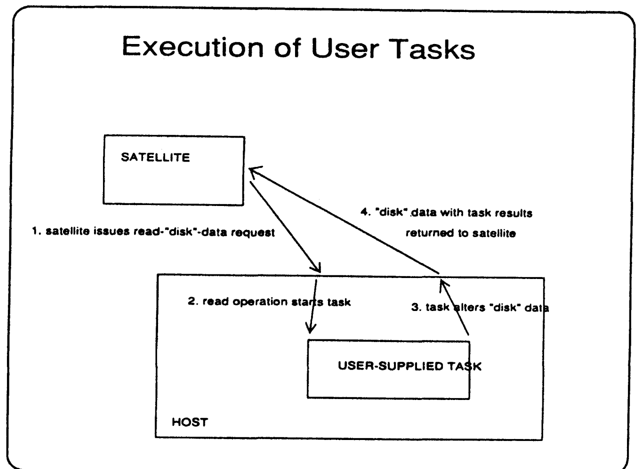



Figure 8: Execution of User-programmed Tasks

**At the RT-11 end (the network looks like a disk named NT):**

```
      PROGRAM ACQUIR
      INTEGER*2 BUFFER(256),CONTRL(256)
c
c   Open channels to data and control files
c   on pseudodisk NT
c
      OPEN(UNIT=1,NAME='NT:DATA.DAT',TYPE=
     *  'OLD',ACCESS='DIRECT',RECORDSIZE=128)
      OPEN(UNIT=2,NAME='NT:CONTRL.DAT',TYPE=
     *  'OLD',ACCESS='DIRECT',RECORDSIZE=128)
c
c   Acquire data and transmit data to host for
c   analysis until instructed to stop
c
10    CONTINUE
        CALL ACQUIR(BUFFER)         !acquire
        WRITE(1'1)BUFFER        !send data over net
c   (causes host READ_ROUTINE to execute)
        READ(2'1)CONTRL         !get control from net
        IF(CONTRL(1).NE.0)GOTO 10
      CLOSE(UNIT=1)
      CLOSE(UNIT=2)
      END
```

**At the VAX end (common blocks can be mapped to network data):**

```
      PROGRAM DAQ
      INTEGER*2 BUFFER(256),CONTRL(256)
      EXTERNAL READ_ROUTINE,WRITE_ROUTINE
c
c   DAQ must be linked with IRANET subroutines
c   and with the page-aligning link options
c     PSE=BUFFER_DAT,PAGE
c     PSE=CONTRL_DAT,PAGE
c   so that the following common blocks are mappable
c   to RT-11 files:
      COMMON/BUFFER_DAT/BUFFER
      COMMON/CONTRL_DAT/CONTRL
c
c   Set up network disks and Ethernet protocol
c   types for communication with the satellite
c   computer; set up READ_ROUTINE and
c   WRITE_ROUTINE to be executed when the
c   satellite reads data from BUFFER.DAT and writes
c   data to CONTRL.DAT respectively; start IRANET.
c
      CALL SETUP_AND_START(READ_ROUTINE,
     *  WRITE_ROUTINE,BUFFER,CONTRL)
c
c   Map common blocks BUFFER_DAT and CONTRL_DAT
c   to RT-11 files BUFFER.DAT and CONTRL.DAT
c   in pseudodisk NT
c
      CALL MAP_BLOCKS(BUFFER,CONTRL)
c
c   hibernate until the satellite sends data
c
      CALL SYS$HIBER()
      END

      SUBROUTINE READ_ROUTINE(BUFFER)
c
c   Analyze data in BUFFER from satellite
c
      INTEGER*2 BUFFER(256)
      ...
      END

      SUBROUTINE WRITE_ROUTINE(CONTRL)
c
c   Instruct satellite to stop acquisition
c   after 100 buffers of data are acquired
c
      INTEGER*2 CONTRL(256),COUNT
      DATA COUNT/0/
      IF(COUNT.LT.100)THEN
        CONTRL(1)=1            !continue
        COUNT=COUNT+1
      ELSE
        CONTRL(1)=0            !stop
      ENDIF
      END
```

# DATA MANAGEMENT SIG

# An Automatic Source-Code Generator Generating Subroutines for Accessing an Rdb Database

**David M. Hansen**
Battelle Pacific Northwest Labs
Richland, Washington

### Abstract

Some databases can be characterized by a number of related yet independent relations. To avoid the many tedious programmer hours devoted to the repetitious coding of functionally identical database access routines, we have developed an automatic source-code generator which generates FORTRAN source-code with embedded Rdb data manipulation commands. The resulting source-code provides routines to read, write, update, and delete data for each relation in th database. This code generator is providing quick prototyping, ease of maintenance, and insulation of application programs from the underlying database for the implementation and evolution of a distributed integrated manufacturing information system.

### Introduction

The purpose of this paper is to describe the circumstances and motivation for the development of the source-code generator which is currently being used in the design and implementation of an Integrated Manufacturing Information System (IMIS). The paper briefly describes the design and function of the code generator, and reports on some of the advantages and disadvantages of using this approach to software generation. This paper is not intended to be a "how to" discussion but rather to share some experiences and ideas, hoping to stimulate the reader's thinking regarding the development of similar general-purpose software development tools.

### Database Characterization

There are any number of ways that a database might be characterized, including the type of data in the database, the size of the database, the users of the database, etc. The method of characterization which is pertinent to this discussion is the characterization of a database by its primary method of use.

Most databases have a definite pattern of use. There are for example, databases which are primarily "analytical" in the sense that their primary method of use is for data analysis. Queries are often complex and sometimes unpredictable or unanticipated. Analytical databases usually grow at measured rates, often slowly or not at all. A "transaction processing" database can be characterized by queries which are more predictable than the analytical database. The database is often dynamic with a good deal of transient data. As a final example, an "archival" database can be characterized as a data collection and storage database. Its use is characterized by a large volume of data entry. Queries might be simple, single table queries.

### The IMIS Database

While any database will certainly be a hybrid of some or all of the elements composing such a classification scheme, the IMIS database can safely be classified as primarily an archival database. Its main function is to replace the factory's current record keeping system with a paperless, semi-automated data collection and storage system. This is not the only purpose of the IMIS system. Data analysis in both production and quality data are perhaps the most important features of the system. However, those tasks do not comprise the bulk of the system nor do they characterize the vast majority of operation of the system.

Tables in the IMIS database tend to have a one-to-one correspondence with the real world. As a result, data entry takes place on a table-by-table basis, independently of other tables in the system. A quality engineer, for example will be dealing with a single table as he/she enters the quality information while inspecting the product throughout the shift. Likewise a production engineer is dealing with a single table as he/she records production figures throughout the shift. Data in the two tables are implicitly related via timestamps. In fact, the database in general is comprised of independent tables related temporally through the use of timestamps. Routine reports perform the majority of the data analysis tasks in the system, relating all areas of production, quality, as well as maintenance

data, thus synthesizing the overall plant activity during a shift.

It is this "independent yet related" character of the tables in the IMIS database, together with the mode of single table operation which provided both motivation and opportunity for development of the code generator.

Another contributing factor to the implementation of the source-code generator was contained in the early functional specifications and structured design documents where functional database access was generically described in some detail. Five basic database access functions were identified for all tables. Two methods of reading data from a table were specified; the ability to access a single row via a unique key, and the ability to sequentially access a number of rows as a subset of the table by matching some selection criteria. The ability to update and delete a single row via the key fields as well as the ability to add new rows to a table were also identified. These basic functional descriptions were deemed general enough to be implemented in most any database, from a simple RMS indexed file system to a complex relational database management package, providing a consistent definition for accessing all tables in the database. This generic approach to database access made the implementation of the functions a perfect candidate for automation.

Perhaps the most important motivation came as we began the task of implementing these database access routines for each table once DEC's Rdb had been selected as our database management package. It was immediately apparent that the code developed was functionally identical for each table in the database with the names of the table and its fields being the only variables from one set of access routines to the next.

### The Source-Code Generator

The functional interface described by the system design documents was implemented using FORTRAN record structures (figure 1) as the means for passing data back and forth between the database access routines and the calling applications. The record definition for each file is stored in a simple file which is then included into any function or routine which needs these definitions. In this way consistency is maintained throughout the software.

As a first step toward automation a functional template was developed. The initial process of coding access routines for new tables was at that time a simple, repetitive cut and paste process. In time, the automated code generator was seen as a more elegant solution to this inefficient use of programmer time.

The code generator was designed to accept the name of the file in which the FORTRAN record definition for a specific table was stored. The generator also accepts a table name which is used as the Rdb table name within the generated database access routines. In addition to reading the FORTRAN record definition file, the generic template (figure 2) is also read by the generator which simply substitutes field and table names into the template at the specified points thus automatically generating a set of access routines for the specified table (figure 3). This processing is made possible in part due to the fact that the applications programmers and system designers have been very regular and consistent in their construction of the FORTRAN record definition files. This makes the record definition files very easy to parse. Field names in the database tables are conveniently defined identically to those in the FORTRAN record definitions in most all cases, the notable exception being arrays. Arrays are prevalent in the IMIS record definitions as they are very appropriate data structures for much of the data being stored. One of the automating features of the code generator is the "unrolling" of arrays for database storage, and the reconstitution of arrays on retrieval. In this way, the concept of arrays are transparently maintained in the application programs and the FORTRAN record definitions. (The use of arrays however, eliminates the shorthand method allowed by the precompilers of correlating a FORTRAN record definition with an identical Rdb table definition through the use of the `A.* = B.*` convention for specifying a complete record transfer from a database record B to a FORTRAN record A).

The original code generator simply read the FORTRAN record definition and the access routine template file and produced FORTRAN source-code with embedded Rdb data manipulation commands ready for compilation. An early enhancement to the code generator was to output an additional file which contained the Rdb data definition commands for the definition of the table as well as the table's primary index. The ability to create a data definition file is due to the method we have chosen for field definition within the database. In the IMIS database, all fields within a table are "based on" simple global field definitions (figure 5). This is practical because the IMIS system is intended as the only database access method and all data integrity and validation checks are made by IMIS. By basing all fields on global definitions, the output of the data definition file by the code generator is simply a matter of determining the type and size of a variable and then basing it on the proper global definition. Thus creation of a new table by application programmers has become a simple process whereby they design the table using a FORTRAN record structure, run the code generator, execute the data definition file (figure 4), and compile the access routines. This has allowed an iterative development and testing process as new tables are designed and added to the database.

### Advantages to Using the Generator

The code generator provides a number of significant benefits for the development of the IMIS system. The most important advantage from a programmer's point of view must certainly be the 30,000 lines of RFO source code it has generated as well as the 2,000 lines of data definition statements. The tedious cut and paste method of code development was replaced with a quicker, automatic method.

Reliability and correctness were also greatly improved as the possibility of accidentally introducing coding or naming errors was eliminated.

Application programmers have been very accepting of this method of database access as it has insulated them from the actual database software itself by providing a generic set of database access functions with which they are able to satisfy the great majority of their database access needs. They are able to define, develop, and test new tables in the database without the slightest knowledge of the workings of the underlying database and in this case with little or no Rdb expertise.

### Future Enhancements (and Current Limitations)

A number of the current features of the code generator are poised for future enhancements which would further automate the process of code development and improve the functionality of the code which is generated.

The IMIS project uses DEC's MMS package to help "make" the entire IMIS system and maintain consistency and currency among the various elements of the software. Currently the MMS specifications are designed to invoke the code generator automatically if the access routine template has changed or the FORTRAN record definition has been altered. This has provided a powerful means for further automating the development of the access routines. A significant enhancement to this process would be to intelligently determine what changes had been made to the FORTRAN record definition and rather than output the data definition statements for defining the entire table, which is obviously not very useful once the table is already defined, output the data definitions necessary to make just the changes. These changes could then be automatically effected using MMS. The process of tuning the contents of tables has been a continuing part of the development of IMIS and this enhancement would give the application programmers complete and automatic control of the database table definitions by simply altering their FORTRAN include files.

The capability to sequentially read a qualified set of rows from a single table is currently implemented using an undocumented form of the "with" clause which does not appear to function with all of the precompilers. Ordinarily the precompilers reject clauses where both objects on either side of a boolean operator are "host" variables i.e.

```
with fortran_var1 > fortran_var2
```

However, through some experimentation we have found that substituting a literal or constant for one of the host variables is acceptable to the FORTRAN precompiler such that:

```
with fortran_var1 > 0
```

will work properly. The need to qualify the contents of a table dynamically at run-time is accomplished by creating a "with" clause which contains every field in the record and checks to see whether the field is equal to the value of the record in the database, or whether the field is set to some predefined wildcard value as in:

```
for p in personnel with
   (p.employee_number = employee_number or
    employee_number = 0) and
   (p.last_name = last_name or
    last_name = ' ') and ...
```

The "or" clause allows only the fields specified by the user via a form to influence the query since other fields will be set to the predefined wildcard value making the right half of the "or" expression true. There are at least two drawbacks to this method. First, we have found experimentally that the query will never use any indices even if an index exists for precisely the fields being used. In fact, if an index over all the fields of the table exists, the query will still never use the index. This is unfortunate and is probably due to our non-standard method of query qualification. Second, it is not elegant and is not extensible. It is a working brute-force method of creating dynamic queries.

There are two alternatives to our dynamic query facility, both of which leave much to be desired. Callable RDO is an alternative but only if we are willing to manage concurrent queries with embedded RDO and callable RDO, and to pay an unacceptable performance penalty. The second alternative would be to generate DSRI calls. We are constrained however, by the fact that the software must be easily maintainable by the client at the end of the implementation phase of the project. Furthermore, developing the capability to construct dynamic DSRI compatible queries is seen as too much effort for too little return when in fact the dynamic query capability is very useful but not central to the operation of IMIS.

Although our current method has been found to ignore the existence of indices, its implementation is simple and its performance acceptable under most circumstances. One would hope that the capability to construct run-time queries would be enhanced under future versions of Rdb, as this is one of Rdb's weaker points when compared with other relational database management systems. Many system implementations stretch databases to their limits and the ability to construct intelligent queries at run-time is of major importance. Having this capability would also allow us to enhance our record selection process by adding relational operators, specifying ranges, handling pattern matching, etc.

One final general enhancement to the source-code generator would be the generation of SQL compatible data manipulation statements for transportability across relational database management systems. This would be a very simple enhancement to make since it would require little more than a reworking of the access routine template.

### Conclusion

The Rdb source-code generator has proven to be a unique and valuable tool for our development environment. It has allowed application developers with little or no database

expertise to design, prototype, and implement database tables and applications which access the database quickly and completely.

The generator and its accompanying template provide a central focus for maintenance and enhancement of some 30,000 lines of generated source code. The template itself has been through numerous revisions and enhancements such that the 30,000 lines of finished source-code reflect only a fraction of the number of lines actually generated throughout the useful lifetime of the source-code generator.

Finally, the modular nature and generic capability of the source-code generator will allow us to use it in the prototyping and implementation of similar databases in the future. While the source-code generator began its life as a simple solution to a tedious problem, the concept has proven valuable and general enough that it has evolved into a more general purpose and generally useful software development tool tailored to our development environment.

**Figure 1:** FORTRAN Record Definition File

```
c+++ production.inc +++

      structure   /production/
      character*6    shift_date    !*
      character*1    shift_code    !*
      integer        line_number   !*
      integer        parts_in
      integer        product_out
      end structure

c--- production.inc ---
```

**Figure 2:** Access Routine Template Fragment

```
cccccccccccccccccccccccccccccccccccccccccccccccccccccccc
         entry read_key_@REC_NAME(@REC_NAME,istatus)
cccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c Initialize the status to our own system defined error
c   for missing data.  If no record matches the key fields,
c   this status will be returned.
         istatus = record_not_found

c Copy the key fields from the structure to local
c   variables for use in the query.
         @COPY_KEY

c Get any record in the database with the given key.
&RDB&    for r in @TABLE_NAME with
            @KEY_LIST
c Read the database record into the record structure.
&RDB&         get
                  @COPY_TO_RECORD
&RDB&         end_get

c Set the status to TRUE indicating a successful read
              istatus = .true.

&RDB&    end_for

         return
```

**Figure 3:** Ready to Precompile Source-Code

```
cccccccccccccccccccccccccccccccccccccccccccccccccccc
         entry read_key_production(production,istatus)
cccccccccccccccccccccccccccccccccccccccccccccccccccc
c Initialize the status to our own system defined error
c for missing data.  If no record matches the key fields,
c this status will be returned.
         istatus = record_not_found

c Copy the key fields from the structure to local
c   variables for use in the query.
         shift_date = production.shift_date
         shift_code = production.shift_code
         line_number = production.line_number

c Get any record in the database with the given key.
&RDB&    for r in production_data with
            r.shift_date eq shift_date and
            r.shift_code eq shift_code and
            r.line_number eq line_number
c Read the database record into the record structure.
&RDB&        get
&RDB&            production.parts_in = r.parts_in;
&RDB&            production.product_out = r.product_out;
&RDB&        end_get

c Set the status to TRUE indicating a successful read
             istatus = .true.

&RDB&    end_for

         return
```

**Figure 4:** Definition for Relation and Primary Index

```
define relation production_data.
    shift_date     based on chr_6.
    shift_code     based on chr_1.
    line_number    based on standard_integer.
    parts_in       based on standard_integer.
    product_out    based on standard_integer.
end production_data relation.

define index production_index for production_data
          duplicates are not allowed.
    shift_date.
    shift_code.
    line_number.
end production_index index.
```

**Figure 5:** Global Field Definitions

```
define field chr_1 datatype is text size is 1
     missing value "<XAO>".

define field chr_6 datatype is text size is 6
     missing vaule "<XAO>".

define field standard_integer datatype is signed longword
     scale 0 missing vaule -2147483646.
```

# DATATRIEVE SIG

Using Datatrieve with VAX-DBMS

Alan H. Beer
ASK Computer Systems
Los Altos CA 94022

Abstract

The Datatrieve interface to DBMS is flexible and very powerful. This paper reviews DBMS structures used in database design and the DTR structures for accessing DBMS data and metadata. Examples illustrate using a database through DTR from READY to FINISH with emphasis on DBMS-specific keywords. DBMS data is merged with data from non-DBMS sources and a RESTRUCTURE from DBMS to RMS is demonstrated.

## I Introduction

DBMS is DIGITAL's network database product. With proper administration, DBMS is suitable for applications requiring large databases with quick on-line retrieval.

Datatrieve, DEC's report writer and small-database manager, has a complete interface to DBMS which allows access to existing databases for query and update. Datatrieve is a useful tool for

- ad-hoc formatted report generation
- color graphics display generation
- quickly interfacing DBMS databases with
  - VAX FMS or TDMS form utilities
  - the VMS SORT utility
  - Rdb databases and RMS files
  - other DBMS databases

## II Structural Considerations

### II.A The DBMS Database

#### II.A.1 Metadata - the SCHEMA

A DBMS database is logically described in its SCHEMA. The SCHEMA is written in VAX DDL and compiled into and maintained in the CDD. In the SCHEMA are named AREAs (data files), RECORDs with data ITEMs, and SETs (pointers) relating different RECORDs.
The SUBSCHEMA is a mask over the SCHEMA, restricting the users access to named RECORDs, ITEMs and SETs. In this respect it is similar to a VIEW DOMAIN.

All DBMS applications access databases thru a SUBSCHEMA. A default SUBSCHEMA is generated when a SCHEMA is first compiled.

##### II.A.1.a RECORDs and ITEMs

RECORDs describe logical entities. Data ITEMS in RECORDs are attributes of the entity -- quantities, measurements, descriptions.

Each individual of a logical entity type is stored as an occurrence of the RECORD of that type. If the RECORD described EMPLOYEES and there were 200 employees to be listed, there would be 200 occurrences of the EMPLOYEES record, each with its own ITEM values.

##### II.A.1.b SETs

SETs are pointers from RECORDs to related RECORDs. DBMS maintains these internally. While the connection they establish is meaningful to the user, the actual pointer values are meaningful only to DBMS.

Fig I - Sample SCHEMA DDL

```
* THIS IS THE USGEOG DBMS SCHEMA.
SCHEMA IS USGEOG

* AREA DEFINITIONS

AREA NAME IS GEOG

* RECORD DEFINITIONS

RECORD NAME IS STATES
  WITHIN GEOG
    ITEM STNAME            TYPE IS CHARACTER 12
    ITEM STCODE            TYPE IS CHARACTER 2
    ITEM STPOP             TYPE IS SIGNED LONGWORD

RECORD NAME IS CITIES
  WITHIN GEOG
    ITEM CTNAME            TYPE IS CHARACTER 15
    ITEM CTPOP             TYPE IS SIGNED LONGWORD

* SET DEFINITIONS

SET NAME IS STSORTSET
   OWNER IS SYSTEM
  MEMBER IS STATES
        INSERTION IS AUTOMATIC
        RETENTION IS FIXED
            ORDER IS SORTED BY
        ASCENDING    STNAME
          DUPLICATES NOT ALLOWED

SET NAME IS STCITYSET
   OWNER IS STATES
  MEMBER IS CITIES
        INSERTION IS AUTOMATIC
        RETENTION IS FIXED
```

Chain SETs - SETs can describe relationships between RECORDs of different types when those differences are innate to the entity the RECORD describes. Such relationships might include ownership (people to things), residency (states to people), products (companies to merchandise), etc.

A SET relating cities to their home states would have the RECORD STATES as OWNER and the record CITIES as MEMBER. An occurrence of the SET might relate Tempe, Tuscon and Phoenix to Arizona.

Fig II - Sample SUBSCHEMA DDL

```
* CDD path to schema is "_CDD$TOP.ALBEER.DBMS.USGEOG"
*-----------------------------------------------------

SUBSCHEMA NAME IS USGEOGDTR FOR USGEOG SCHEMA

REALM GEOG
    IS GEOG

RECORD NAME IS STATES
    ITEM STNAME TYPE IS CHARACTER 12
        QUERY_HEADER IS "STATE"
    ITEM STCODE TYPE IS CHARACTER 2
        QUERY_HEADER IS "STATE" "CODE"
    ITEM STPOP  TYPE IS SIGNED LONGWORD
        QUERY_HEADER IS "STATE" "POPULATION"
         EDIT_STRING IS "ZZ,ZZZ,Z99"

RECORD NAME IS CITIES
    ITEM CTNAME TYPE IS CHARACTER 15
        QUERY_HEADER IS "CITY"
    ITEM CTPOP  TYPE IS SIGNED LONGWORD
        QUERY_HEADER IS "CITY" "POPULATION"
         EDIT_STRING IS "ZZ,ZZZ,Z99"

SET NAME IS STSORTSET

SET NAME IS STCITYSET
```

System-owned SETs - SETs can also determine sequence (sort order) or placement (using hashing) of a single type of RECORD within the database. These sets are "System owned" and have one occurrence in the entire database. All the sequenced or placed records are members.

A sorted SET with STATES as MEMBERs would present any user "walking" that set with all stored STATES in pre-sorted order. All STATES are sequenced within the single occurrence of that SET.

II.A.2 Database Implementation

When a database is created from the SCHEMA, DBO creates a root file and area files. The root file serves as the database header, with pointers to and status of all area files. A copy of the SCHEMA and SUBSCHEMAS is incorporated in the root.

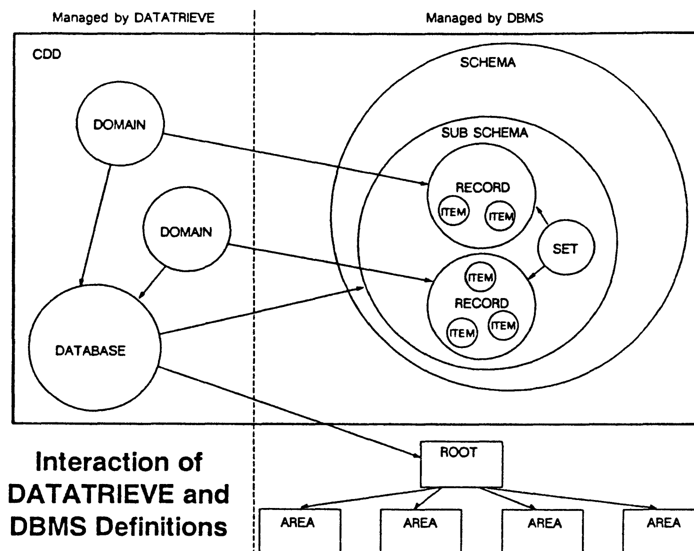
The various area files hold the actual data for which the database was created. The areas are READYed as needed for data retrieval and update.

II.B The Datatrieve - DBMS Interface

II.B.1 Datatrieve Structures

The Datatrieve DATABASE structure references a DBMS database and the SCHEMA from which the database was created. The DATABASE and SCHEMA may be kept in the same or different CDD nodes.

Datatrieve learns the structure of the DBMS database by retrieving the SCHEMA and SUBSCHEMA named in the DATABASE. Datatrieve then binds the root file of the database specified. The Database Control System (DBCS) then processes the Datatrieve DBMS queries.




Interaction of DATATRIEVE and DBMS Definitions

Datatrieve users can optionally define DOMAINs to reference individual records in a DATABASE. These domains may link records to FMS or TDMS forms.

Fig III - DATATRIEVE Structures for Accessing DBMS

```
DEFINE DATABASE US_GEOG USING SUBSCHEMA USGEOGDTR OF
        SCHEMA CDD$SCHEMA.USGEOG ON DB$DIR:USGEOG.ROO;

DEFINE DOMAIN CITIES USING CITIES OF DATABASE US_GEOG;

DEFINE DOMAIN STATES USING STATES OF DATABASE US_GEOG;
```

II.B.2 Report Format Items

SUBSCHEMAs can specify query headers, edit strings and query names for data ITEMS. All these are used by Datatrieve if defined in the SUBSCHEMA.

III Accessing a Database

III.A READYing with DBMS

Databases can be READYed from Datatrieve RECORD by RECORD or all at once. The command

```
DTR> READY USGEOG
```

will ready all RECORDs in the USGEOG database for SHARED READ. The command:

```
DTR> READY USGEOG USING STATES READ, CITIES WRITE
```

will allow adding CITIES to existing STATES but prevent adding STATES. In this example, STATES and CITIES are the RECORDs, not the DOMAINs.

When RECORD DOMAINs have been defined, the command

```
DTR> READY STATES
```

provides access only to STATES and STSORTSET (see figure above).

VIEWs can only be defined based on other DOMAINs. As a result, DBMS VIEW definitions require the use of RECORD DOMAINs. A VIEW definition naming a DATABASE will not READY.

### III.B FINDing COLLECTIONs

Most Datatrieve user training makes liberal use of the FIND command. FIND will create a COLLECTION of RECORDs to be accessed in some way. Streams can be formed from COLLECTIONs as they can be from files or databases.

Unfortunately, when a COLLECTION is formed, locks are placed on all records within the COLLECTION. KEEPLISTs are maintained and system resources are tied up. This is known to slow down all users on a VAX, particularly other users of the database.

COLLECTIONs are not needed for simple streaming of records for reports or modifications. Locking can be minimized by incorporating RSEs in FOR or REPORT statements. If the use of FINDs is unavoidable, try to use the DROP statement to unlock individual RECORDs no longer needed.

### III.C Committing Transactions and Rolling Back

All changes to the database made during a Datatrieve session are volatile until the user issues a COMMIT or ROLLBACK. These statements affect all READY databases and do not accept arguments.

COMMIT will make the STOREs, MODIFYes and ERASEs on all READY databases permanent. It also retains any COLLECTIONs the user may have.

ROLLBACK will leave the database as it was when the session began (or since the last COMMIT). ROLLBACK releases all COLLECTIONs and leaves the user with the database freshly READYed.

When a user FINISHes their last READY database, either explicitly or by EXITing Datatrieve, all transactions that have not been ROLLBACKed are COMMITted automatically. This includes those on databases which were FINISHed earlier in the session.

## IV SET Relationships through Datatrieve

The power of Datatrieve as a relational database package is the ability to gather records of different types that contain related data. For these ends Datatrieve logically links records in inner lists or in CROSSes.

While we discuss ways to use the DBMS relationships in conditional RSEs, remember that Boolean conditional RSEs are still an available option. SETs optimize retrieval of related records, but simple value comparisons will still prove useful when

- no existing SET suits your need, - or -
- combining database data with data from outside sources.

### IV.A Recognition of SETs

As DBMS RECORDs are readied in Datatrieve, all SETs with ready OWNERs and MEMBERs are recognized. System owned SETs are recognized when the MEMBERs are readied.

Datatrieve will fail to recognize a SET if no occurrences of that SET exist in the database.

Figure IV - Sets Recognized as Records are Readied

```
DTR> READY STATES
DTR> SHOW SETS

Set: STSORTSET
       Member: STATES, automatic fixed

DTR> READY CITIES
DTR> SHOW SETS

Set: STSORTSET
       Member: STATES, automatic fixed

Set: STCITYSET
       Owner: STATES
       Member: CITIES, automatic fixed

DTR> FINISH
DTR> SHOW SETS
No sets are currently useful.
```

### IV.B Walking SETS - Retrieval

#### IV.B.1 Formatting Inner Lists

A one-to-many relationship is often represented on paper as a list of the many subordinate to the one. All cities found within their state boundaries would be formatted:

```
ALABAMA
    MOBILE
    MONTGOMERY
    SELMA
ALASKA
    ANCHORAGE
    FAIRBANKS
    JUNEAU
  :
      :
```

While we read the city and state names, the relationships are implied by the position of the inner lists within the main list.

This list would be produced from our database with the command

```
DTR> FOR STATES PRINT STNAME, ALL CTNAME OF CITIES
         WITHIN STCITYSET
```

The "ALL printlist OF domain" syntax is vanilla Datatrieve. The "WITHIN set" conditional expression in the RSE is the only DBMS twist.

Interchangeable Keywords

> When specifying SET conditional RSEs, the keywords OWNER and MEMBER specify the direction of the relationship within the SET. The keyword WITHIN can be substituted for either keyword OWNER or MEMBER.

IV.B.2 Flat Records from CROSSes

Crossing records in Datatrieve is, in effect, taking several related records of different types and making one long record out of them. One record is found for every permutation of source records meeting any conditions in the CROSS RSE. CROSSed records can be sorted on any field(s) within the resulting combination record.

A CROSS WITHIN a SET retrieves OWNER-MEMBER combinations. CITIES without OWNER STATES (like Washington, D.C) and STATES without cities would not be included in the resulting stream.

The "flatness" is illustrated by printing crossed records. Note the repitition of data where a STATES record completed more then one CROSS.

Figure V - The Flatness of CROSSed RECORDs

```
DTR> SORT BY CTNAME
DTR> PRINT
No record selected, printing whole collection.

               STATE   STATE                          CITY
   STATE       CODE  POPULATION        CITY        POPULATION

CALIFORNIA      CA    23,668,562 ANAHEIM              221,847
CALIFORNIA      CA    23,668,562 BAKERSFIELD          105,611
NORTH DAKOTA    ND       652,717 BISMARCK              44,485
MASSACHUSETT    MA     5,737,037 BOSTON               562,994
ILLINOIS        IL    11,426,518 DECATUR               94,081
     :          :         :          :                   :
```

IV.B.3 Saving SET-walks in VIEW DOMAINs

VIEW DOMAINs can be built from DBMS DOMAINs as readily as from RMS DOMAINs. The VIEW provides a permanent specification of CROSSes and hierarchies for retrieval. The OCCURS FOR clause can elaborate a CROSS:

```
DFN> 01 BIGCITIES OCCURS FOR CITIES CROSS STATES
         WITHIN STCITYSET WITH CTPOP > 200000.
```

OCCURS at higher levels can use SET hierarchy:

```
DFN> 01   BIGSTATES OCCURS FOR STATES WITH STPOP >
          5000000.
DFN>  03  STATES FROM STATES.
DFN>  03  BCITIES OCCURS FOR CITIES WITHIN STCITYSET.
DFN>   05 CITIES FROM CITIES.
```

The VIEW can also specify TDMS or FMS forms, edit strings, etc.

Another use for VIEWs is establishing links between a DBMS database and other sources of data. An agricultural database can be connected to a weather database through a CROSS kept permanent in a VIEW:

```
DTR> DEFINE DOMAIN RAINGROW OF
         ASTATES,...,MSTATES,...
DFN> 01 GROW OCCURS FOR ASTATES CROSS MSTATES OVER
         STNAME.
DFN>    :
```

The source DOMAINS could access various DBMS or Rdb databases or RMS files.

IV.C CONNECTing RECORD occurrences to SET Occurrences

IV.C.1 Automatic Insertion

When storing MEMBERs of an AUTOMATIC chain SET like STCITYSET, the OWNER-to-be must be current. This currency is established with the SELECT statement or a FOR structure. Whichever of the OWNER type RECORDs is current will become the OWNER of that MEMBER occurrence when it is stored.

No currency is needed for System-owned SETs. DBMS determines placement of these RECORD occurrences within sorted or hashed sets.

IV.C.2 Rearranging SET Memberships

Once stored, a MEMBER-type-RECORDoccurrence of SETs that are not FIXED can be RECONNECTed to a different OWNER--type-RECORDoccurrence within that SET. If Michigan were to annex Cleveland we might try

```
DTR> FOR STATES WITH STCODE = "MI"
CON> FOR CITIES WITH CTNAME = "CLEVELAND"
CON> RECONNECT CITIES TO STATES.STCITYSET
```

In similar actions we could CONNECT Washington, D.C. to Georgia or DISCONNECT San Francisco from California. Frightening...

V Pseudo-unload of a DBMS Database

Datatrieve can be used to copy DBMS data to machine-usable ASCII files. The RESTUCTURE feature copies data from domain to domain, matching field names and converting data formats.

1. A view of DBMS records is defined, if needed

```
DEFINE DOMAIN CITYDBMS FROM CITIES, STATES
01 CITY OCCURS FOR CITIES SORTED BY CTNAME.
   03 CTNAME FROM CITIES.
   03 STCODE FROM STATES OWNER STCITYSET.
   03 CTPOP FROM CITIES.  ;
```

2. An RMS record is defined with the format desired.

```
DEFINE RECORD CITYRMSREC
01 CITYDATAREC.
      !  ASCII, 35 byte records
   03 CTNAME PIC X(15).
   03 STCODE PIC X(2).
   03 CTPOP PIC 9(8).  ;
      !  EXTERNAL format
```

3. A domain is defined to use the RMS record

```
DEFINE DOMAIN CITYRMS USING CITYRMSREC ON CITY.DAT;
```

4. A file is created to recieve the data

```
DEFINE FILE FOR CITYRMS
```

5. The view is made READY for READ

```
READY CITYDBMS
```

6. The new domain is made READY for WRITE

```
READY CITYRMS WRITE
```

7. Restructure

```
CITYRMS = CITYDBMS
```

Data is converted and moved field for matching-name field.

This technique is not limited to creating ASCII data files; subsets of DBMS data can be extracted for any software application.

# Managing ALL-IN-1 with Datatrieve

Bart Z. Lederman
World Communications Inc.
New York, NY 10004-2464

ABSTRACT

This session will illustrate some of the uses for Datatrieve in managing ALL-IN-1, include the ALL-IN-1 environment and processing the logging file.

All of the Datatrieve definitions used in this session, and many others, may be obtained through the DECUS Library, on the Datatrieve/4GL SIG Library Tape collection, or the VMS SIG Tape Collection.

In addition to the data that ALL-IN-1 and WPS Word Processing manage for you, these products create and use their own database files in their own operation. While they also come with their own utilities and commands for the examination and maintenance of those files, there are times when Datatrieve allows faster, easier, or more versatile access or management of this information.

I will present a number of such uses, starting with the document database (the DOCDB.DAT file) which contains the list of documents for each user of ALL-IN-1 and/or WPS-Plus. One example is shown here:

```
DOCUMENT       : DEC
REFNUM         : 999886
TITLE_HASH     : DRAFTR
FILENAME       : [.DOC4]ZRNWAXDCF.WPL
DAF_POINTER    : P
TITLE          : draft review
AUTHOR         : Bart Z. Lederman
TYPE           : DOCUMENT
UNUSED_SETUP   : NONE
FORMAT         :
KEYWORDS       :
CREATED        : 28-Jul-1987
MODIFIED       : 24-Aug-1987
MAIL_ORIG      :
F3             :
MAIL_STATUS    :
F2             :
DOCNUM         :    114
DELETABLE      : Y
MODIFIABLE     : Y
CREATED        : 28-Jul-1987
CREATED_TIME   : 11:17:13.45
MODIFIED       : 24-Aug-1987
MODIFIED_TIME  : 10:52:24.59
V1TYPE         :
DSAB           : WPSPLUS
F6             :
```

## Why access this other than with ALL-IN-1?

The VMS file name can be obtained, and correlated to the files on the system. Uses for this include rebuilding DOCDB.DAT if it becomes corrupted, and if VMS file expiration dates are being used to purge old files and they must be correlated with the appropriate documents.

It should be noted that if your DOCDB.DAT becomes corrupted, you should

first try to rebuild it using the verification procedures supplied with ALL-IN-1, accessible to the system manager. I have found that with V2.0 they did not work too well, but with V2.2 they have done a good job of re-correlating files with documents. As a general rule, you should always let ALL-IN-1 try to fix itself first, before manipulating the files directly yourself. And if you do ever have to modify DOCDB.DAT, run the verification procedure afterwards.

Another use is to obtain faster reports, or reports different from those supplied with ALL-IN-1.

```
DELETE DOCDB_REPORT;
REDEFINE PROCEDURE DOCDB_REPORT
!
! Get a listing of a users' documents with corresponding VMS file names.
!
! B. Z. Lederman
!
REPORT DOCDB ON *."file specification"
SET COLUMNS_PAGE = 132
SET LINES_PAGE = 42
PRINT REFNUM, FOLDER, TITLE USING T(48), FILENAME USING T(24)
END_REPORT
END_PROCEDURE
```

This report format looks like this:

```
REFNUM    FOLDER                  TITLE               FILENAME

999998  DICTIONARIES   PERSONAL                    USER$DEVICE:[LEDERMAN]PERSONAL.LGP
999874  EXAMINER       install_process             [.DOC6]ZRQJAQVXO.WPL
999852  OUTBOX         accruals blocks only        OA$SHARE1:ZRUCATMDI.WPL
999877  PICTURES       EIA null modem/Ethernet     [.DOC3]ZRPTAYCQL.WPL
999872  WPS            rendition                   [.DOC8]ZRRMARIIS.WPL
```

Reporting with Datatrieve allows locating documents by directory or disk; locating documents by creation or modification dates; locating documents on multiple fields (a word contained within a title and an author name); reporting all documents by author, and other formats not supplied by ALL-IN-1.

A Datatrieve report procedure can be put into a form and included as an ALL-IN-1 application so users can type one command obtain a complete index of their documents on their default printer.

**Shared definitions and system management.**

Rather than define one for every user, you can take advantage of the logical name created during login to set the correct directory to access:

```
DEFINE DOMAIN DOCDB USING DOCDB_RECORD
    ON OAUSER:DOCDB.DAT;
```

You might also want to define one domain just for the postmaster:

```
DEFINE DOMAIN DOCDB_POSTMASTER USING
     DOCDB_REC ON
SYS$SYSDEVICE:[ALLIN1.POSTMASTE]DOCDB.DAT
```

This allows you to look at the POSTMASTER's database without logging into VMS POSTMASTER account: the release notes for ALL-IN-1 V2.1 warn against logging into the POSTMASTER account as it can lock out the message router and cause mail messages not to be delivered.

**The user profile.**

The data looks like this:

```
USER          : LEDERMAN
VMSNAM        : LEDERMAN
FULNAM        : Bart Z. Lederman
TITLE         : Project Engineer
DEPART        : Advanced Systems Dev
STATUS        :
PASWRD        :
PHONE         : 212-607-2657
RESERVED      :
DCL           : Y
SUP           : Y
ERR           : Y
CMD           : Y
SRC           : Y
CPHD          : Y
LOG           : Y
MULTI_NODE    : Y
RSVD_FOR_TCS  :
ADDR1         : 2572 E. 22nd Street
ADDR2         : Sheepshead Bay, NY
ADDR3         :
ADDR4         : (718) 743-9593
ZIPCOD        : 11235-2504
NOTICE        : Y
BATCH_NOT     : Y
PRINT_NOT     : Y
MAIL_READ_REC : Y
TICKLER       : Y
ACTITEM       : Y
DIRECTORY     : USER$DEVICE:[LEDERMAN.OA]
FORMLIB       :
INIT_FORM     : MAIN
EDITOR        : WPSPLUS
PRINTER       : LN03$PRINT
NODE          :
PRINT_PORT    : Y
TERM_MODE     : S
MAIL_FORWARD  :
MAIL_REPLY    :
MAIL_MENU     : EMC
MAIDES        : ALLIN1
CALTIMEING    : 00:15
SETUSR        : Y
YESDAYS       : N
STARTD        : 2
ENDD          : 5
STARTH        : 08:00A
ENDH          : 4:30P
MEALS         : 12:00P
MEALE         : 01:00P
CALDAY        : 2
UFLAG1        :
UFLAG2        :
UFLAG3        :
UFLAG4        :
UFLAG5        :
UFLAG6        :
UFLAG7        :
UFLAG8        :
UFLAG9        :
UFLAG10       :
CLASS         :
LANGUAGE      : ENGLISH
END           :
```

**Processing multiple entries.**

Since ALL-IN-1 gives you access to these fields, why would you use Datatrieve? One answer is that ALL-IN-1 management is oriented to processing one user at a time. For example, If you want to find out which users have DCL access enabled you have to go through several menus and screens to get an index of users, write down their names, then examine them all one at a time to find the ones with DCL. With Datatrieve, you can ready the PROFILE domain and enter the command 'PRINT PROFILE WITH DCL = "Y"' to find all such users. Similarly, operations on large numbers of users such as turning DCL access, turning logging on or off for everyone, finding users whose accounts point to certain disks and/or re-assigning them to other disks, or finding which users are assigned to which default printer, are easier in Datatrieve than in ALL-IN-1 as presently supplied. You can also use Datatrieve to produce formatted reports of all users or groups of users, and you can select which information fields are printed in that report.

```
DTR> for profile print user, directory
```

| USER | DIRECTORY |
|---|---|
| SMITH | USER$DEVICE:[SMITH.OA] |
| BROWN | USER$DEVICE:[BROWN.WPSPLUS] |
| IVP | SYS$SYSDEVICE:[ALLIN1.IVPUSER] |
| LEDERMAN | USER$DEVICE:[LEDERMAN.OA] |
| MANAGER | SYS$SYSDEVICE:[ALLIN1.MGR] |
| POSTMASTER | SYS$SYSDEVICE:[ALLIN1.POSTMAST |

Because ALL-IN-1 manipulates several files for user profiles, it is not a good idea to add or delete user profiles using Datatrieve, though it can be done in emergencies.

**The network profile.**

OA$DATA:NETWORK.DAT also contains information.

```
USER_NAME        : LEDERMAN
NODE             : SYS31
LAST_UPDATE      :  9-Jan-1987
UPDATE_TIME      : 15:10:15.81
FULL_NAM         : Bart Z. Lederman
TITLE            : Project Engineer
DEPARTMENT       : Advanced Systems Dev
TELEPHONE        : 212-607-2657
ADDR1            : 2572 E. 22nd Street
ADDR2            : Sheepshead Bay, NY
ADDR3            :
ADDR4            :
ZIPCOD           : 11235-2504
NETWORK_ADDRESS  : LEDERMAN AT A1 AT
                   SYS31
TIMESTAMP        :
M_NODE           : Y
DELETED          : N
```

**Cleaning up after ALL-IN-1.**

Removing a user from ALL-IN-1 does not remove the entry in this file for that user, it just marks the user as "DELETED". There may be cases where you want to know that a user used to be on a system (or is located elsewhere: see the next section), but when a user is off my system I want them to be completely off of the system, and when a user is deleted I want the entry to be removed. Datatrieve is a useful tool for cleaning up after ALL-IN-1.

**Copying users from the Corporate Telephone Directory?**

In a conversation I had with some users at the Symposium, I was asked if it would be possible to copy entries from the Corporate Telephone Directory to the network profile. Their configuration was that many users on many nodes were entered into the directory so people can find each other. They then also enter every user into the network profile, so when you send mail to someone it will be routed to them even if they are not on your node. They were doing this by hand, and wanted to know if it could be automated.

First, it is easy to read the telephone directory with Datatrieve. [I have not shown this here because it's not part of managing ALL-IN-1, but it has been published in the newsletter and is on the DTR/4GL SIG tapes and VAX SIG tapes. The telephone directory actually works better in Datatrieve than in ALL-IN-1.] I have some reservations about writing new data into the network profile: but as long as a backup copy of the un-modified file is made first I see no reason why the experiment should not be tried.

One precaution which should always be taken is to open all files SHARED so you won't accidentally lock out ALL-IN-1. An easy way is to issue a command similar to the following:

```
$ DEFINE DTR$READY_MODE "SHARED"
```

preferably in your LOGIN.COM file. This way, Datatrieve will by default open all files SHARED.

**Time Management.**

Here is a definition for OA$DATA:CALACCESS.DAT, the file which determines which users are allowed to access which other users' calendars.

```
REDEFINE RECORD AI1_CALACCESS_RECORD
01 AI1_CALACCESS_REC.
    10 GRANTUSER PIC X(30) EDIT_STRING T(16)
          QUERY_HEADER "User"/"Granting"/"Access".
    10 ACCUSER PIC X(30) EDIT_STRING T(16)
          QUERY_HEADER "Access"/"Given"/"To".
    10 READ PIC X QUERY_HEADER "Read"/"Your"/"Calendar".
    10 WRITE PIC X QUERY_HEADER "Schedule"/"for"/"You".
    10 PHONE PIC X QUERY_HEADER "".
;
```

| User Granting Access | Access Given To | Read Your Calendar | Schedule for You |
|---|---|---|---|
| ERSKINE | STEVE | Y | Y |
| ERSKINE | ELLIOTT | Y | Y |
| LEROY | REGGIE | Y | Y |
| LEROY | HOWIE | Y | N |
| LEROY | STEVE | Y | Y |
| RKELLY | VICKIC | Y | N |
| STEVE | ERSKINE | Y | N |
| VICKIC | BEATE | Y | N |
| VICKIC | IRENE | Y | Y |
| VICKIC | RKELLY | Y | N |

**Documenting use of data.**

The additional information placed in the record definition which describes what the fields are used for (which also automatically prints out as column headers when the data is accessed), and additional comments which can be stored in the Datatrieve definition, is good documentation as to what the data is and how it's used which is not always found in ALL-IN-1 FMS forms descriptions.

**More cleanup after ALL-IN-1.**

ALL-IN-1 did not delete users from this file when I removed their user profile, nor from the others shown below. Using Datatrieve, it is easy to locate and delete all records which have the name of a person who is no longer an ALL-IN-1 user in either the GRANTUSER or ACCUSER fields and thus clean up the file. I also find it easier to obtain a quick listing of all users in the CALACCESS file with Datatrieve than it is with ALL-IN-1.

OA$DATA:ATTENDEE.DAT contains the list of meetings and attendees.

```
NAME1   : IVP
DATE    :  1-Jan-2010
TIME    : 08:00
LENGTH  : 0001
NAME2   : NOT_A_REAL_NAME
FLAG    :
YES_NO  : YES
MESSAGE : I will be a little late.
    DATE    : 18-Nov-1858
    TIME    : 00:00
    LENGTH  :
    DATE    : 18-Nov-1858
    TIME    : 00:00
    LENGTH  :
    DATE    : 18-Nov-1858
    TIME    : 00:00
    LENGTH  :
    DATE    : 18-Nov-1858
TIME    : 00:00
    LENGTH  :
    DATE    : 18-Nov-1858
    TIME    : 00:00
    LENGTH  :
    DATE    : 18-Nov-1858
    TIME    : 00:00
    LENGTH  :
END     :
```

**Purge old users and meetings for better performance, and to save space.**

Removing a user from ALL-IN-1 does not remove that users' meetings from the data file, nor does there appear to be any mechanism to remove past meetings automatically: Datatrieve can be used to easily remove them. I have not found an easy way to get ALL-IN-1 to tell me which days (especially in the past) still have meetings assigned, and the tendency may be for users to simply leave all past meetings scheduled. Using date comparisons within Datatrieve makes it easy to find and delete all past meeting records, or a past meetings could be moved into a separate domain for record keeping.

OA$DATA:MEETING.DAT is the file which contains a list of meetings.

```
SCHEDULER : IVP
DATE      :  1-Jan-2010
TIME      : 08:00
LENGTH    : 0000
DATE      :  1-Jan-2010
TIME      : 21:00
LENGTH    : 000
PURPOSE   : To test Installation
            Verification Procedures
LOCATION  : Charlotte
PRIORITY  : A1
A         :
NAME2     :
B         :
```

As before, this domain contains records of obsolete meetings, meetings for persons no longer using the system, and IVP meetings that are scheduled for the future but really don't need to remain in the file once the Installation Verification Procedure finishes. Datatrieve is an easy way to remove them.

ATTENDEE and MEETING files should match: you shouldn't have meetings without attendees, and vice versa. Datatrieve can find records that match (and records that don't):

```
DTR> for meeting cross attendee over
     meeting_pointer -
CON> print scheduler, attendee_name,
     date
```

| SCHEDULER | ATTENDEE NAME | DATE |
|---|---|---|
| ERSKINE | RALPH | 29-Aug-1985 |
| ERSKINE | STEVE | 29-Aug-1985 |
| ERSKINE | IRENE | 30-Aug-1985 |
| HOWIE | STEVE | 30-Aug-1985 |
| IRENE | PATMC | 30-Aug-1985 |
| IVP | NOT_A_REAL_NAME | 1-Jan-2010 |
| IVP | NOT_A_REAL_NAME | 1-Jan-2010 |
| IVP | YHTALEOJ | 1-Jan-2010 |
| LEDERMAN | MANAGER | 15-Dec-1986 |

**Pending mail count.**

OA$DATA:PENDING.DAT contains the pending mail count.

| PENDING KEY | C | A | B | D | ENUM |
|---|---|---|---|---|---|
| FETCHER QUEUE | 0 | 4 | 0 | | 0 |
| MAIL JONES | 0 | 4 | 0 | | 34 |
| MAIL SMITH | 0 | 152 | 148 | | 2 |
| MAIL DEMO | 0 | 0 | 0 | | 0 |
| MAIL IVP | 0 | 0 | 0 | | 0 |
| MAIL MANAGER | 0 | 0 | 0 | | 0 |

Once there is a Datatrieve domain which accesses this data, it's easy to find out how many messages are waiting (without going into ALL-IN-1):

```
REDEFINE PROCEDURE PRINT_PENDING
!
! Find out how much ALL-IN-1 Electronic
! Mail is Pending
!
! B. Z. Lederman
!
READY AI1_PENDING SHARED
DECLARE KEY_FIELD PIC X(65).
!
! Get the pre-defined user name
! and make it into a retrieval key
!
KEY_FIELD =
  "MAIL " | FN$TRANS_LOG ("USER_NAME")
!
FOR AI1_PENDING WITH
    PENDING_KEY = KEY_FIELD
    PRINT "Pending Messages = ", ENUM(-)
!
FINISH
END_PROCEDURE
```

A logical name is used here so one procedure could be used by everybody in the system (invoked from SYLOGIN.COM for example) A complete description of this process has been published in the combined DECUS Newsletter.

**ALL-IN-1 usage and tuning: the logging file.**

ALL-IN-1 can generate a logging file, but doesn't do much with the information generated. A small portion of the raw data looks like this:

```
FACIL    MSG       PROC       SYS            SYS         ELAP
 ID      ID         ID        DATE           TIME        TIME   INPUT      FUNCTION      TEXT

 287  18842411  10329 19-May-1987 13:27:44.02    0 Function:  SET_MENU        DEFAULT
 287  18842411  10329 19-May-1987 13:27:44.13   11 Function:  FORM            MAIN
 287  18842411  10329 19-May-1987 13:27:44.24   21 Function:  OA$FORM_MENU
 287  18842411  10329 19-May-1987 13:27:45.04   65 Function:  GET             #CURDOC="$WPDOC"
 287  18842411  10329 19-May-1987 13:27:45.06   67 Function:  FORM            FC1/MORE=WP
 287  18842411  10329 19-May-1987 13:27:45.40   90 Function:  OA$FORM_MENU    /MORE=WP
 287  18842411  10329 19-May-1987 13:27:45.41   91 Function:  .               .IF @#CURDOC:6:30 N
                                                                              OR @ #URDOC:6:30 NE
                                                                              THEN CAB CURRENT
                                                                              @#CURDOC ELSE CAB C
 287  18842411  10329 19-May-1987 13:27:45.52   95 Function:  CAB             CURRENT @#CURDOC
 287  18842403  10329 19-May-1987 13:28:35.18  155 i{CR}
 287  18842411  10329 19-May-1987 13:28:35.19  156 Function:  OA$FLD_DONE
 287  18842411  10329 19-May-1987 13:28:35.21  158 Function:  GET             INDEX="Document"
 287  18842411  10329 19-May-1987 13:28:35.22  158 Function:  FORM            WPINDX/STYLE=CHOICE
 287  18842411  10329 19-May-1987 13:28:35.47  175 Function:  GET             #INPROMPT='0'
 287  18842411  10329 19-May-1987 13:28:35.48  176 Function:  GET             #goldAexit = 3
 287  18842411  10329 19-May-1987 13:28:35.49  177 Function:  DISPLAY         Enter search fields
                                                                              press RETURN, or EX
                                                                              SCREEN to return  t
```

Datatrieve can process the data to extract useful information: for example, FORM and DO script usage, including matching it up to the library where the form or script is stored. I include the procedure here to demonstrate an important point.

```
REDEFINE PROCEDURE AI1_NORMALIZE
!
! Process ALL-IN-1 logging file so that the form names are extracted
! and normalized.  This allows looking them up in a table to
! find out which library they are in, and to allow summation
! for statistics on use.
!
! B. Z. Lederman
!
DEFINE FILE FOR AI1_NORM
READY AI1_NORM WRITE
READY AI1_LOG
!
! need a few working variables
!
DECLARE A_FORM PIC X(24).
DECLARE B_FORM PIC X(24).
DECLARE E1 PIC 99 EDIT_STRING Z9.
DECLARE E2 PIC 99 EDIT_STRING Z9.
!
! Go through the logging file and pick out uses of forms and
! scrips
!
FOR AI1_LOG WITH FUNCTION = "FORM", "DO" BEGIN
!
```

```
! Now comes the fun part.  We want to extract only the form (or
! script) name and normalize it.
!
    E1 = 0                          ! initialize end of string
    E2 = 0                          !   position counters
    A_FORM = FN$UPCASE (FORM_NAME)   ! force upper case
    E1 = FN$STR_LOC (A_FORM, " ")    ! look for end of form name
    E2 = FN$STR_LOC (A_FORM, "/")    ! may have command attached
    IF E1 > 0 E1 = E1 - 1            ! want last character
    IF E2 > 0 E2 = E2 - 1            ! not search character
!
! take out the form name if it is not null: the form either ends
! with a null or space or with a slash if there was a command
! attached.
!
    IF ( (E2 > 0) AND ( (E2 < E1) OR (E1 = 0) ) ) THEN
        B_FORM = FN$STR_EXTRACT (A_FORM, 1, E2) ELSE
        B_FORM = FN$STR_EXTRACT (A_FORM, 1, E1)
!
! If we can find this form (script) in the domain table we
! created which lists the library each form is in, use that
! libraries' name, otherwise use a blank space.
!
    IF (B_FORM IN FORM_TABLE) THEN
        A_FORM = B_FORM VIA FORM_TABLE ELSE A_FORM = " "
!
! we now have nicely normalized data: store it.
!
    STORE AI1_NORM USING BEGIN
        FACIL_ID  = FACIL_ID
        MSG_ID    = MSG_ID
        PROC_ID   = PROC_ID
        SYS_DATE  = SYS_DATE
        ELAP_TIME = ELAP_TIME
        FUNCTION  = FUNCTION
        NAME      = B_FORM
        LIBRARY   = A_FORM
    END
END
END_PROCEDURE
```

The information, after processing, looks like this:

| FACIL ID | MSG ID | PROC ID | SYS DATE | SYS TIME | ELAP TIME | FUNCTION | NAME | LIBRARY |
|---|---|---|---|---|---|---|---|---|
| 287 | 18842411 | 10329 | 19-May-1987 | 13:27:44.13 | 11 | FORM | MAIN | MEMRES |
| 287 | 18842411 | 10329 | 19-May-1987 | 13:27:45.06 | 67 | FORM | FC1 | OAFORM |
| 287 | 18842411 | 10329 | 19-May-1987 | 13:28:35.22 | 158 | FORM | WPINDX | OAFORM |
| 287 | 18842411 | 10329 | 19-May-1987 | 13:28:51.74 | 292 | DO | WPLIST | OA$DO |
| 287 | 18842411 | 10329 | 19-May-1987 | 13:28:52.35 | 324 | FORM | OA$LIST | MEMRES |
| 287 | 18842411 | 10329 | 19-May-1987 | 13:29:01.10 | 433 | DO | WPDELETE | OA$DO |
| 287 | 18842411 | 10329 | 19-May-1987 | 13:29:08.76 | 486 | DO | FCDELFDR | OA$DO |
| 287 | 18842411 | 10329 | 19-May-1987 | 13:29:14.69 | 517 | FORM | WPINDX | OAFORM |
| 287 | 18842411 | 10329 | 19-May-1987 | 13:29:34.49 | 635 | FORM | WPINDX | OAFORM |
| 287 | 18842411 | 10329 | 19-May-1987 | 13:29:46.97 | 730 | DO | WPMDMP | OA$DO |
| 287 | 18842411 | 10329 | 19-May-1987 | 13:29:52.52 | 863 | FORM | WPINDX | OAFORM |

### Why is this done in Datatrieve?

A program in a "third generation" language such as COBOL or FORTRAN which processes this data would be much longer and more complicated, and it would take much longer to write and test it, than the Datatrieve procedure shown above. In addition, many "office environment" systems aren't going to have a traditional programming language available (or programmers to use them), but will have Datatrieve.

There is a lot of information which can be extracted from this data. For example: how many users access ALL-IN-1; when they use it; the sequence of commands entered; what features within ALL-IN-1 are being used; which FORMS and DO scripts are or are not being used; etc. Datatrieve allows quick ad-hoc querying and reporting of the data with much less work than "3rd generation" languages:

```
REDEFINE PROCEDURE AI1_NORM_RPT
!
! Report summarized use of forms and
! scripts
!
! B. Z. Lederman
!
READY AI1_NORM
REPORT AI1_NORM WITH LIBRARY NE " " -
    SORTED BY FUNCTION, NAME ON
    *."TT or file name"
AT BOTTOM OF NAME PRINT FUNCTION,
    SPACE 1, LIBRARY, SPACE 1, COUNT,
    SPACE 1, NAME
AT BOTTOM OF FUNCTION PRINT NEW_PAGE
END_REPORT
END_PROCEDURE
```

| FUNCTION | LIBRARY | COUNT | NAME |
|---|---|---|---|
| FORM | MEMRES | 3 | AUTO |
| FORM | OAFORM | 1 | DISPREMINDER |
| FORM | OAFORM | 4 | EMC |
| FORM | OAFORM | 1 | EMHEAD |
| FORM | MEMRES | 4 | MAIN |
| FORM | MEMRES | 1 | OA$EDIT |
| FORM | MEMRES | 2 | OA$LIST |
| FORM | OAFORM | 1 | WP |
| FORM | OAFORM | 2 | WPINDX |

### Other files and information: gaining insight into ALL-IN-1.

There are quite a few other ALL-IN-1 files which may be read with Datatrieve: the lists of font styles, file formats, printer device types, queue priorities, communications hosts and lines, and others. Most of the reasons given for manipulating the files shown before don't apply to these: either because they are so little used, because the ALL-IN-1 facilities are adequate, or because they are never (or should never be) modified. There is value in looking at these files, however, because examining them may give you a better insight into what data is needed by ALL-IN-1 for it's operation and maintenance, and how this data may be used or modified for the user environment.

### How the Definitions Were Obtained.

The easy way: get the Datatrieve / Forth Generation Languages SIG Library Tape Collection from the DECUS Library, or from the VAX SIG Symposia Tape. Some definitions have also been published in the DECUS Newsletter.

The hard way: DUMP, ANALYZE/RMS, and a Datatrieve record definition with just a few large fields. Field boundaries usually become obvious as the data "lines up", and field names could usually be derived from how the information is used (for example, the PROFILE form). Specifically for ALL-IN-1: there may also be a form with the same name as the file, which contains field definitions. I have not been able to find ALL-IN-1 forms which display ATTENDEE.DAT or PENDING.DAT so I cannot be certain that I have the same names for these fields that ALL-IN-1 uses.

If there is an ALL-IN-1 form which matches a data file you can sometimes use it within ALL-IN-1 to manipulate the data, though I have not found any forms, menus, or scripts in ALL-IN-1 that will

lead you to the DOCDB form (and I have looked), and once you are in this form it is tedious to use as you can only get one record at a time by using the FIXER field (entering a FOLDER doesn't seem to work). Manipulating some of the other files is easier. This access method does use FMS to spread the fields out on one screen, which can be nice, and it will allow you to add, delete, or modify records, but I prefer Datatrieve: I can always define my own form (or even copy the ALL-IN-1 form) and use it from within Datatrieve if I want to.

**Questions from the presentation.**

A user stated that he was seeing corruption in electronic mail (apparently in the association of mail files with users) and asked for comments. As I have not seen this problem myself, I cannot comment on it. There was no other response from the audience.

A representative from DEC made the statement that everything I was doing with Datatrieve could be done with ALL-IN-1 scripts, and that the script language contained facilities to manipulate all of the fields in all of the files mentioned. He then further stated that it was "dangerous" to perform any operation on an ALL-IN-1 data file with anything other than ALL-IN-1.

My response to this is threefold.

First: although I did not say so, it should be obvious that I should not have to be doing most of this work at all. For example, ALL-IN-1 should not leave meetings and calendar accesses assigned for users which have been deleted, and I shouldn't have to go in and correct this.

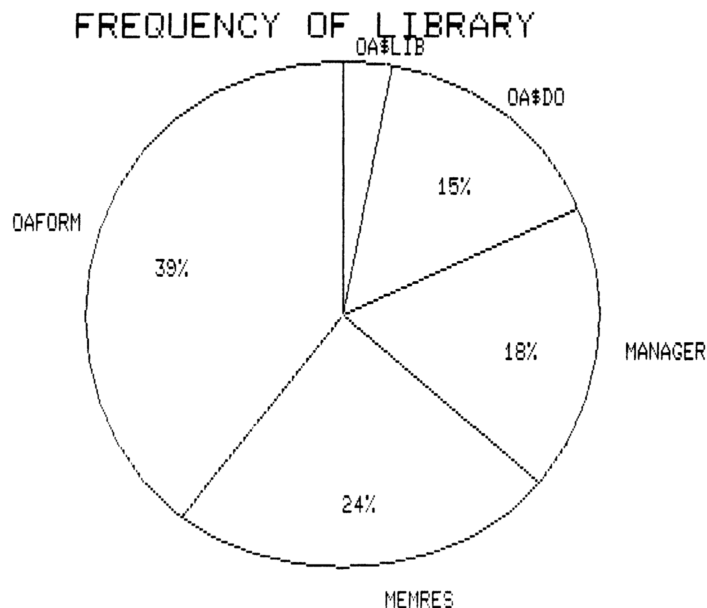
Second: given the fact that we must deal with the product as it currently exists, I, as a system manager, must choose the most efficient tool for the job. I have worked a little with ALL-IN-1 scripts and find them to be cumbersome, hard to learn, difficult to understand, and not very flexible. If I were to write a script which, for example, changed all of my users' profiles to enable DCL, that script would not find all users assigned to a particular disk: an entirely new script would have to be written. With Datatrieve, once I have the record definition for the user profile I can do whatever I want without having to write new code. Besides, writing ALL-IN-1 scripts is not my job: managing the system is. My time is valuable, and using major portions of it to learn to write scripts just to be able to manage ALL-IN-1 is not practical.

Finally, I take exception to the statement that accessing these files will somehow damage or compromise ALL-IN-1. I do agree that one should try to use ALL-IN-1 first, and I have stated clearly that, for example, one should not try to add a new user just by entering a record in the profile. On the other hand: all of these files are identified to VMS as normal RMS files, and if I access them using normal system services I should not be able to damage them. If this upsets ALL-IN-1, then ALL-IN-1 must be doing something drastically incorrect in the way IT is accessing these files, such as bypassing normal RMS access. From a practical standpoint, I and others have been manipulating these files with Datatrieve for several years with no ill effects yet. And in the case of processing the ALL-IN-1 logging file to extract useful information on how ALL-IN-1 is being used: once the file is written and logging is stopped ALL-IN-1 isn't using it anymore, and reading it with Datatrieve cannot cause any harm. Other DEC personnel have reacted favorably to this one particular use of Datatrieve for managing ALL-IN-1.

Of course, the more powerful a tool is the more dangerous it is, and it is entirely possible for a person to accidentally or deliberately enter incorrect data into one of these files and cause problems: but this could be done by the same person with EDT or even with the VMS COPY or RENAME commands. Until DEC can demonstrate that a problem really exists (or provides tools that make this alternate access unneccessary)

I intend to continue using Datatrieve to make my live as a system manager responsible for ALL-IN-1 easier.

A subject which was only briefly covered is that Datatrieve has graphics. It is often much easier to understand information in a graph than in tables of numbers. One example is in analyzing the logging file to see where the forms and scripts which are being used are stored. The following is just one simple example of the type of graphs which may be obtained.



# Solving Equations in Datatrieve

Bart Z. Lederman
World Communications Inc.
New York, NY 10004-2464

ABSTRACT

This paper highlights some of the methods of solving equations by using the mathematical, logical and statistical functions available in Datatrieve.

## Why?

Although there may be applications which are totally non-mathematical math functions, and some totally numerical applications that would probably not be implemented in Datatrieve, there are no sharp boundaries between data retrieval which is totally non-mathematical and data retrieval which requires some math. It is perfectly reasonable to implement applications which primarily store and retrieve data in Datatrieve which also require some math, such as inventory control, accounting, payroll, and many other applications, and it may be easier to implement the entire application in Datatrieve than to do some pieces in Datatrieve and other pieces in some other language. There is also the practical consideration that many people who are able to quickly learn and use Datatrieve do not have any "traditional" programming background, and may not have access to other programming resources and are faced with the prospect of doing it entirely in Datatrieve or not doing it at all.

## One Alternative: Callable Datatrieve

There is an alternative for programs which require a large amount of math but for which you would still like to use Datatrieve for data storage, retrieval, and reporting, and that is to use the call interface. You can write a program in most VAX languages and have it call Datatrieve: this allows you to write the math portion in your favorite language, and then have it pass data to or retrieve data from Datatrieve and execute Datatrieve statements from within your program. This is not part of this presentation.

## Basic Requirements

Datatrieve has all of the basic requirements for solving mathematical or logical equations, which are:

1. Mathematical operators:

   | | |
   |---|---|
   | Addition | + |
   | Subtraction | - |
   | Multiplication | * |
   | Division | / |

2. The ability to control the flow of calculations by logical (Boolean) operators (IF-THEN-ELSE).

3. The ability to repeat an action until a condition is met (FOR and WHILE).

This is enough to solve almost any

equation: it is, in fact, all that any computer has, or what any person would have if the equation were to be solved by hand.

### Other Functions

Most languages have libraries of functions for commonly used complex calculations (such as Logarithms, Trigonometry, Statistics, etc.), and so does Datatrieve. In addition, it is possible to add new functions to Datatrieve, either as "true" functions, or by writing procedures which are then used like subroutines or functions.

### First Example

In order to illustrate the process, I will set up a sample domain and run through a series of examples. The record definition is:

```
01 SAMPLE_REC.
    03 ITEM PIC 9.
    03 A PIC 999 EDIT_STRING ZZ9.
    03 B PIC 999 EDIT_STRING ZZ9.
    03 C PIC 999 EDIT_STRING ZZ9.
    03 T1 PIC 9999 EDIT_STRING ZZZ9.
    03 T2 PIC 9(6) EDIT_STRING ZZZ,ZZ9.
;
```

The domain is SAMPLE, and ITEM is a keyed field. This very simple domain is for demonstration purposes only.

The first example will be to calculate T1 by the formula T1 = (A + B) * C. While this could easily be done by making T1 a COMPUTED BY field it serves as a simple starting point. The FOR statement will be used as it is the easiest way to perform the same calculation for every record in a domain or collection.

For demonstration purposes, I've put the following data into the sample domain.

| ITEM | A | B | C | T1 | T2 |
|---|---|---|---|---|---|
| 1 | 3 | 5 | 7 | 0 | 0 |
| 2 | 7 | 5 | 3 | 0 | 0 |
| 3 | 2 | 6 | 4 | 0 | 0 |
| 4 | 7 | 3 | 4 | 0 | 0 |

A possible command sequence is to perform the calculation is:

```
READY SAMPLE MODIFY
FOR SAMPLE MODIFY USING T1 = (A + B) * C

PRINT SAMPLE SORTED BY DESC T1
```

After the commands, it looks like this:

| ITEM | A | B | C | T1 | T2 |
|---|---|---|---|---|---|
| 1 | 3 | 5 | 7 | 56 | 0 |
| 4 | 7 | 3 | 4 | 40 | 0 |
| 2 | 7 | 5 | 3 | 36 | 0 |
| 3 | 2 | 6 | 4 | 32 | 0 |

### Next: Running Totals

To do this it is necessary to store data from one record to another in some sort of variable or field, and this raises the first important point concerning "programming" in Datatrieve: there are no default variables as there are in BASIC or FORTRAN. All fields must be DEFINEd in a record or DECLAREd, and you must make the field large enough to hold the data planned for it.

```
DECLARE RUNNING PIC 9(6).
RUNNING = 0
FOR SAMPLE MODIFY USING T1 = (A + B) * C
FOR SAMPLE SORTED BY DESC T1
    MODIFY USING BEGIN
        RUNNING = RUNNING + T1
        T2 = RUNNING
        END
```

Since the running total will be in field T2, RUNNING has been declared to be the same size as T2 (though it doesn't have to be: it just has to be large enough

to hold the largest number which will be encountered). Notice that RUNNING must be initialized to zero: Datatrieve does not initialize any fields, though sometimes you get lucky and get a blank area of memory. The data is placed in the current collection rather than storing the running totals as the collection is being totaled by field T1 rather than by the primary key field of the sample domain.

| ITEM | A | B | C | T1 | T2 |
|---|---|---|---|---|---|
| 1 | 3 | 5 | 7 | 56 | 56 |
| 4 | 7 | 3 | 4 | 40 | 96 |
| 2 | 7 | 5 | 3 | 36 | 132 |
| 3 | 2 | 6 | 4 | 32 | 164 |

### More Difficult: Square Roots

This is useful for standard deviation and other statistical calculations. To test my algorithm, I will make a procedure which will accept a number and calculate the square root, printing out the value to see if it's correct.

```
DEFINE PROCEDURE TEMP
DECLARE T1 PIC 9999 EDIT_STRING ZZZ9.
DECLARE ROOT PIC 9999V99
    EDIT_STRING ZZZZ.Z9.
DECLARE TRY PIC 9999V99
    EDIT_STRING ZZZZ.Z9.
DECLARE DIF PIC S9999V99
    EDIT_STRING SZZZZ.Z9.
T1 = *.INPUT
TRY = 2
DIF = 1
WHILE DIF > 0.01 BEGIN
    ROOT=T1/TRY
    TRY=(ROOT+TRY)/2
    DIF=ROOT - TRY
    IF DIF<0 DIF=DIF*-1
    PRINT ROOT,TRY,DIF
    END
PRINT T1,ROOT,TRY,DIF
END_PROCEDURE
```

Notice that ROOT, TRY and DIF all have 2 decimal places reserved: if they did not, the square root would be calculated to the nearest whole number only. Note also that DIF has space reserved for a sign, as the difference between the last try and the present try could be positive or negative.

The WHILE statement is indispensable for this type of calculation as there are no labels and no GOTOs in Datatrieve. There is generally only two methods of performing repetitive calculations: the FOR statement which is used to perform some operation once on each record of a domain, and the WHILE statement for other repetitive calculations as it is not tied to a domain. The WHILE statement repeats until DIF (the difference between the present guess and the previous guess) is less than .01, this being the chosen limit of accuracy as the numbers were declared to have two decimal places. The constant 0.01 could be another field or variable, but if it is explicitly stated as it is here, it must have the leading zero.

The next three lines are the algorithm: divide the number by a guess and average the difference between the guess and the answer to form the next guess, repeating the process until the required accuracy is obtained. Notice that while spaces around math operators are usually optional, if you enter the second line as DIF=ROOT-TRY Datatrieve will tell you that field "ROOT_TRY" is undefined or used out of context. I recommend using spaces between items to make things more "readable", to avoid the minus sign vs. dash problem, and it usually makes things easier to edit.

The next line forces the value of DIF to be positive (the absolute value) to meet the condition of the WHILE statement, otherwise any negative value for DIF would end the calculation prematurely. The loop ends not when DIF is calculated but at the end of the block, which is how most "do loops" operate. The print line within the BEGIN-END block is a debugging aid: by placing a print statement here I can watch the values for each variable for each pass through the loop and determine if my logic is correct. When the procedure is correct, this line may be removed so that the final answer is printed by the last PRINT statement.

Another method of debugging is to place the commands in an indirect command file: this way syntax errors are more visible as each line is printed when read in by Datatrieve. (Remember to SET VERIFY to see things happening on a VAX.)

Two samples of the printout (with debug) look like this:

```
DTR> :TEMP
Enter INPUT: 25

 ROOT      TRY      DIF

  12.50     7.25     5.25
   3.44     5.34     1.90
   4.68     5.01      .33
   4.99     5.00      .01

 T1    ROOT      TRY      DIF

  25     4.99     5.00      .01

DTR> :TEMP
Enter INPUT: 35

 ROOT      TRY      DIF

  17.50     9.75     7.75
   3.58     6.66     3.08
   5.25     5.95      .70
   5.88     5.91      .03
   5.92     5.91      .01

 T1    ROOT      TRY      DIF

  35     5.92     5.91      .01

DTR>
```

One of the advantages of Datatrieve is that it appears to the user as an "interpreter", like the original BASIC: this means that you can take statements and execute them immediately without having to go through some intermediate compilation process. Since you can also edit your procedures from within Datatrieve, and examine your data before and after executing the procedure within Datatrieve, the development cycle can be quick, and the user only has to work with one product (or two, if you count the editor separately).

**Making the Procedure Useful**

Now that this procedure works, I will put it into a form where it can be used elsewhere, and call it SQRT. This is a way to build up a library of "functions" or "subroutines" usable in Datatrieve (which will even work in Datatrieve-11)

```
DEFINE PROCEDURE SQRT
IF T1 LE 0 ABORT "No Negative Numbers"
DECLARE ROOT USAGE IS REAL.
DECLARE TRY USAGE IS REAL.
DECLARE DIF USAGE IS REAL.
TRY = 2
DIF = 1
WHILE DIF > 0.01 BEGIN
     ROOT = T1 / TRY
     TRY = (ROOT + TRY) / 2
     DIF = ROOT - TRY
     IF DIF < 0 DIF = DIF * -1
     END
END_PROCEDURE
```

The print statements and definition of T1 have been removed: T1 must be defined before the procedure is called (so the calling procedure will make the space reservation and assign a value to it before calling this procedure), and ROOT will contain the answer when finished. It is the responsibility of the person writing the procedure to document carefully the fields which must be defined before the procedure is used, what types of fields they should be, and what field will contain the answer when finished.

Since negative numbers have no real square root, the first line has been added to insure that input to this procedure is not negative. The variable declarations are also slightly different. Rather than limit the range and accuracy of the procedure, the use of REAL variables allows these fields to accept very large or very small values. This procedure is now ready to be used as part of another procedure.

```
READY SAMPLE MODIFY
FIND SAMPLE
FOR CURRENT MODIFY USING BEGIN
    T1 = A + B + C
:SQRT
    T2 = 100 * ROOT
    END
```

The current domain now looks like this:

| ITEM | A | B | C | T1 | T2 |
|---|---|---|---|---|---|
| 1 | 3 | 5 | 7 | 15 | 387 |
| 2 | 7 | 5 | 3 | 15 | 387 |
| 3 | 2 | 6 | 4 | 12 | 346 |
| 4 | 7 | 3 | 4 | 14 | 374 |

It should be noted that there are alternate methods of dealing with an incorrect value for T1. One method is:

```
DEFINE PROCEDURE SQRT
  DECLARE ROOT USAGE IS REAL.
  ROOT = 0
  WHILE T1 GT 0 BEGIN
     DECLARE --- variables as before ---
            --- initialize variables ---
     WHILE ...
            ---- procedure as before ---
     END
  END
END_PROCEDURE
```

In this case, the entire procedure will be executed only if T1 is greater than zero, otherwise nothing is done, and ROOT defaults to zero (the rest of the procedure is unchanged). Another alternative is:

```
DEFINE PROCEDURE SQRT
    DECLARE ROOT USAGE IS REAL.
    IF T1 GT 0 BEGIN
        ---- procedure as above ---
    END ELSE
        ROOT = 0
    END
END_PROCEDURE
```

Here the IF-THEN-ELSE statement is used to execute the procedure if T1 is valid, and return a dummy value of zero for the root if T2 is invalid. The last three lines could be condensed into one, but writing it this way brings it closer to "normal" programming. One caution: most "structured" programmers would put the ELSE statement at the beginning of a new line, to clarify the structure. is <u>not</u> possible in Datatrieve: the verb ELSE <u>cannot</u> be the first word on a line. (<u>Generally</u> speaking: there are some "tricks" that can be done, but they generally aren't worth doing.)

The last example could also be performed in this manner:

```
DEFINE PROCEDURE SQRT
DECLARE ROOT USAGE IS REAL.
     IF T1 GT 0 BEGIN
                 ---- procedure as above ---
     END
     IF T1 LE 0 ROOT = 0
END_PROCEDURE
```

This appears to be both less structured and less efficient than the previous version, but it has one advantage in that it "compiles" faster under some circumstances (and uses less pool space in Datatrieve-11, though this won't bother VAX-DTR or DTR-20 users). In the first version, all of the statements from the IF to the last END (before the END_PROCEDURE) must be "compiled" before any part of the IF statement is executed, including evaluation of the IF condition itself, and this takes time (and pool). In the second version, each IF statement is "compiled" separately and executed separately. If you are going to be going through a repetitive series of calculations many times, it's usually faster to put all of the statements into one big WHILE statement or BEGIN-END block, let it all be compiled once when you enter the routine, and then let it run. The program may seem to "stall", while the statements are compiled, but once done the procedure will run very fast, about as fast as code compiled in other languages. If, however, you have a procedure which will probably be executed once only (or once in a while), and it contains a large number of IF conditions, it may be better to put them in as separate IF statements so you won't waste time compiling everything just to execute one small statement.

It should be noted that the

statement

```
WHILE DIF > 0.01 BEGIN
```

could have been written in many different ways, such as

```
 WHILE (DIF > 0.01 OR DIF < -0.01) BEGIN
```

or

```
WHILE DIF BETWEEN -0.01 AND 0.01 BEGIN
```

or any other valid Boolean expression. If any of these had been used, the line

```
IF DIF < 0 THEN DIF = DIF * -1
```

which converts negative values to positive values would not be required.

It should also be noted that the procedure name SQRT isn't really very descriptive. It would probably be better to call it something like "SQUARE_ROOT" or even "SQUARE_ROOT_2" (for 2 decimal places), and place it in a common dictionary for everyone to use once it has been debugged. The above example of use would then look more like this:

```
READY SAMPLE MODIFY
FIND SAMPLE
FOR CURRENT MODIFY USING BEGIN
    T1 = A + B + C
    :CDD$TOP.USER$LIBRARY.SQUARE_ROOT
    T2 = 100 * ROOT
    END
```

**Procedures versus Functions**

Should this be a procedure, or should a function be added to Datatrieve? Adding a function to Datatrieve is not at all difficult, especially if you are adding one of the VMS library routines, as you don't have to write any code. Doing the calculation in a function is often more efficient as Datatrieve doesn't have to "compile" the code each time it's used, and this is especially important if the function is going to be used often. If you want to add a function of your own, however, the first step is to write a subroutine to implement the function in some programming language that supports the VMS calling standard. This is going to be the first stumbling block for many Datatrieve users, who don't have a traditional "programming" background. Next, the function must be linked into the Datatrieve image: this isn't difficult, but many system managers resist change. Of more practical difficulty is what happens if your Datatrieve procedures have to be distributed to many different systems (for example, if you have developed something that is going to be throughout a company or corporation). If you do everything as Datatrieve procedures, you can distribute the Datatrieve code and be certain it will work: if you depend upon special functions, then you must be certain that those functions have been built into every Datatrieve image where your code will run. This can be especially difficult if the corporation is widely distributed, and, as often happens, different systems are running different versions of the operating system and Datatrieve.

**An Application: Least Squares Data Fit**

This is the "least squares" method of fitting the best line to a set of data points, and is often used for such things as predicting future growth. Though there is a least squares fit for many PLOTS in Datatrieve, the values are not retrievable for use within Datatrieve and this procedure will make the values usable for storage in a domain or other use.

The procedure is:

```
DEFINE PROCEDURE TREND
DECLARE SUMX USAGE IS REAL.
DECLARE SUMY USAGE IS REAL.
DECLARE SUMXY USAGE IS REAL.
DECLARE SUMXSQ USAGE IS REAL.
DECLARE SUMYSQ USAGE IS REAL.
DECLARE SLOPE USAGE IS REAL.
DECLARE INTERCEPT USAGE IS REAL.
DECLARE FIT USAGE IS REAL.
DECLARE TEMP USAGE IS REAL.
DECLARE N USAGE IS INTEGER.
N = 0
```

```
SUMX = 0
SUMY = 0
SUMXY = 0
SUMXSQ = 0
SUMYSQ = 0
READY SAMPLE
FOR SAMPLE BEGIN
    SUMX = SUMX + ITEM
    SUMY = SUMY + T1
    SUMXY = SUMXY + (ITEM * T1)
    SUMXSQ = SUMXSQ + (ITEM * ITEM)
    SUMYSQ = SUMYSQ + (T1 * T1)
    N = N + 1
    END
TEMP = ((SUMX * SUMY / N) - SUMXY)
SLOPE = TEMP /
    ((SUMX * SUMX / N) - SUMXSQ)
INTERCEPT = (SUMY - SLOPE * SUMX) / N
FIT = SLOPE * TEMP /
    (SUMYSQ - (SUMY * SUMY / N) )
PRINT SLOPE USING ZZZ9.9999,
    INTERCEPT USING ZZZ9.9999,
    FIT USING ZZZ9.9999
FINISH SAMPLE
RELEASE N
RELEASE TEMP
RELEASE FIT
RELEASE INTERCEPT
RELEASE SLOPE
RELEASE SUMYSQ
RELEASE SUMXSQ
RELEASE SUMXY
RELEASE SUMY
RELEASE SUMX
END_PROCEDURE
```

The FOR statement is used to process the domain and sum up some values which will be required for the calculation. The the procedure is summing up the values for X (ITEM) and Y (T1) and counting up the number of items in N rather than using the SUM and COUNT commands because it has to go through the domain once anyway to sum the squares of the variables and the products of the two variables, and it is more efficient to also sum the other values at the same time than to have Datatrieve make additional passes through the domain to to the summing and counting. Also note the use of an intermediate calculation for the value of TEMP: this expression is used in two other places, and it is more efficient to store the value than to calculate it twice, and it is also faster.

```
 ITEM   A    B    C  T1        T2

  1     0    0    0 1200        0
  2     0    0    0 1800        0
  3     0    0    0 1600        0
  4     0    0    0 1900        0
  5     0    0    0 1800        0
  6     0    0    0 2100        0

DTR> :TREND

   SLOPE   INTERCEPT    FIT

  137.1429 1253.3334    0.6954

DTR>
```

## A Few Suggestions

More complex problems may be approached by breaking them down into smaller sections, each of which should yield to one of the methods presented. The following subjects in the Datatrieve manual will be of interest: the ABORT, DECLARE, FOR, WHILE, CHOICE, and IF-THEN-ELSE commands; arithmetic and Boolean expressions; (procedures and indirect command files; optimization; and especially the section dealing with the USAGE clause, which describes the internal format of the different types of numbers. COMP {INTEGER, BYTE, WORD, LONG, QUAD} is usually the most efficient type of storage; for real numbers REAL {FLOAT} and DISPLAY (the default) should be the next most efficient. The author recommends avoiding COMP_3 {PACKED}, COMP_5 {ZONED}, and COMP_6 except when needed to read data written by other programs, and DATE (except for date calculations)

## Built-In Functions

VAX-Datatrieve has the following built-in functions which might be used for mathematical operations (not including the date functions):

```
FN$ABS  FN$ATAN FN$COS   FN$EXP FN$FLOOR
FN$HEX  FN$LN   FN$LOG10 FN$MOD FN$NINT
FN$SIGN FN$SIN  FN$SQRT  FN$TAN
```

Functions are quite simple to use.

```
FOR SAMPLE MODIFY USING BEGIN
    T1 = A + B + C
    T2 = 100 * FN$SQRT(T1)
END
```

or, if the intermediate value of T1 isn't needed:

```
FOR SAMPLE MODIFY USING T2 = 100 *
FN$SQRT(A + B + C)
```

Or you can modify the original record definition:

```
01 SAMPLE_REC.
    03 ITEM PIC 9.
    03 A PIC 999 EDIT_STRING ZZ9.
    03 B PIC 999 EDIT_STRING ZZ9.
    03 C PIC 999 EDIT_STRING ZZ9.
    03 T2 COMPUTED BY
        FN$SQRT(A + B + C).
;
```

Or, if you don't want to store the value:

```
FOR SAMPLE PRINT FN$SQRT(A + B + C)
```

Functions have the advantage that they can be incorporated into places where procedures cannot be used, or cannot be easily used.

To add your own functions to Datatrieve, you have to modify a file, DTRFND.MAR, supplied with Datatrieve. When installing Datatrieve, you are asked if you want to save certain customization files: say YES to save the function file. Although this is a Macro-32 language source file, it doesn't really look like assembler language as it simply consists of function definitions

```
; FN$POWER - Raise a real number to a real power
;
; Output is a floating value in R0, R1
; Input is two floating values passed by immediate value
;

$DTR$FUN_DEF FN$POWER, OTS$POWRR, 2
    $DTR$FUN_OUT_ARG  TYPE = FUN$K_VALUE, DTYPE = DSC$K_DTYPE_F
    $DTR$FUN_HEADER  HDR = <"Power">
    $DTR$FUN_IN_ARG  TYPE = FUN$K_VALUE, DTYPE = DSC$K_DTYPE_F, ORDER = 1
    $DTR$FUN_IN_ARG  TYPE = FUN$K_VALUE, DTYPE = DSC$K_DTYPE_F, ORDER = 2
$DTR$FUN_END_DEF
```

In this instance you don't have have to write your own routine to do the work as it uses a routine in a library supplied with VMS. More function definitions like this, and a Datatrieve procedure that generates the definitions, may be found in the Datatrieve / Fourth Generation Languages SIG Library tape, which is in the DECUS library and on the VAX SIG Symposia tape.

### Where to find Equations

Books on the particular subject (for example, a book on statistics for standard deviation or trend line fitting) are a good beginning, especially the older books which give instructions for solving the equations by hand. They will also give worked examples, so the user can compare the answer obtained in Datatrieve with the answers in the book to determine if the equation has been correctly solved. There are also books published for high-school and college math classes containing nothing but formulas, and some have functions expanded into series, which are particularly suitable for solution by computer.

EDUSIG

# VOICE: A COMPUTER CONTROLLED TELEPHONE INFORMATION SYSTEM

Lisa M. Rotunni
Edward C. Hohmann
Son V. Phan
James A. Rounds
College of Engineering
California State Polytechnic University
Pomona, California 91768

The College of Engineering at California State Polytechnic University, Pomona announced a new experimental information system called VOICE in January of 1987. VOICE is a computer controlled system which can be accessed from any touch-tone telephone. The object of the system was to provide access to the best information available on such items as the engineering course schedule, and faculty office hours, the College of Engineering and its academic departments, scheduling deadlines and other important dates.

The VOICE system consists of a computer driven voice synthesizer and a telephone. The computer is able to take input from the buttons on the phone the same way it can from a computer terminal. Through a series of menus, the caller is able to interactively acquire information, rather than listening to a fixed tape-recorded message.

This paper will discuss the planning of the VOICE menus, the database information which VOICE puts on-line over the phone, the programming required to control the telephone interaction, and the current operation of VOICE.

## BACKGROUND

The College of Engineering at Cal Poly University, Pomona, has 8 undergraduate majors and a Master of Engineering program with approximately 4,000 graduate and undergraduate students. We offer 800 course sections per quarter in each of 4 quarters per year and we have about 300 full and part-time faculty and staff members.

In the College of Engineering Dean's office, we are using the database RDM from Interactive Technologies Inc. on a PDP 11/23 computer for administration and instructional planning. Our administrative personnel database system contains all employees and their positions, the salary schedule, and other support data files. It is used for faculty hiring, cost estimation, and directory reports. The course scheduling system is used to prepare the course offerings for a given quarter. It is not an on-line registration system. It contains the course schedule, course catalog, and various support files. In our student petition tracking system, we log the status of change of major and general academic petitions which students must submit to our office for approval.

These database systems, developed over the past three years, have now reached a fairly mature state. However, when we looked at the system we realized that something was lacking. We had all of this information on the computer, but it was only available to us, or to other people through printed reports. Much of the information we had compiled was of interest to others on campus, and even people off campus. So, we instituted the VOICE project. The purpose of VOICE is to make information more easily available on campus, to make information available off campus, and to unburden secretaries from having to answer routine, often repeated questions.

## THE DECtalk

The hardware which makes the VOICE system possible is the DECtalk. The DECtalk is a box that converts written text into speech. Send the DECtalk a page of written text, and the DECtalk will read it aloud. An internal dictionary tells it how to pronounce a great many words, and it will attempt to sound out words it does not know. The DECtalk can be hooked up to a computer and to a terminal and/or telephone. Using the escape sequence commands which the DECtalk understands, you can get it to answer the phone, take input from the phone buttons and relay it to the computer, and speak the information the computer sends.

The DECtalk comes with nine defined voices, which DEC has given names like

Perfect Paul, Beautiful Betty, Frail Frank, and Huge Harry. The instructions provide some information on how to create your own voice. However, they discourage you from attempting it because the voice development process is very difficult.

The computer controls the DECtalk, taking input from the phone and sending information to be spoken. We decided to write our controlling program for the system in Fortran, because it is easily portable between computers and because we had available sufficient expertise in that language. Any programming language should be able to control the DECtalk.

**PRONUNCIATION**

As soon as we started dealing with the DECtalk, we discovered that it had some limitations in interpreting written text. It does well, when you consider that it cannot think as it reads in the same way that we do. However, we make many assumptions in converting written text into speech; DECtalk does not always make the same assumptions. The person at the other end of the phone has no idea what the written word looks like, thus care must be taken to make the spoken word understandable.

For example, the DECtalk does interpret numbers correctly. If you write the number 1,300, the DECtalk will say "One thousand three hundred". But we do not always read numbers that way. We certainly don't read phone numbers that way! And we also don't read room numbers that way. If you write the numbers 214 and 301, the DECtalk will say "Two hundred fourteen" and "Three hundred one". But, when those numbers are rooms we usually say "Two fourteen" and "Three oh one". The easiest way to get the DECtalk to say these rooms properly is to split up the numbers when printing them; for example, 2 14. However, if you write 3 0 1, the DECTalk will say "Three zero one". You have to put the letter O in the middle to get the usual pronunciation.

Our preferred approach has been to adjust the text we send to the DECtalk to get the proper pronunciation. The DECtalk can handle phonetic spellings, and words spelled phonetically can be mixed in with ordinary words in your text. However, phonetic spellings are somewhat difficult and definitely time-consuming to produce.

We have come up with some other ideas to adjust the pronunciation of words without resorting to phonetic spelling. Short forms of words present a special challenge to the DECtalk. They are often not spoken as they are written. For example, the California State Polytechnic University, Pomona is locally called just "Cal Poly University". The Poly is short for Polytechnic and is pronounced like the name "Polly". However, according to the rules of English language pronunciation, "poly" rhymes with "holy", not "holly" ("polly"). The DECtalk speaks according to the rules. The phonetic spelling for "polly" is "[p'awliy]", and we could substitute that in our text every time as "Cal [p'awliy] University", but rather than have to remember or use that, why not just write "Cal Polly University". A simple spelling change will allow the DECtalk to pronounce the word properly.

Some words have to be adjusted to local pronunciation. When the DECtalk says the word "for" it sounds like "fur" rather than "four". We substitute "four" for "for". "Status" comes out as "state us", so we separate out the syllables to get "stat us", which sounds better - at least here in Southern California.

It is also easiest to leave out anything you do not want read, like leading zeros in numbers.

The DECtalk speaks with expression. It is not a computer monotone. It adjusts it's tone with punctuation as well, so that a sentence followed by a question mark has the rising inflection of a question. An exclamation point works well to add emphasis, especially on words of once syllable. On multisyllabic words the end effect sounds less desirable.

Unfortunately, not all problems can be as easily solved with spelling changes. Several parts of our application read faculty last names. Our experience has been that the DECtalk will get about 50% of last names correct with no adjustments, which is not a bad average. But we couldn't have it mispronouncing half the faculty! In this case, we did opt to use phonetic spellings. We added a field to our faculty database called Phonetic Last Name. Whenever a new faculty member is entered, Phonetic Last Name is entered as the normal spelling of that person's last name. The pronunciation is then tested on the DECtalk, and if it is properly pronounced, no phonetic spelling is needed. If the name is not correct on the DECtalk, a phonetic spelling is entered in the field and adjusted until the DECtalk reads the name properly.

**APPLICATION DESIGN**

It is always important to remember that the user cannot see what he is listening to. This critical fact has led us to rephrase an old proverb - VOICE is "Heard but not Seen" - and is the primary factor in designing the application. This leads to our two most important design considerations. The first is limiting the number of choices per menu; you cannot pick from a list of twenty read to you over the phone. Four or five is about the most that can be reasonably handled. With these limitations, you will need menus, submenus, and even sub-submenus. The second most important design consideration is to group items into reasonable categories, making it possible to search through the menu in a logical fashion.

Our application is called VOICE. In VOICE, we chose to have several defined movement keys, used throughout the application. The asterisk will always return you to the previous menu; zero, typed from a menu, will repeat that menu. We considered it important also to have some overt way for the user to end the conversation. People are used to logging off computers or saying good-bye at the end of a phone conversation. The pound sign terminates the session, causing VOICE to say "Thanks for calling. Goodbye." and hang up the phone. The DECtalk will hang up the phone automatically after a certain amount of time with no input. If the user doesn't bother to hit the pound sign and just hangs up the phone, the DECtalk will hang up. However, if the user wants to feel that he has completed the call, he can hit pound and have VOICE say good-bye to him. This use of symbols seems consistent with what other people are using in other phone systems we have seen.

For our application, we chose to use the voices Perfect Paul and Beautiful Betty. These are the most "normal" voices the DECtalk provides. Contrary to DEC's recommendation, we have made minor adjustments to Paul which we think improve his sound.

The format for our menu files can be seen in Figure 1. In the design of our controlling program, any line which begins with a period is interpreted as a command and not sent to the DECtalk to speak. In the case of a menu, the ".FILE" command indicates what menu to read next based on what menu choice the user selects.

**SCHEDULE MENU**

```
Now,
select from the schedule menu.

To look up courses in the class schedule,
press 1!

To look up faculty class schedules, press 2!

To look up faculty or staff office hours,
press 3!

At any time, to back up to the previous menu,
press *.

Make your choice any time!

.FILE:1,"CLASSES.MNU
.FILE:2,"FAC_CLASSES.MNU"
.FILENAME:3,"OFFICE_HOURS.MNU"

.CHOOSE
```

Figure 1

**DATA**

It is easy to get the DECtalk to speak prepared text like the menu. It is little more than a very flexible tape recorder if that is all it does. This has led us to the very emphatic conclusion that IT IS ALL USELESS UNLESS YOU HAVE THE DATA! If you do not have an established data management system, it would be a monumental task to put information into a computer for the sole purpose of having the information read over the phone. However, if you already have the data in a computer, and are using it for other applications as well, it is relatively simple to hang a DECtalk onto the system. Then you can develop an application which gets the desired information out over the phone, to those who would not be able to access it otherwise.

In our case, information from the employee data file, department codes file, faculty office hours file, and course schedule all go to make up the VOICE class schedule, faculty class schedule, faculty office hours, and phonebook. The change of major petitions and general academic petitions stand alone as data files, which go out directly to VOICE.

VOICE does not access the database directly. We run reports from our database to excerpt the public information and put it into the VOICE data file format. VOICE then reads those reports. In that respect, VOICE is not really on-line information. We update the reports periodically as changes occur in the related data. However, VOICE is still the best possible information available, considerably more accurate than any printed report, and the only information available over the phone. Figure 2 shows how our academic personnel system data flows into the VOICE system.

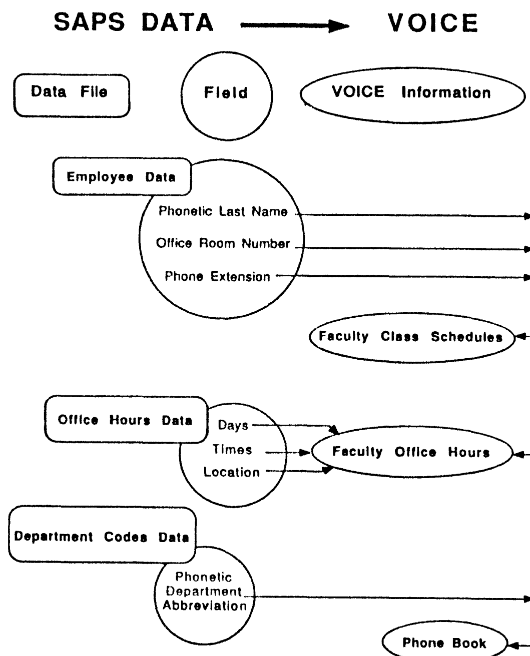



Figure 2

VOICE data files are constructed differently from the VOICE menus. We use fixed length records for ease of searching the files, so the first item in the file is always the number of lines per record. Then, we have the message which is always spoken when someone searches the file. This message tells the date that the file was last updated. Then comes the message that is spoken after an unsuccessful search of the file. Finally we have the data which is searched by and the information which is read on a successful search. This format is shown in the class schedule data file sample in Figure 3.

**CLASS SCHEDULE DATA**

```
007

.MESSAGE
Fall schedule was up dated on 10 29 87.
.END

.NOFIND
I cannot find an engineering course,
with that code in the schedule.
.END

6111001
['iyjiy'ar] 1 10, section 1, meets,
Monday, Wednesday, Friday,
8 ['eyehm], through, 10:50 ['eyehm],
in Building 9, Room 1 33,
with instructor Hudspeth.

6330201
[s'iyeych'iy], 3 02, section 1, meets,
Monday, Tuesday, Wednesday, Friday,
7 [p'iyehm], through, 7:50 [p'iyehm],
in Building 13, Room 2 14,
with instructor [jhorjhiy'aadiyz].
```

Figure 3

For the course schedule, we search by the course code, which is a seven digit number. It was more difficult picking the search data for the faculty schedules. We decided not to try and work out some way of searching by name. This would by necessity be quite complicated, what with three letters per phone key and Q and Z missing entirely.

Almost every faculty member has his own phone extension, so we chose to use the four digit phone extension to search for faculty data. If two faculty members do share a phone, VOICE tells you that and asks you which one you want. Besides ease of implementation, we considered phone extension to be a good choice because it was readily available, common information. And VOICE is a phone oriented system. In the student system, students can search by their university personal identification numbers. This has the added benefit of providing some security, in that generally the individual student is the only non-administrative person who would know that number.

Since we were using phone extensions, we needed a way for people to get these numbers if they didn't happen to have a campus phone book handy. This led to the VOICE phone

book. The phone book reads phone extensions for the entire faculty of a department when you enter the phone extension for that department. This list was very difficult to listen to, so we added a pause every three names.

**ADDITIONAL FEATURES**

Additional, non-data features of the VOICE System were designed primarily to answer oft repeated questions. One of these is the important dates and deadlines section. The university puts out a flyer each quarter listing such things as the last day to add classes, the last day to drop without record, and so on. We put this information into VOICE. Currently, we have this information in a text file, which must be updated manually to remove dates which are past. We are working on an advanced feature in which we can bracket the information with a conditional statement indicating between which dates and times to read the information. Then only the appropriate announcements will be read at any given time.

Several of our engineering programs are impacted and have elevated admission standards. These standards are somewhat complicated, since they depend on how many college units the student has, their high school GPA, SAT scores and/or college GPA, as well as requiring certain mathematics and physics courses. Many questions are asked by potential students regarding the impaction criteria, and so we wanted this information available on VOICE. Written out in words for VOICE to read, the impaction criteria for one major alone takes an entire page of text. The only way to understand the information is to be ready on the phone with pencil and paper. The best feature of this part of the system is that it give the phone number of an answering machine where the applicant can leave a message to have the admission criteria mailed out in written form. The DECtalk is capable of dialing out, and we are working on having it transfer people to the answering machine so that they will not need to dial again.

Lastly, we have included narratives. In the University Catalog they have a brief narrative at the beginning of each major describing the department and its emphasis. It was virtually free to include this in VOICE, and so we did.

For ourselves in the system, we have created a log file which our controlling controlling program updates automatically. It shows when a caller dialed in, what menus were progressed through, what data was searched. For diagnostic purposes we included where in the data file the data was found. The last item shows when the call ended whether the system timed out or the caller exited using the pound sign.

When we first announced VOICE last January, we put out a flyer showing the menus and their interrelationships, the phone number of the system, and some important notes. For example, you can key on the phone as fast as you like. Your input will be processed immediately and you will not need to listen to the intermediate menus. Callers who have this flyer can progress rapidly to the information they want out of the system. This flyer is shown on the following page in Figure 4.

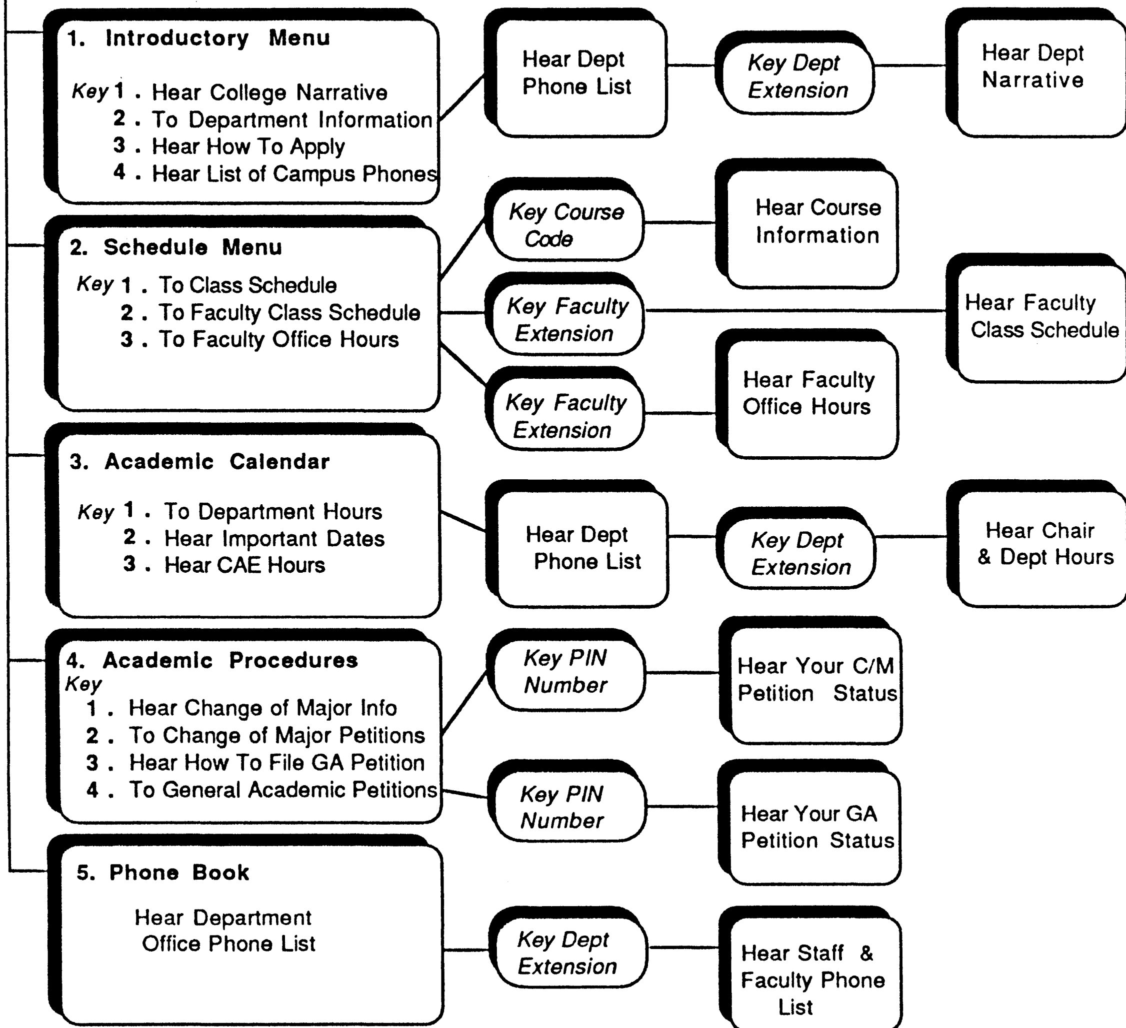
**CONCLUSIONS**

At this point we are very happy with VOICE. It is being used. Calls are more frequent at the beginning of the quarter than at the end, but this is expected. No program is finished until the users start complaining, and I actually had a faculty member complain to me last quarter that I had not updated the office hours; that made me feel that I had accomplished something. My greatest concern with the DECtalk is that it sounds like it learned English as a foreign language. We have the single-line DECtalk, and I learned at DECUS that the multi-line version is supposed to sound better. We are working on getting an upgrade of our system to take advantage of DEC's new letter to sound conversion rules. We are still working on improving our controlling program and our embedded command structure to make them more powerful and easier to work with.

Cal Poly University, Pomona

## College of Engineering

**Interactive VOICE Information System at (714) 869-2604**




August 1, 1987

# GRAPHICS APPLICATIONS SIG

# Report Generation Using a Visual Programming Interface

Tim Dudley
Cognos Incorporated
3755 Riverside Drive
Ottawa, Ontario
K1G 3N3
CANADA
(613) 738-1440

Command language interfaces are not always the most appropriate tool at the initial stages of report design. A loosely constrained graphical notation can be much more useful. Visual programming techniques introduced on the Xerox Star™, and popularized by the Apple Lisa™ and the Macintosh™, have now made the use of such a graphical notation much more feasible. Also, the direct manipulation techniques described by Schneiderman are now viable because of the wide-spread availability of bit-mapped graphics screens and pointing devices such as the mouse.

This paper briefly discusses visual programming concepts, and then describes the implementation of a visual programming interface (VPI) for a 4GL report writer. The basis for the design is an object-action syntax. A set of icons was designed which represent atomic report entities, and a graphic editor built to manipulate these entities into a report structure. Attribute sheets associated with each of the report entities allow definition of the report entities to the data dictionary. A menu bar controls menus of all possible actions to be performed on the objects. A facility to switch easily between the graphical and textual representation of the report is provided, with direct manipulation editing available in both representations. Modifications made in one representation are automatically reflected in the other.

The combination of the VPI with a 4GL makes the design and modification of reports remarkably straightforward, and suitable both for end users and application programmers.

## Introduction:

In late 1984 and early 1985, Cognos Inc. was involved in a consulting contract with the Ice Branch of Environment Canada, to produce a conceptual design, functional specification, hardware-software specification, and detailed implementation plan for a system which was to archive all available information on sea and lake ice, and iceberg conditions, in Canadian and adjacent waters [1]. Two constraints had a considerable impact on the approach that was taken in the design of the system: only two people were available to produce the work, and the work was to be done over an elapsed time of four months, including the Christmas and New Years holidays. The limited resources and timeframe forced us to rely primarily on computer-generated diagrams, made up from a minimum closed set of icons (which we designed), as the basis for the work. We simply didn't have time to produce a textual specification. We found that our design approach was completely altered as a result. In the process of cleaning up system diagrams, we discovered connections in the diagram that were incorrect. Some areas of the diagram had become extremely cluttered; attention was being drawn to those areas, strictly because of their visual appearance. By rearranging the diagrams to eliminate the clutter, we were able to remove unwanted redundancies, minimize the number of interconnections between entities, and in effect produce a "canonical form" drawing of the system. One system diagram was shown to a colleague who had been involved earlier in the project. He looked at the diagram for about fifteen seconds, and asked where the connection was between two of the system modules, knowing that that connection had been part of the original User Requirements. He had found a mistake in the design, which had appeared visually as a blank area in the system diagram. At that point, we realized that we could literally design graphically at the high level. We also realized that we could concisely communicate a tremendous amount of conceptual information, by primarily using diagrams illustrated by text, rather than by using text illustrated by diagrams. The diagramming also forced us to modularize our design, and do it fairly rigidly, without overspecifying or overconstraining the individual modules or their

interfaces, which could have imposed hidden restrictions on the design.

Our experience with using icons to design this system has prompted us to investigate the potential of applying a similar approach to business applications. Graphics hardware, particularly bit-mapped screens and pointing devices such as the mouse, have now become widespread and relatively inexpensive. Windowing systems are becoming common, and object-oriented programming is fairly well understood. This combination of events has resulted in some interesting software development techniques, particularly in the areas of direct manipulation and visual programming. The remainder of this paper describes our current research, utilizing these techniques, toward the development of a visual programming interface (VPI) to a report writer.

## Background:

The terms "visual programming" and "program visualization" are sometimes used to refer to the same thing, when in fact they represent entirely different concepts. According to Meyers [2], visual programming refers to a system that allows a user to specify a program graphically, while program visualization, on the other hand, allows a conventional, textually-specified program to be viewed graphically. This distinction is blurred in the literature, but can be easily remembered by thinking of visual programming as the specification stage of programming, and of program visualization as the documentation or analysis stage.

Another important concept in this context is that of "direct manipulation" [3]. Direct manipulation is the set of principles which include visual representation of the objects of interest, selection and physical actions instead of keyword commands, and rapid incremental reversible operations [4]. It is the principle used by such systems as the Xerox Star™, and the Apple Macintosh™ and Lisa™, as well as most video games. It lends itself well to object-oriented programming, and is sometimes referred to as the "point-and-shoot" approach. This is the approach in which the designer selects an object (points), then causes some action to be performed on the object (shoots). An example of this point-and-shoot technique in a word processing application is to highlight a block of text, and then to choose a CUT or COPY or DELETE action from a menu. This approach can be very straightforward and easy to learn. File manipulation becomes almost automatic: To delete a file, one drags a picture of it onto a picture of a trash can. It isn't necessary to try to remenber whether the command to delete a file is **DELETE , DEL, REM, X, KILL, RMFILE,** and so forth The principle at work here is that our reading vocabularies are considerably larger than our speaking or writing vocabularies, and that we can do more error-free work by pointing at things and moving them around, than we can by writing about them.

The visual programming and direct manipulation techniques also make it easier for a designer to present a system to the user in the user's own framework, rather than in computer-ese. Users "...develop conceptual models - mental representations of the workings of the system." [5]. The user's conceptual model must correctly predict the behavior of the system. The system designer must therefore anticipate the conceptual model, and present a consistent *external myth* which will reinforce it. (The reason the word "myth" is used is because it is a representation of the internal workings of the system, and may not correspond to the actual internals of the system. [5] ). If the user's conceptual model corresponds well to the designer's external myth, the user will be able to deal with his problem at a higher level of abstraction, and not get mired in the workings of the application or the user interface. The use of icons and their direct manipulation lends itself well to the presentation of an external myth. The most common example of an external myth is the familiar desktop metaphor.

## Design of the VPI:

The visual programming interface which we are designing is based on a noun-verb-adjective/adverb syntax, in which nouns are represented by icons, verbs by menu items, and adjectives and adverbs by property sheets and dialog boxes. This syntax is presented to the user, using the desktop metaphor, as shown in Figure 1:

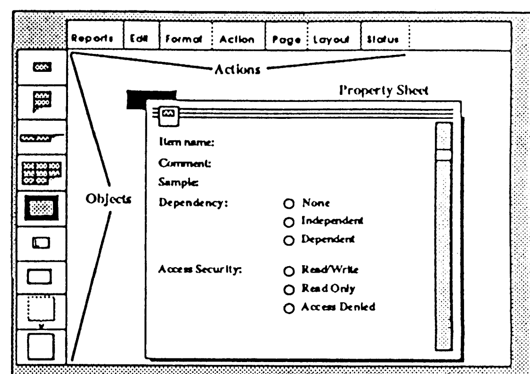



Figure 1

The user creates a report specification by selecting the appropriate icons from the icon menu, arranging them on the desktop workspace according to how the final report is to look, and defining each of the report elements (represented by the icon and its attributes or properties) to the dictionary, through the use of menu selections and property sheets.

Each icon has an associated property sheet, which can be considered as the window into the dictionary for the particular report element represented by that icon. The report writer itself is completely defined by the determination of what report elements are made available to the user through the icon menu, which actions are available through the menu bar, and what attributes are available on the property sheets associated with each report element. Dialog boxes are available for actions which require clarification. The combination of icon, menu items and property sheets must present a consistent graphics vocabulary to the user in order to be effective.

The icons used in our VPI, and their definitions, are listed in Figure 2:

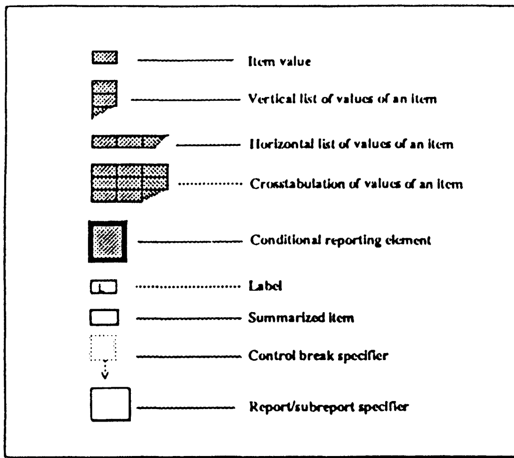



Figure 2

(The design of the icons, and in fact the design of the whole desktop, is a critical part of the success of a visual programming interface. This issue is very well addressed by Verplank in [6]. The icons must visually resemble the report elements which they represent, and give the illusion of directly manipulable objects. The whole desktop needs to present visual order, and provide user focus. In addition, the entire system needs to reveal a structure which is consistent with the user's conceptual model, so that the user always knows where s/he is, and what will happen if s/he hits the DELETE key [6], [7]. It must be noted that in our system, the graphical design of the icons and the desktop has not been finalized, and the representations shown here are intendedto act only as prototypical vehicles, in order to convey the general idea of the interface.)

The best way of describing the VPI is with an example. (For this example, it is assumed that the hardware includes a bitmapped graphics screen and a single-button mouse.) Suppose a report is to be created which consists of a sorted list of an organization's employees and their telephone extension numbers. The telephone numbers are four digits long, the first digit of which indicates the floor on which the employee works. The list is to be sorted alphabetically, by floor, with appropriate titles.

Using the VPI to produce the structure for this report, the user creates the diagram shown in Figure 3:

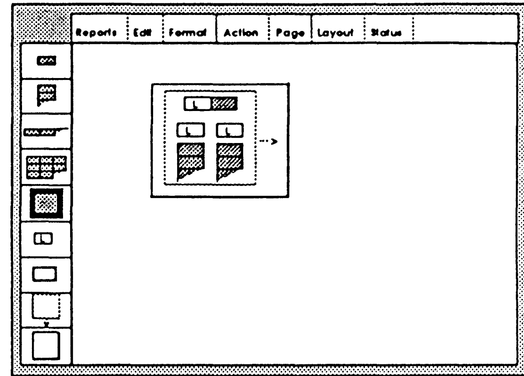



Figure 3

Icons are placed on the desktop by clicking on them in the icon menu with the mouse, then dragging them into place on the workspace. In this example, two vertical list icons are placed next to each other, representing lists of the employee names and the employee extensions. Label icons are then placed above each list. A label and item icon are placed above that group, adjacent to each other. This picture represents the portion of the report for one floor. Because this group is to be repeated for each floor, the control-break-specifier icon is placed around it. (In the cases of the control-break-specifier icon and the report/subreport-specifier icon, once they are placed on the desktop workspace, they can each be selected and stretched to any rectangular shape, in order to enclose other icons.) The direction indicator on the control-break-specifier icon is set to point to the right, indicating that the group is to be repeated horizontally, instead of vertically. The report/subreport-specifier icon is then placed such that it encloses the entire group. The resulting diagram defines the structure of the report, and all that remains is to identify each of the report elements to the dictionary.

Figure 4 shows how items are identified:

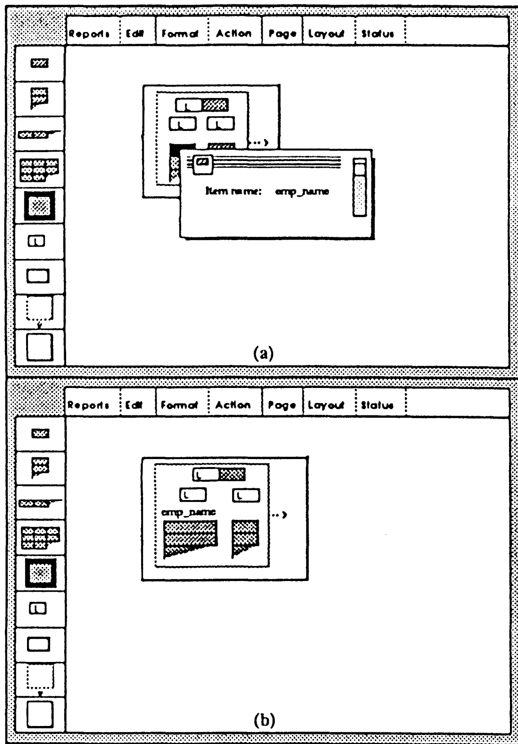


Figure 4

The property sheet for the left list is brought up by double-clicking the icon (Figure 4a). When the property sheet is displayed, the user keys in the item name for the list. If that item is defined in the dictionary, its attributes are placed on the property sheet. The user then clicks the close box in the upper left corner of the property sheet. The resulting picture (Figure 4b) indicates that the item is defined by displaying the item name at the top of the list. The icon is also expanded by the system to the size necessary to correspond to its size attribute in the dictionary.

Figure 5 shows how items are defined which aren't already in the dictionary. The item icon in the title for the repeating group is double-clicked, bringing up its property sheet. When the item name is keyed in, and the name is not found in the dictionary, the system prompts for a definition. In this case, the item is the floor number, which is calculated as indicated on the property sheet in the diagram. Default attributes are assigned, depending on what is keyed in. The resulting picture is shown in Figure 5b.

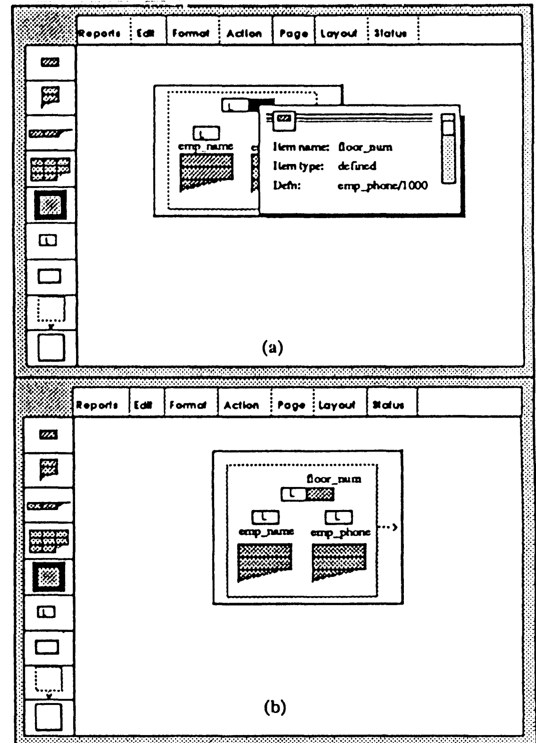


Figure 5

Figure 6 shows how control breaks are specified:

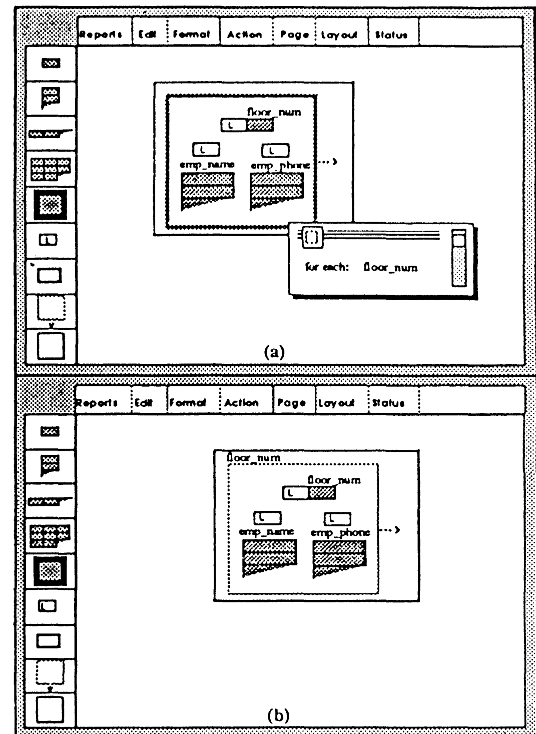


Figure 6

The control-break-specifier icon is double-clicked, bringing up its property sheet, and the control break variable is keyed in (Figure 6a). Note that the control break variable does not have to appear on the report itself. The resulting picture is shown in Figure 6b.

The employee list is to be sorted. This is an action, which is invoked as shown in Figure 7:

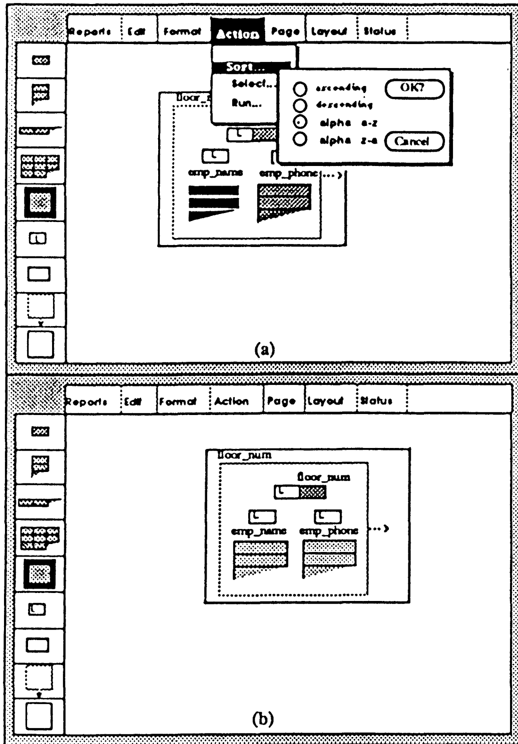



Figure 7

The employee name list icon is highlighted by clicking on the top item, then dragging the cursor down the list. This selects the entire list, as opposed to just the first item in the list. While the list icon is highlighted, the **Action** menu is pulled down, and the **Sort** item is selected (Figure 7a). This causes a dialog box to be displayed, asking what type of sort is to be performed. Because the list associated with the selected icon is alphabetic, the numeric sort options are disabled. (This is an example of how the system can be constructed to prevent the user from making errors.) When the **OK?** box is clicked, the menu is hidden, and the resulting picture is displayed (Figure 7b). Note that the fill pattern of both lists has been changed. This shows two things: that some action has taken place on the indicated report element, and that the indicated report element has some dependencies on other report elements. The actual dependencies are not shown on the report structure diagram, but are available on the relevant property sheets.

Figure 8 shows how label strings can be defined. Note that the string value appears inside the icon, becoming part of that icon. The icon is also expanded to accomodate the string. The font and size of the string can either be set or modified on the icon itself, by highlighting the icon, then selecting the size and font, or on the property sheet.

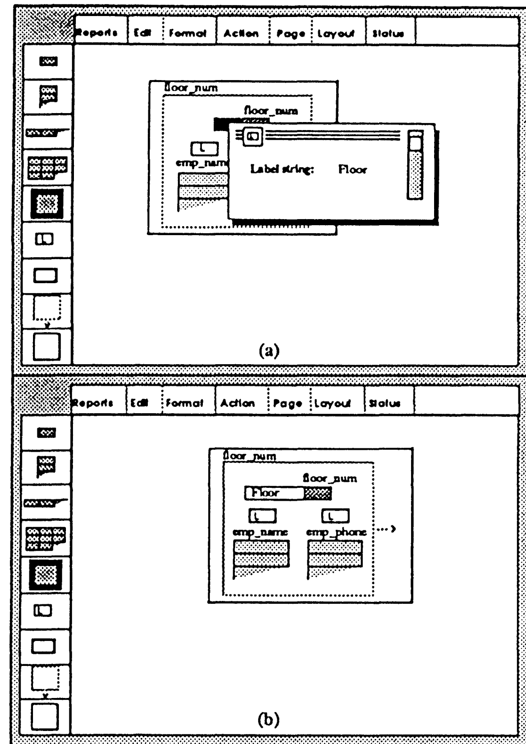



Figure 8

Figure 9 shows how the report/subreport is named:

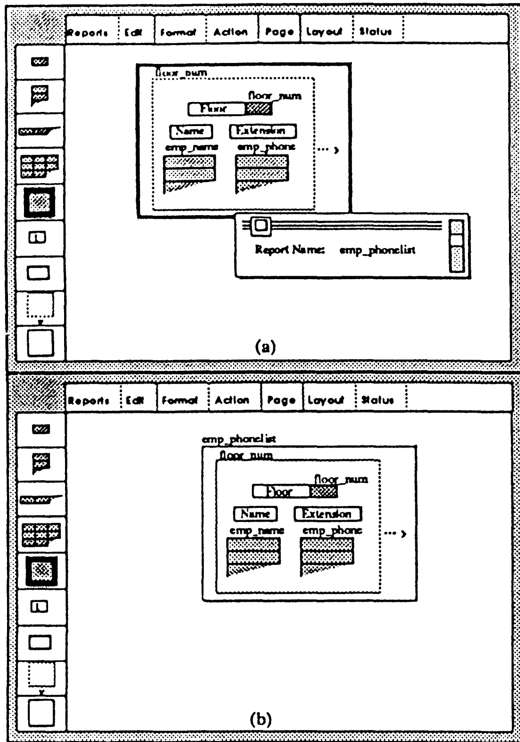


Figure 9

Once completed, the report/subreport can be saved by use of the **Save** item in the File menu.

All physical layout on the page in the above example is done by defaults in the system, depending on the relative spacing of the icons in the report structure diagram. However, in the case of pre-printed forms, certain layouts are predetermined, and the report must fit the layout. The **Layout** menu provides the facility for accurately placing report elements on a page, through the facilities of grids, rulers, and calipers (Figure 10).

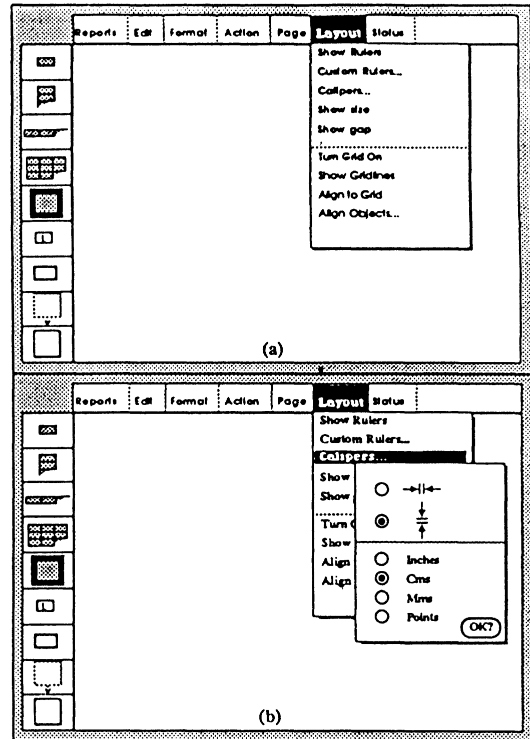


Figure 10

Rulers can be displayed across the top and down the left side of the screen, and can be set to a variety of units (centimeters, inches, points, etc.). As the cursor is moved across the work area, the current cursor position is tracked on both of the rulers. Used in conjunction with an enabled grid, this makes it quite straightforward to accurately position report elements on a physical page.

The calipers provide a mechanism for directly specifying distances between report elements, and sizes of report element fields. (The caliper icon does not appear in the icon menu, because its use is one of action, and it is not part of the report structure.) The calipers are used by selecting the horizontal or vertical caliper icon from the **Layout/Calipers** menu and dragging it onto the work area. One end of the caliper can be locked by clicking on it, and the other end positioned by dragging to the desired (horizontal or vertical) position, then clicking on that end to lock it. If the **Show size** or **Show gap** menu items have been selected, the distance spanned by the caliper will be continuously displayed between the caliper ends while the caliper is being set. If not, it will not be displayed until both ends are locked into position. In order to force a dimension onto the caliper, the user locks it into position, then clicks on the displayed dimension, and keys in the desired dimension. The caliper will be adjusted to the new dimension by the system, and can then be repositioned as desired.

The actual positions of report elements can be seen on the property sheet associated with the report/subreport icon, or by highlighting an icon or pair of icons and selecting the **Show size** or **Show gap** menu items from the **Layout** menu.

## Some Problems:

One of the major principles in the design of user interfaces is "Know thy user". This presents some severe difficulties in designing an interface to something like a report writer (which is in fact a graphical language), because of the diversity of potential users. These users range from the hacker/guru, to the application programmer, to the Vice President of Finance, to the CEO's administrative assistant. Each of these users approaches the system with a different conceptual model, and with different expectations of how to use it. In spite of the popularity of systems that utilize direct manipulation and icons, many of these users simply don't take them seriously...they don't believe that such systems provide enough flexibility to allow them to do what they want to do. This, in turn, poses the question of whether or not to design a closed system that does only the tasks which have been specified for it in the Task Analysis (which is another of the major principles of system design...).

We are addressing this problem of appealing to a diversity of users by providing several interfaces, and making it easy to switch among them. The Visual Programming Interface produces an internal report definition, which can then be edited using a syntax editor, or which can be run to create a report. The report itself can then be manipulated using the What-You-See-Is-What-You-Get (WYSIWYG) principle, in combination with direct manipulation techniques. If the report structure is not satisfactory ( in the example above, maybe the control break group should be printed vertically rather than horizontally), the system provides the capability for the user to switch to the interface which is best suited for making the required changes. (Incidentally, this change is extremely easy to make using the VPI...one just changes the direction of the control-break-specifier icon. The system does the remainder of the formatting.) The problem that this approach (the provision of several interfaces) leads to is how to make a smooth transition between interfaces, and how to maintain an internal representation of the report which can be efficiently operated on by all three interfaces.

Another problem with which we have been dealing is that of conditional reporting. One of the underlying principles of the VPI is that people recognize documents initially by their visual appearance...that is, how they are laid out. The VPI takes the approach that the user lays out the report how s/he wants it to look, then goes about defining each report element to the dictionary. However, the case arises in which the report format may change, depending on some condition. In other words, there may be instances when a report has three columns in a group, but other instances where it may only have two. Our first attempts to solve this were seriously frustrating, and we eventually decided that it couldn't be done in the VPI context. We have subsequently decided to include the concept of a Black Box, (and developed an icon for it), to represent a conditional reporting situation. Our current thinking is that the Black Box will appear in the report structure wherever there is a conditional reporting situation. When the Black Box icon is opened (by double-clicking, for example), the alternative report structures will be displayed, as will the determining condition associated with each of them. At the time of this writing, this problem had not been fully addressed.

The conditional reporting problem causes some philosophical consternation with the VPI. The diagrams generated using the VPI were intended to be analagous to the schematics for an electrical diagram or printed circuit board. The idea was that the diagram was essentially a software schematic...a diagrammatic representation of the actual report, which was recognizable immediately, and fairly clearly understood, because of its visual shape (the idea of "revealed structure" again). However, the analogy suffers with conditional reporting, particularly when the conditional reporting variable doesn't physically appear on the report. The analogy suffers further, because a printed circuit board schematic doesn't necessarily look like the finished board, but we are saying that the report structure "schematic" strongly resembles the finished report. We are still struggling with this one.

## Future Work:

At the time of this writing, the implementation of the actual VPI was just beginning, and consequently, we have not yet been able to test our ideas in a prototypical environment.

We believe that we will benefit considerable from building the prototype, and will be able to refine the design to provide an excellent interface. The main area where we expect to learn is in the definitions of the actions which appear in the menu bar. Because we are building a closed system with this interface, we must ensure that this set of actions is at least necessary and sufficient, and we can't expect to determine a priori whether this is the case.

We have not yet decided how to handle file access and linkages. We are considering having the system infer which files are required, and how they should be linked, from the report structure. Other alternatives are to have the user select file names from a scrolling dialog box, or to provide another desktop at the file level, and have the user specify the required files and linkages graphically.

We also haven't finalized the characteristics of the property sheets...what goes on them, how they are organized, and how they interface to the dictionary. We are considering using a scrolling dialog box on the property sheet, from which the user can select item names instead of keying them in. There is also the question of whether the propertry sheets should look the same to all users, or if they should be different, depending on the user's security access to the dictionary.

When the interface is stabilized, we expect to have learned enough about how to deal with visual programming and direct manipulation to provide a VPI for the entire STORM environment, and potentially to operating systems in general.

## Summary:

We have designed, and are in the process of building, a visual programming interface to a report writer. This interface will allow users to graphically design report structures, which can then be executed to produce finished reports. We believe that this approach makes the design and modification of reports remarkably straightforward, and can significantly improve the productivity both of application programmers and of end users.

## References:

[1] Tim Dudley. "Graphics in Software Design." *Computer Graphics World,* **9(2)**, February 1986.

[2] Brad Meyers. "Visual Programming, Programming by Example, and Program Visualization: A Taxonomy." *Human Factors in Computing Systems: Proceedings SIGCHI '86.* Boston, MA, USA. April 13-17, 1986

[3] Ben Schneiderman. "Direct Manipulation: A Step Beyond Programming Languages," *IEEE Computer.* **16(8)** August 1983

[4] Ben Schneiderman. "Direct Manipulation: An Object-Oriented Visual Programming Language," *Human Factors in Computing Systems: Tutorial 17. SIGCHI '86.* Boston, MA, USA. April 13-17, 1986

[5] Richard Rubinstein and Harry Hersh. *The Human Factor: Designing Computer Systems for People*, Digital Press, 1984

[6] Bill Verplank. "Designing Graphical User Interfaces", *Human Factors in Computing Systems: Tutorial 1. SIGCHI '86.* Boston, MA, USA. April 13-17, 1986

[7] Adele Goldberg. Keynote address, *SIGCHI '86.* Boston, MA, USA. April 13-17, 1986

# INTRODUCTION TO SMG, THE VMS SCREEN MANAGEMENT UTILITY

Robert L. Hays
KMS Fusion, Inc.
Ann Arbor, Michigan

ABSTRACT

This article introduces VAX/VMS application programmers to the Screen Management Utility (SMG) provided with VMS and includes FORTRAN examples. Windows, pull-down menus, pop-up windows and menus, and forms are all only a few SMG calls away. SMG concepts and ways to use SMG to perform screen management for a friendlier user-interface are discussed.

## INTRODUCTION

Digital Equipment Corporation's Screen Management Utility (SMG) provides a set of native-mode, run-time library (RTL) routines for terminal display management. Disk space and link time decreases when SMG is used because RTL routines are not included in the executable image as code, but instead as a transfer address.

SMG controls display and input at terminals in a device-independent fashion and at a higher level than terminal control sequences like Regis. All SMG features are not available on all terminals; for non-VT terminals, a terminal description, called a terminal table, must be created that describes the way the terminal supports SMG functions.

The Screen Management Utility keeps track of mundane display problems like redrawing areas of the screen that were occluded, simple line and box drawing commands, and also asynchronous input and broadcast message trapping and display. SMG can even be used with FMS, the Forms Management System, with caution.

Reference information is provided by Digital Equipment Corporation in the following:

> Volume 8C, <u>VAX/VMS System Routines</u>
> <u>Guide to Programming on VAX/VMS</u>

Playing with other programs and systems is also essential for gleening the most useful techniques for user interfaces. Books and papers on interactive graphics can provide additional user interface ideas.

## THE SMG ROUTINES

The three major elements of SMG input and output are pasteboards, virtual displays, and virtual keyboards. Each of these are associated with an identifier returned when any of these elements are created.

### Pasteboards

Pasteboards are like a bulletin board: information can be pasted onto the pasteboard, removed from the pasteboard or moved on the pasteboard, much like paper on a bulletin board. The RTL call used to create a pasteboard is SMG$CREATE_PASTEBOARD. SMG$CREATE_PASTEBOARD accepts, as input, the output device desired for the pasteboard, which defaults to SYS$OUTPUT, and a flag for clearing the screen on pasteboard creation. The function returns a pasteboard identifier and the number of rows and columns for the output device.

### Virtual Displays

Virtual displays are like paper; you put down what you want to appear on the paper, and then paste the display on the pasteboard, which makes the display visible. SMG$CREATE_VIRTUAL_DISPLAY creates a virtual display and returns a display identifier used to address the display. The call specifies the number of rows and columns for the display, any display attributes desired, such as a display border or control character display, and video attributes, such as bold or reverse video. SMG$PASTE_VIRTUAL_DISPLAY pastes, or outputs, the data on the virtual display specified by the virtual display identifier to a specified pasteboard. This call associates a window defined by the virtual display with physical screen rows and columns. SMG$MOVE_VIRTUAL_DISPLAY changes the location for a virtual display on a pasteboard. Virtual Displays can also be removed and deleted. SMG$UNPASTE_VIRTUAL_DISPLAY removes a display from the pasteboard, and therefore from the output device. SMG$DELETE_VIRTUAL_DISPLAY deletes a virtual display. Until a display is deleted, it can be pasted, unpasted, and then pasted again as many times as desired. Virtual Displays can also be 'popped', which removes all displays created after the specified display. SMG$POP_VIRTUAL_DISPLAY deletes all displays created after the specified virtual display identifier.

Output to a virtual display can be done with the following routines:

```
SMG$PUT_CHARS(_HIGHWIDE)(_WIDE)
SMG$PUT_LINE(_HIGHWIDE)(_WIDE)
```

SMG$PUT_CHARS, SMG$PUT_CHARS_HIGHWIDE, and SMG$PUT_CHARS_WIDE places a character string starting at a specified virtual display location, leaving the cursor at the end of the characters output. SMG$PUT_LINE, SMG$PUT_LINE_HIGHWIDE, and SMG$PUT_LINE_WIDE places characters at the current cursor location for the entire line length, padding the line with blanks at the end if necessary. The SMG$PUT_LINE routines also automatically scroll a window if text reachs the top or bottom line, depending on the scroll direction specified in the PUT_LINE call. Video attributes can be controlled by values found in $SMGDEF:

| | |
|---|---|
| SMG$M_BOLD | Bold video |
| SMG$M_REVERSE | Reverse video |
| SMG$M_BLINK | Blinking video |
| SMG$M_UNDERLINE | Underscore video |

SMG$PUT_VIRTUAL_DISPLAY_ENCODED allows multiple video attributes on one line in one call. SMG$PUT_WITH_SCROLL acts similarly to SMG$PUT_LINE, but supports line wrapping.

Buffering output to the terminal decreases the number of I/O operations performed. One way to buffer output is with SMG$BEGIN_DISPLAY_UPDATE, which batches all output to the virtual display until a call to SMG$END_DISPLAY_UPDATE occurs. Additionally, the SMG$CONTROL_MODE call can set the pasteboard to enable buffering, enable minimal buffering, clear the screen on exit, and not use tabs in screen output.

Virtual Keyboards

Virtual keyboards permit user input from an input device, for example a terminal keyboard. SMG$CREATE_VIRTUAL_KEYBOARD associates a keyboard identifier with an input device. A default file specification and the recall buffer size can be added to the call. Like virtual displays, keyboards can be deleted, in this case by a call to SMG$DELETE_VIRTUAL_KEYBOARD. SMG provides the following calls for performing user input from a virtual keyboard:

SMG$READ_STRING
SMG$READ_KEYSTROKE
SMG$READ_COMPOSED_LINE
SMG$READ_VERIFY

SMG$READ_STRING reads multiple characters into a data buffer until a terminator or maximum string length occurs. Terminators are defined in the include file $SMGDEF in the form SMG$K_TRM_<key_name>, where <key_name> is UPPERCASE_<letter> or LOWERCASE_<letter> for any <letter> in the alphabet, ONE through NINE (note that ZERO is missing) for the typewriter number keys, CTRL<letter> for the CTRL control key held while pressing a letter key, KP<number> for the keypad keys, PF<number> for the PF keys, F<number> for the function keys, HELP for the help key, DO for the DO key, UP for the up arrow, DOWN for the down arrow, LEFT for the left arrow, RIGHT for the right arrow, and E1 through E6 for the FIND through NEXT SCREEN keys. The other keys also have names; extract the $SMGDEF module from SYS$LIBRARY:FORSYSDEF.TLB and read it for more information.

Special terminators, such as '?' for help, can be specified in SMG$READ_STRING and SMG$READ_VERIFY through use of the terminator set arguement. This is a descriptor, passed by reference, containing the table length in bytes and the address of a bit mask, where each bit corresponds to a key, e. g. the zero bit of the first word is Control-A. SMG$READ_VERIFY acts like SMG$READ_STRING, accept that it verifies the input string with a picture of the desired string, as defined in the VAX/VMS I/O User's Reference Manual: Part I.

SMG$READ_KEYSTROKE waits for any key, and returns a unique value for the keystroke. SMG$READ_COMPOSED_LINE accepts input of characters and keys, and translates the key strokes based upon values in the key definition table, which can be set up with a calls to SMG$CREATE_KEY_TABLE, SMG$ADD_KEY_DEF, SMG$LOAD_KEY_DEFS, and SMG$DEFINE_KEY; this is useful for redefining key functions.

The input functions accept a modifiers arguement which accepts values from $TRMDEF; typical choices are TRM$M_NORECALL and TRM$M_NOECHO.

Unsolicited Input

SMG$ENABLE_UNSOLICITED_INPUT enables asynchronous input via AST routine. Whenever unsolicited input occurs, the AST routine is called with the arguements: pasteboard identifier, an optional arguement specified by SMG$ENABLE_UNSOLICITED_INPUT, registers R0, R1, the program counter and the program status register. The AST routine must perform any reads desired; control is all that is passed to the routine. SMG$ENABLE_UNSOLICITED_INPUT uses the mailbox interface to pass status messages to the AST routine. Note that a call to SMG$FIND_CURSOR_DISPLAY returns the most recently pasted virtual display with the cursor in it, which can be used by an AST routine to operate in a particular window. SMG$DISABLE_UNSOLICITED_INPUT disables all AST routines for a pasteboard.

Control Keys and Broadcast Messages

Out-of-band ASTs, such as CTRL-T, and broadcast messages, such as operator messages and new mail messages, can be trapped and displayed through SMG. SMG$SET_OUT_OF_BAND_ASTS can enable or disable out-of-band character trapping. The routine uses a key mask to define which characters to trap and which to pass. An AST routine is also specified, with the first arguement a structure containing the pasteboard identifier and an optional value specified in the call to SMG$SET_OUT_OF_BAND_ASTS; the remaining arguements are the registers R0, R1, PC, and PSL. Broadcast messages are trapped when SMG$SET_BROADCAST_TRAPPING is called. An optional AST routine and arguement is available. When a broadcast message is sensed, the AST routine is called. Typically, this routine will open a small window to warn the user that a message is waiting. Then, when the user wants to see the message(s), SMG$GET_BROADCAST_MESSAGE will return each message one at a time until the error status SMG$_NO_MORMSG is returned, indicating no more messages waiting. SMG$DISABLE_BROADCAST_TRAPPING will stop broadcast trapping. Be careful, as there is a note that using the LIB$SPAWN routine while trapping broadcast messages can cause problems.

## Non-SMG output while SMG is using a device

Non-SMG output can be done while using SMG; it is critical to save the current screen and then redisplay it once non-SMG output is complete. Create a virtual display of one row and one column, then specify this display in the call to SMG$SAVE_PHYSICAL_SCREEN before non-SMG output and SMG$RESTORE_PHYSICAL_SCREEN after non-SMG output. Note that SMG will clear the screen by default on the calls to SMG$SAVE_PHYSICAL_SCREEN and SMG$RESTORE_PHYSICAL_SCREEN.

Screen or line repainting is performed with SMG$REPAINT_SCREEN and SMG$REPAINT_LINE, respectively. Alternatively, SMG$INVALIDATE_DISPLAY marks a given display as invalid, ie. corrupted, and repaints the display only.

## SOME RULES FOR USING SMG

Separate I/O from calculations as much as possible. By using separate modules for screen I/O, program flow is more obvious and such modules can be reused.

Always pass the pasteboard ID to any routine that opens a window. Event flags get allocated with each call to SMG$CREATE_PASTEBOARD and are not deallocated if the pasteboard already exists, so eventually all the local flags get used up. While it might seem better to call SMG$CREATE_PASTEBOARD in any self-contained routine and check for the status return of SMG$_PASALREXI, it doesn't work correctly now; this should be fixed in V. 5.0 of VAX/VMS.

If a routine must do I/O to a window thats already created, pass the display ID; pass the keyboard ID if user response is required.

Only delete a pasteboard if the current level created it; this simplifies maintenance later. The error return from SMG$CREATE_PASTEBOARD indicates whether the pasteboard existed before the call to SMG$CREATE_PASTEBOARD. Delete windows and keyboards at same level as creation, also.

Borders take up one column and one row, so leave room on your display for them. If you start a display in physical column one with a border, the left hand border will not appear.

Bold the current menu title ( could DEC support bolding the border some day, please? ). This gives the user a visual indicator of where current activity is.

Watch out for control-y and control-z. Control-z can mess up a display quickly, leaving a reverse video Exit message.

For error processing, either write one routine and call it, assuring that the routine has access to a window, outputs info, waits for user response and then cleans up, or use a one or more line area, open the window at the start, and then pass the required IDs to all other routines. If the first option is chosen, then the error routine can either open and close the display as needed, or instead resize, reposition, and then repaste the display. Another alternative for error handling is a condition handler that uses SMG for output. This requires the use of the MESSAGE utility for all error values and messages, but can trap all system errors and faults as well.

Don't perform output to an occluded window. SMG can get the display confused if this rule is not followed. Unpaste the occluding window, perform output, and then repaste, or move the occluding window around so that it no longer occludes the display.

Watch out for requesting input past the edge of the terminal screen; SMG does not care, and can hang up in this case.

When using SMG$READ_STRING, by default up arrow is a call to the recall buffer. To use up arrow as a terminator, you must include the TRM$M_NORECALL value to the call to SMG$READ_STRING in the modifiers field. Using the TRM$M_NOECHO modifier will allow for no echo input, for example passwords. Note that if you use NORECALL, there will be no additions to the recall buffer for the given input. If you don't use TRM$M_NOEDIT to turn off line editing, then the returned terminator for up arrow is SMG$K_TRM_CTRLB, not SMG$K_TRM_UP.

Provide a repaint screen feature. This is very important and easy to do, using SMG$REPAINT_SCREEN, SMG$REPAINT_LINE, and SMG$INVALIDATE_DISPLAY.

## EXAMPLES USING SMG

The following code fragments show ways to use SMG calls. The first fragment (Figure 1) creates a pasteboard, a virtual keyboard, and a virtual display. The size of the display is number_of_rows by number_of_columns, and is positioned on the physical device at row first_row and column first_column. The code in Figure 2 turns the cursor off and labels the window border with a title. Note that the title is displayed in bold face. Data is displayed in a window in Figure 3. The code in Figure 4 displays the current item at row i in reverse-video, waits for user input, and then clears the reverse video. Final clean up, consisting of redisplaying the cursor, forceing any waiting output to the screen, and deleting the virtual display, is shown in Figure 5

## MENU SYSTEM FUNCTIONS

One use for SMG is a bar and pulldown menu-based user interface. The following functions are required:

INITIALIZE_DISPLAY
CREATE_SMG_WINDOW
MAIN_PULLDOWN_MENU
PULLDOWN_MENU
MULTIPLE_PULLDOWN_MENU
PULLDOWN_LIST
MULTIPLE_PULLDOWN_LIST

where MAIN_PULLDOWN_MENU is a bar menu routine, PULLDOWN_MENU pulls down lists for selection, INITIALIZE_DISPLAY creates a pasteboard and virtual displays for error messages and saving SMG displays for non-SMG output, and CREATE_SMG_WINDOW creates a virtual display used by MAIN_PULLDOWN_MENU and any other user-designed displays. An additional routine, MULTIPLE_PULLDOWN_MENU, is useful; this routine allows multiple selections before returning, much like checking off items on a list. List, or non-modifiable, versions of the pulldown menu routines are also useful for displaying information.

In all cases, a standard set of terminators are used:

| | |
|---|---|
| Up arrow - | in pulldown menu, up one selection; |
| Down arrow - | in pulldown menu, down one selection; in bar menu, pull down menu or select item; |
| Left arrow - | in bar menu, move left one selection; |
| Right arrow - | in bar menu, move right one selection; |
| Enter - | select current item and exit to next level down, such as pull down a menu; |
| Return - | Same as Enter; |
| PF2 - | Help |
| Help - | Help |
| Prev Screen - | Up one page or left one line; |
| Next Screen - | Down one page or right one line; |

Cursoring past either end of a menu wraps the cursor back to the opposite end of the menu.

The current item is displayed in reverse video. In multiple selection menus, the currently selected items are also displayed in bold video. The current window title is displayed in bold video.

The menu routines return a character string for the selection, where an item in the bar menu is demarcated by one or more blanks; the pulldown menu routines use two character string arrays, one for item display and a corresponding one for the return values. They can be the same array.

A windowed help interface that uses SMG and the LBR$OUTPUT_HELP utility call to access standard help library modules is important; this was the first routine written, and hooks exist in all the modules for inclusion of help. A general help interface technique, such as having each menu layer be a numbered layer of help, is not developed yet.

Creating non-menu displays is easy. Use INITIALIZE_DISPLAY to create a pasteboard and virtual keyboard. CREATE_SMG_WINDOW makes a display the desired size and in the desired location on the screen. Fill the display using SMG$PUT_CHARS and SMG$PUT_LINE. Whenever input is desired, use the SMG$READ routines as appropriate. Positioning is much like most graphics systems, where the virtual display coordinates are like world coordinates and pasteboard coordinates like device coordinates.

## USING SMG FOR A WINDOWED USER INTERFACE TO VMS

A menu toolbox makes user interfaces much faster to whip up. The only ingredient missing is a simple, small forms driver using SMG. This would allow display and updating of system and user information quickly and easily. Other tools using SMG are already available for use in a VMS interface.

SWING is a tool by Eric Andresen of General Research Corporation from the Fall 1986 DECUS Symposium Tape. SWING displays the directory tree visually from left to right, up to down, and uses the arrow cursor keys to change directories in the tree. The routine uses SMG for all input and output. Modifying this routine to operate in a sized window is not difficult, and can then be used inside a program to allow users to specify a directory to work in.

RADIX is also on the Fall 1986 DECUS Symposium tape and was written by Phil Worth at E-Systems. RADIX provides a friendly way to perform base conversion and ASCII conversion in bits, bytes, words, and long words. RADIX originally used a local VT100 screen management package, but has been converted to use SMG.

Calculators are easy to program using SMG. As a matter of fact, almost any application is fairly easy to apply SMG to. DCL-like SHOW commands are good candidates, since the information is often paged or listed.

A general interface to DCL, one not unlike the Macintosh user interface, is a long-term goal. This has worked quite well so far except for SPAWN commands and broadcast trapping. Some VMS facilities are not available to higher-level languages, such as wild-card searches on anything other than file specifications. Some system services are not clearly documented and require some playing with to get right.

A simple interface for updating a general interface with new routines and to allow non-SMG programs to be executed from within the shell is needed. One possible portion of the answer is the VPW subprocess routines, since V. 5.0 will probably support SMG across subprocesses, a necessity for creating a complete environment that is currently lacking.

```
 status = SMG$CREATE_PASTEBOARD ( pbid, 'SYS$OUTPUT', , , )
 status = SMG$CREATE_VIRTUAL_KEYBOARD ( kdid, 'SYS$INPUT',
+                                       , , 0 )
 status = SMG$CREATE_VIRTUAL_DISPLAY ( number_of_rows,
+                                      number_of_cols, vdid,
+                                      SMG$M_BORDER, , )
 status = SMG$PASTE_VIRTUAL_DISPLAY ( vdid, pbid, first_row,
+                                     first_column )
```

Figure 1

```
 status = SMG$SET_CURSOR_MODE ( pbid, 1 )
 status = SMG$LABEL_BORDER ( vdid, main_name(1:i),
+                           , , SMG$M_BOLD, , )
```

Figure 2

```
 IF ( last_page .NE. curr_page ) THEN
     DO i = first_item_page, last_item_page
         curr_col = 2
         curr_row = curr_row + 1
         status = SMG$PUT_CHARS ( vdid, prompts(i),
+                                 curr_row, curr_col )
     END DO
 END IF
```

Figure 3

```
 status = SMG$CHANGE_RENDITION ( vdid, i, 1, 1, num_cols,
+                               SMG$M_REVERSE, )
 status = SMG$SET_KEYPAD_MODE ( kdid, 1 )
 status = SMG$FLUSH_BUFFER ( pbid )
 status = SMG$READ_KEYSTROKE ( kdid, term_code, , , vdid )
 status = SMG$CHANGE_RENDITION ( vdid, i, 1, 1, num_cols,
+                               SMG$M_REVERSE,
+                               SMG$M_REVERSE )
```

Figure 4

```
 status = SMG$SET_CURSOR_MODE ( pbid, 0 )
 status = SMG$FLUSH_BUFFER ( pbid )
 CALL SMG$DELETE_VIRTUAL_DISPLAY ( vdid )
```

Figure 5

"Prepared for the Department of Energy under Contract No. DE-AC08-87DP10560."

"By acceptance of this article, the publisher and/or recipient acknowledges the U.S. Government's right to retain a nonexclusive, royalty-free license in and to any copyright covering the article."

NOTE: This notation need not appear in the published article.

# LANGUAGES AND TOOLS SIG

# Customizing VAXLSE for your language

**Jana Van Wyk**
SAS Institute Inc.
Cary, North Carolina 27512

**Abstract**

Many software developers are not familiar with all the syntax rules of a particular language, especially if that language has many features or they are novices in the use of the language. Program development can be enhanced with a customized application of the VAX Language Sensitive Editor (VAXLSE) that contains language specific information to help programmers develop programs quickly and accurately.

This paper presents an application of VAXLSE that was developed to be used with the SAS language. The SAS language is a programming language used with the SAS System, a software system for data management and analysis, statistical analysis, report writing and presentation graphics. This paper includes a description of the definition of VAXLSE language constructs and templates and the formation of tokens and placeholders. An example of how to customize VAXLSE by transforming a language's context free productions and terminals into the corresponding VAXLSE placeholders and tokens is presented.

The intended audience for this paper is programmers who are familiar with the idea of the grammar of a language and who wish to develop a tool which helps themselves or others in learning and using the syntax of a specialized language.

## Overview of VAXLSE – What is VAXLSE and why use it?

VAXLSE is an advanced text editor with language specific information that aids in writing and compiling source code. VAXLSE provides:

- pre-defined formatted language constructs for many VAX/VMS supported languages like Ada, PL/1 and C.
- templates for VAX subroutine libraries.
- complete integration with VAXSCA, VAX DEC/CMS and VAXTPU.
- source can be compiled, errors reviewed and corrected within one editing session.
- a VAXLSE definition language which allows you to customize VAXLSE by creating templates for your own specialized language needs.

These features can be of help to programmers in many different ways. They can help a novice programmer learn about the syntax of a language and what individual components make up a language. We have all experienced the frustration of sorting through program statements trying to figure out where we omitted a semicolon or where we need an extra END statement. VAXLSE can help you avoid this frustration by providing the semicolon or END statement for you when it is needed. The customization capabilities of VAXLSE allow you to help automate the writing of any kind of program by creating a language specific VAXLSE environment file.

You can write a program using VAXLSE by learning the use of 4 control key sequences plus the normal editing commands of either EVE or EDT. The use of language templates can be completely controlled by the use of

- control-e,
- control-p,
- control-n and
- control-k.

Control-e allows you to "expand" a placeholder, Control-n moves the cursor to the beginning of the "next" placeholder, Control-p moves the cursor to the beginning of the "previous" placeholder and Control-k erases or "kills" the current placeholder. More experienced VAXLSE users might also want to unerase placeholders (PF1 control-k ) and unexpand tokens (PF1 control-e).

### Terminology of VAXLSE

Before presenting a sample VAXLSE session, a few definitions are needed. The main concepts in VAXLSE are templates, placeholders and tokens. They are language constructs that define the syntax and grammar of the language.

- A *placeholder* is a marker inserted into the buffer by VAXLSE that "holds the place" of appropriate syntax of a language in a given context. It indicates a location in the source code where the user must provide additional program text by expanding the placeholder or typing over the placeholder.
- *Tokens* are words that are typed into the editing buffer by the user and expanded to provide templates for their corresponding language constructs.
- A *template* is any combination of text and placeholders which is brought into the editing buffer by VAXLSE in response to a user's expansion of their current placeholder. It is in fact a substitution of one word (the name of the placeholder) by several words and strings.

Placeholders are key elements of VAXLSE and require a little more explanation. Placeholders can be required or optional. For example, you may have a programming language where the second statement in a module clause is optional but the first is required. This can be represented with the appropriate VAXLSE symbols. There can also be list placeholders where either zero or more or one or more substitutions of the placeholder can occur.

The operations that can be performed on placeholders are

- directly type over,
- erase or
- expand the placeholder.

When the placeholder is directly typed over, the placeholder is deleted by VAXLSE and the text the user types is added to the current editing buffer. When the placeholder is erased, it is deleted from the buffer. One of three things can happen when a placeholder is expanded.

- If the placeholder is a *nonterminal* placeholder, it is replaced with a template consisting of more placeholders and possibly text, such as language keywords or punctuation.
- If the placeholder is a *terminal* placeholder, an explanatory description of the text needed for the placeholder appears in a separate window. The user types in the text, and the window and placholder is erased.
- If the placeholder is a *menu* placeholder, a menu appears in a separate window that the user can scroll through and then select the item that they wish to expand. The item can be a template that is text only, a placeholder which is expanded or a combination of the two.

The following sample VAXLSE session will highlight how to:

- Expand nonterminal and terminal placeholders.
- Delete placeholders.
- Type over list placeholders.
- Expand menu placeholders.
- Expand tokens.

After invoking VAXLSE by typing LSEDIT EXAMPLE.SAS, the initial placeholder {@sas_statement@} is placed in the buffer as shown below. The cursor is on the first 's'. The curly braces indicate that this is a required placeholder and the ellipsis (...) means that it is a list placeholder where more that one occurence of the placeholder may be substituted.

```
----------------------------------------
|{@sas_statement@}...                  |
|                                      |
|                                      |
|                                      |
|                                      |
|                                      |
|                                      |
|BUFFER EXAMPLE.SAS                    |
----------------------------------------
```

To expand the {@sas_statement@} placeholder, the user types control-e and sees this on the screen:

```
----------------------------------------
|{@sas_statement@}...                  |
|                                      |
|BUFFER EXAMPLE.SAS                    |
|->DATA_STEP : A SAS Data Step         |
|  {@procedure_statement@} : Procedure|
|  {@global_statement@} : Global stmt |
|  "ENDSAS;" : Ends a SAS session      |
|CHOOSE ONE OR PRESS HELP KEY          |
----------------------------------------
```

Since the placeholder {@sas_statement@} is a menu placeholder, a new window with the menu is displayed. Using the up and down arrow keys, the user can scroll through the menu and choose one of the items by hitting RETURN, control-e or ENTER when the arrow is pointing to the chosen item. If the DATA_STEP item is chosen and the RETURN key is pressed the following screen is displayed.

```
---------------------------------------
|DATA [@dataset_name@];                 |
|     [@datastep_statement@]...         |
|                                       |
|                                       |
|                                       |
|                                       |
|                                       |
|BUFFER EXAMPLE.SAS                     |
---------------------------------------
```

Note that the word DATA and the semicolon which is required in the SAS syntax has already been added by the VAXLSE template. The placeholder [@dataset_name@] is optional since it has square brackets. At this point the user can type control-k to erase the placeholder or control-e to expand it. If control-e is typed the following explanatory sentence is printed at the bottom of the screen.

```
---------------------------------------
|DATA [@dataset_name@];                 |
|     [@datastep_statement@]...         |
|                                       |
|                                       |
|                                       |
|                                       |
|BUFFER EXAMPLE.SAS                     |
|-- A string of letters and digits      |
---------------------------------------
```

When the user begins typing the dataset name, the placeholder is deleted by VAXLSE and the screen result is the following.

```
---------------------------------------
|DATA X;                                |
|     [@datastep_statement@]...         |
|                                       |
|                                       |
|                                       |
|                                       |
|                                       |
|BUFFER EXAMPLE.SAS                     |
---------------------------------------
```

After the user types the X the cursor is, of course, positioned immediately after the X. To continue to write the program, the user types control-n to go to the next placeholder then types control-e to expand the placeholder named [@dataset_statement@]. Note that this is an optional list placeholder. The control-e gives this result:

```
---------------------------------------
|DATA X;                                |
|     [@datastep_statement@]...         |
|                                       |
|BUFFER EXAMPLE.SAS                     |
|-> ASSIGNMENT : Assignment statement   |
|   INPUT : Input statement             |
|   IF : If statement                   |
|   DO : Do statement                   |
|CHOOSE ONE OR PRESS HELP KEY           |
---------------------------------------
```

If the user positions the arrow at IF and presses RETURN, the following screen results.

```
---------------------------------------
|DATA X;                                |
|     IF {@expression@} THEN            |
| {@datastep_statement@}        |
|     [@else_statement@]                |
|     [@datastep_statement@]...         |
|[@sas_statement@]                      |
|                                       |
|BUFFER EXAMPLE.SAS                     |
---------------------------------------
```

The placeholder [@datastep_statement@] was replaced by a template with the syntax for an if-then-else statement. Note that the placeholders {@expression@} and {@datastep_statement@} are required but the placeholder [@else_statement@] is optional.

For programmers who are experienced with most of the syntax for a language and view all the control key typing as combersome but who still want occasional help in remembering punctuation, tokens are provided. If the implementor of the VAXLSE environment file for a language has declared a string to be a token, the user can type the string and then press control-e. For example, if the user typed means, for the SAS procedure MEANS, and then typed control-e the following template for procedure MEANS would be inserted into the editing buffer.

```
---------------------------------------
|PROC MEANS [@means_options@]... ;      |
|     [@var_statement@]                 |
|     [@by_statement@]                  |
|     [@freq_statement@]                |
|     [@weight_statement@]              |
|     [@id_statement@]                  |
|     [@output_statement@]              |
|                                       |
|BUFFER EXAMPLE.SAS                     |
---------------------------------------
```

## The SAS Language and Grammar

### What is SAS and why does VAXLSE make SAS easier to use?

The SAS language is a programming language developed to be used with the SAS system, a software system for data management and analysis, statistical analysis, report writing and presentation graphics. There are two main parts of the language: the DATA step and the PROC (short for procedure) step.

The data step is used to create and manipulate flat data files and SAS datasets. It has similar statements to other high level languages like assignment, if-then-else and do-while-loop statements but the input and output statements are particularly easy to learn and use.

Once a SAS data set is created, SAS procedures analyze and process that data set by reading the data set, performing various calculations on the data and printing the results of the calculations. For example, the PRINT procedure reads your SAS data set, arranges the data values in an easy to read form and prints them. The MEANS procedure reads your SAS data set, computes the mean and other descriptive statistics, and prints those statistics or creates another SAS data set containing the results of the computations.

The DATA and PROC steps get their names from the SAS statements DATA and PROC, which start off the steps. For example, this DATA step

```
DATA TEST;
INPUT X Y Z;
CARDS;
1 2 3
4 5 6
```

begins with a DATA statement and creates a SAS data set. This PROC step

```
PROC PLOT;
 PLOT Y*X;
 TITLE 'PLOT OF EXPERIMENTAL DATA';
```

begins with a PROC statement and processes a data set.

As with most languages that have many features and functionality, some of the SAS DATA step syntax is hard to remember. VAXLSE can provide templates for this syntax. The syntax of the PROC step is usually straightforward but each procedure has many options. VAXLSE is particularly well-suited to prompting the user with a choice of strings for these options.

This is a small sample of some very high level productions of the grammar for the SAS language.

```
SAS_PROGRAM ::=          SAS_STATEMENT+

SAS_STATEMENT ::=        DATA_STEP |
                         PROCEDURE |
                         GLOBAL_STATEMENT

DATA_STEP ::=            "DATA"
  [DATASET_NAME]";" <DATASTEP_STATEMENT>

DATASTEP_STATEMENT ::=  IF_STATEMENT |
                 ASSIGNMENT_STATEMENT |
  INPUT_STATEMENT |
  PUT_STATEMENT |
                 DO_STATEMENT

PROCEDURE ::=     MEANS_PROCEDURE |
  PRINT_PROCEDURE |
 (etc. for each procedure in
   the SAS System)

MEANS_PROCEDURE ::=
  "PROC MEANS" [MEANS_OPTION]* ";"
  [@VAR_STATEMENT@]
  [@BY_STATEMENT@]
  [@FREQ_STATEMENT@]
  [@WEIGHT_STATEMENT@]
  [@ID_STATEMENT@]
  [@OUTPUT_STATEMENT@]
```

### How to Define a Language using its Context Free Grammar

One of the most useful features of VAXLSE is that it allows a customized application for a specific language to be developed, that is, you can extend an existing environment file for a language or create a new one. This feature can be applied in a variety of ways. One result could be that a particular user of C might wish to extend the existing C language template and add a placeholder for a template of a module comment header which can be expanded while using VAXLSE for C. The second, more powerful, customization for a user of VAXLSE is creating the templates for a new language from scratch and that is what the rest of this paper describes.

Before creating the VAXLSE language constructs it is important to become familiar with the language and how it is used. If the environment will be used mostly by users who are learning the language, then they will probably want many layers of placeholders to make the structure of the language as clear as possible. If it will be used by users who will often edit existing files, then many tokens are required. Also, it is necessary to be aware of how the grammar productions may be "collapsed" so that fewer VAXLSE keystrokes are required to fully expand a program.

Next, the creator of the VAXLSE environment should translate the context-free grammar into VAXLSE language definitions. This involves understanding how the context-

free grammar terminology relates to the VAXLSE language constructs. The following table illustrates my point.

```
Context-free grammar terms  VAXLSE terms
--------------------------  ------------
PRODUCTION where body of    NONTERMINAL
rule has at least one       PLACEHOLDER
nonterminal production

PRODUCTION with terminal    TERMINAL
as body of rule             PLACEHOLDER

TOKEN                       strings
                            within a
                            template

no equivalent                TOKEN
```

A production of a context-free grammar in which the body of the production contains at least one nonterminal can be thought of as a nonterminal placeholder in VAXLSE terminology. A production with only a terminal as the body of the rule is equivalent to a VAXLSE terminal placeholder.

A token returned by a lexical analyzer and used as a terminal in a grammar is a context-free concept which is not used by VAXLSE. This is because VAXLSE does not accept source and carve it into tokens, that is to say it does **not** parse. It instead is a tool which starts with the beginning production of a grammar and PRODUCES the defined templates associated with each nonterminal until no more nonterminals appear in the buffer. But you can think of the strings within the quoted text of a body of a placeholder as a type of token. The VAXLSE term token is confusing because the definition is not closely related to the traditional definition of token. It instead means that whenever you want to allow a user to type a word into the buffer and expand that word to a template or menu you must define that word as a token.

### VAXLSE Language Definitions

There are only three VAXLSE language definitions used to define a language. The commands are

- DEFINE LANGUAGE
- DEFINE PLACEHOLDER
- DEFINE TOKEN

The DEFINE LANGUAGE command has numerous qualifiers which can be used to describe the languge for which you are creating a new VAXLSE application The DEFINE PLACEHOLDER defines either a nonterminal, terminal or menu placeholder. The DEFINE TOKEN defines a token and the associated template. Several examples of these language constructs follow.

### Examples of VAXLSE statements

The following is the DEFINE LANGUAGE command used to define the SAS language. The qualifiers of most interest are the delimiters which indicate the optional and required placeholders and the initial string.

```
DEFINE LANGUAGE SAS -
    /FILE_TYPES = (.SAS) -
    /IDENTIFIER_CHARACTERS = -
"abcdefghijklmnopqrstuvwxyz
         ABCDEFGHIJKLMNOPQRSTUVWXYZ
         0123456789_$." -
    /PUNCTUATION_CHARACTERS = -
        ";',:()*-+/" -
    /OPT = ("[@","@]") -
    /OPTL = ("[@","@]...") -
    /REQ=("{@","@}") -
    /REQL=("{@","@}...") -
    /INITIAL_STRING= -
        "{@sas_statement@}..." -
    /TAB_INCREMENT = 4 -
    /COMPILE_COMMAND = "sas "
```

The following is an example of the definition for a required list placeholder that is a menu. Note that this placeholder is nonterminal.

```
DEFINE PLACEHOLDER sas_statement -
    /LANGUAGE = SAS -
    /DESCRIPTION = "Sas statement" -
    /DUPLICATION = VERTICAL -
    /TYPE = MENU
    "DATA_STEP" -
       /TOKEN -
       /DESCRIPTION="A SAS Data Step"
    "procedure_statement" -
       /PLACEHOLDER/NOLIST -
       /DESCRIPTION= -
    "A procedure statement."
    "global_statement" -
       /PLACEHOLDER/NOLIST -
       /DESCRIPTION= -
    "A global statement."
    "ENDSAS;" -
       /DESCRIPTION= -
    "Ends the SAS program."
END DEFINE
```

You can see how this directly correlates to the grammar production:

```
SAS_STATEMENT ::=       DATA_STEP |
                        PROCEDURE |
                        GLOBAL_STATEMENT
```

In general, a production with several OR clauses is a good candidate for a menu placeholder.

Taking one of these menu options and continuing with the language definition produces the following statement. This menu placeholder has every item defined as a token.

```
DEFINE PLACEHOLDER procedure_statement -
    /LANGUAGE = SAS -
    /DESCRIPTION= -
      "A procedure statement" -
    /TYPE = MENU
    "PRINT" -
       /TOKEN -
       /DESCRIPTION= -
      "The Print Procedure"
    "SORT" -
       /TOKEN -
       /DESCRIPTION= -
       "The Sort Procedure"
    "MEANS" -
       /TOKEN -
       /DESCRIPTION= -
       "The Means Procedure"
END DEFINE
```

Expanding the "MEANS" token into its definition gives this DEFINE TOKEN statement.

```
DEFINE TOKEN MEANS -
    /LANGUAGE = SAS -
    /DESCRIPTION = "Means procedure"
    "PROC MEANS [@means_option@]... ;"
    "     [@var_statement@]"
    "     [@by_statement@]"
    "     [@freq_statement@]"
    "     [@weight_statement@]"
    "     [@id_statement@]"
    "     [@output_statement@]"
END DEFINE
```

The quoted strings make up a template for the token MEANS so that when a user types MEANS and expands it, the template is inserted into the buffer. Continuing with the development of these definitions, we take the means_option placeholder that is within the template and define it to get the following definition.

```
DEFINE PLACEHOLDER means_options -
    /LANGUAGE = SAS -
    /DESCRIPTION = "Means options" -
    /DUPLICATION = HORIZONTAL -
    /SEPARATOR = " " -
    /TYPE=MENU
    "DATA={@dataset_name@}"
    "NOPRINT" -
         /DESCRIPTION= -
         "Do not print statistics."
     "MAXDEC={@number@}" -
         /DESCRIPTION= -
      "Max decimal places when printing."
     "VARDEF={@number@}" -
         /DESCRIPTION= -
   "Divisor used in calculating variance"
     "keyword"/PLACEHOLDER/LIST
END DEFINE
```

This definition is an example of a menu placeholder where the first, third and fourth items are a mixture of text and a placeholder, the second item is text only and the last item is a list placeholder. Defining the placeholder dataset_name produces the final example that I would like to look at.

```
DEFINE PLACEHOLDER dataset_name -
    /LANGUAGE = SAS -
    /DESCRIPTION = "Sas dataset name" -
    /TYPE = TERMINAL
"String of letters and digits starting
 with a letter."
END DEFINE
```

The placeholder dataset_name is a terminal placeholder with help text that is displayed when the user tries to expand it.

### Creating and using an environment file

After creating the file with VAXLSE statements, an environment file can be created by following these steps.

- Invoke VAXLSE using the file with the languge definitions by typing

  ```
  $ LSEDIT FILE.LSE
  ```

  where FILE.LSE is the file containing the definitions.

- Type control-z to place the cursor at the command line and then type

  ```
  LSE>DO
  ```

  to process the definitions.

- To save the language definition in an environment file type

  ```
  LSE>SAVE ENVIRONMENT NEWLANGUAGE
  ```

  This creates an environment file named NEWLANGUAGE.ENV.

- To use the environment file with VAXLSE you can either define a logical name LSE$ENVIRONMENT to point to a full pathname or invoke VAXLSE using the command **`$ lsedit /environment = pathname_of_NEWLANGUAGE`**.

### Conclusion

In conclusion, I'd like to encourage you to consider designing an application of a customized VAXLSE environment. I hope this paper will help you get started. The application will be useful and easy to use and the reward should be satisfied user's of your language.

# Automating a Software Development Environment

**Linda L. Craddock**
SAS Institute Inc.
Cary, North Carolina

### Abstract

Thorough source code management, bug tracking, regression testing, and performance coverage and analysis are all essential parts of software development. Since we develop several software products at our company, we decided that we wanted to automate all of these tools and integrate them into one software development environment.

Using a source code management system and bug tracking system developed in-house along with a complete regression testing system using the VAX/DEC TEST Manager (DTM) and the VAX Performance and Coverage Analyzer (PCA), we were able to automate our methods for software development testing and source code coverage and performance analysis; thus, ensuring software integrity.

### Introduction

To work toward our objectives of software integrity, we decided that the following issues needed to be addressed before project development could begin:

**Source development cycle requirements** How could we ensure compatibility of our software products between operating systems and source code levels?

**Automation needs** What utilities were needed during the course of our software development cycle?

**Automation tools** What types of tools should we use to address these needs?

**Tool Integration** How would we implement such a system into our current software development environment?

### Software Development Cycle Needs

In order to produce better software, we felt that we needed to adhere to more restrictions in our development cycle than we had in the past. Therefore, we decided that we should focus more attention on source code management, bug tracking, regression testing, and performance and coverage analysis on our software products.

### Automation Needs

Historically, our typical software development cycle per release takes at least a year to a year and a half; therefore, we needed to make sure that we covered both immediate and long-term automation needs.

We needed a source code maintenance utility that closely monitored source code activity at both the development and system level. I'll quickly review what I mean by "development" and "system" levels:

- At the development level, we needed a source code management system that allowed developers to access different levels of existing source code. They need the ability to "pull" or extract source code from one of the source code levels in order to perform fixes. They also need to be able to "push" or integrate their source code into fixes or new source code the source code level, referred to as the INTegration level.
- At the system level, we needed a Source Code Management System that allows the Source Code Manager to "push" or move entire levels of source code at appropriate time intervals.

  The source code levels that the Source Code Manager manages include: INTegration level—where all new source code or source code fixes are "pushed" or integrated into existing software products; TEST level—where the INTegration level is pushed to once it passes several regression tests; and MASTER level—where TEST is pushed to after regression testing and other operations are performed—this is the most stable or "frozen" level and is where our final software products eventually evolve from.

We also found that we needed a bug reporting system that would not only be used to effectively to report

software errors, but could also be integrated into our regression testing cycle. This facility would consist of a bug reporting mechanism that would also keep track of bug fixes. By performing both functions, bug verification and removal of bugs from the outstanding bug report would become an easier task for the Test Administrator.

It was also critical for us to obtain a regression testing system that would play an important role in the early detection of any new problems or regressions in any of our software products. Again, we needed to find a regression testing system that was flexible enough to run at both the development and system levels. During development, regression testing needed to become an essential step when testing new source code and/or testing software fixes before the source is pushed into the INTegration level to reduce the chance of introducing new bugs into our software. Additionally, if we could detect possible regressions between source code levels during the integration phase, we would prevent the introduction of bugs into more stable source code levels.

Finally, we decided that we needed to do more performance and coverage analysis on the source code that makes up our software products. By doing such analyses, we can be more confident of better performance and quality of our software products before they are sent out to the customer.

## Automation Tools

When looking for tools to use in our automated system, we decided that we wanted to use some tools already developed in-house and combine them with existing software development tools. Before choosing a tool, we took into consideration its ease of use and the degree of difficulty it would require to integrate with other tools. The tools chosen for automation included:

- XLIB (a tool developed in-house) for our source code management system
- SAS(R) software products for our bug tracking system
- VAX DEC/Test Manager (DTM) for our regression testing system
- VAX Performance and Coverage Analyzer (PCA) for performance and coverage analysis

### Source Code Management System (Using SDS Commands and XLIB)

Approximately three years ago, as the number of software products offered on the VAX continued to grow rapidly, we realized that we needed to upgrade our source code management system that consisted of several independent command procedures. These procedures required too much human intervention. We investigated several other tools that were offered by several vendors before deciding to write the tool in-house. Due to the fact that our eight software products consist of 4638 different source modules existing in 92 different directories—which translates to over 3,313,926 lines of code and produces 366 executable images, we felt that we could better customize a source code management system to address all the needs of such a large system rather than to try and supplement an existing tool. As you can imagine, when you are dealing with a system of this size, speed of execution for each update operation is of great importance and we were able to build a system that could handle the size of our software products as well as execute operations quickly and efficiently.

The source code management tool developed in-house, XLIB, is virtually transparent to the developer, because it is used in conjunction with "system development service" or "SDS" commands. The SDS commands specify what source management application is to be performed. These commands act as a "front-end" to each host system's source code management system. Since the SDS commands are a company-wide source management application, the functionality of all commands are the same across all host systems. Therefore, a developer can log onto any host system and issue an SDS command to perform a source management application and the operation would execute the same as it would on any other host system.

XLIB coupled with SDS commands, makes it possible to perform source code pulls and pushes across several source code revision levels and separate software product tracks. Developers are able to "pull" or extract source from any of the three source code levels but are only able to "push" source into the INTegration level. On the other hand, the Source Code Manager is able to perform any update operation at any level.

All source access is recorded in an audit trail. This information is later used to create status reports of the daily or weekly activity at the INTegration level. Additionally, we often refer to the audit trail when investigating a new bug that has surfaced in the INTegration level.

Another access feature that is essential, is "locking" or "checking out" source code when it is "pulled" from one of the source code levels. Therefore, when another developer tries to pull a piece of code that has already been checked out, the developer is notified that the file is currently locked and by whom in order to discourage simultaneous updates. This locking mechanism is sufficient for our environment because our developers have very specialized areas of expertise and the need for simultaneous update is minimal, yet requires some coordination effort.

Source push verification is achieved when the developer's push request has been successfully executed. XLIB compiles the source code modules and links any executable images associated with the update request. Thereafter, the VAX DEC/Test Manager is invoked and regression testing is performed. XLIB also checks for differences between the new source code and existing code at the INTegration level. If differences exist, XLIB archives the existing code onto tape. Only if all steps are successful, XLIB pushes the source code, object code, and image(s) into the INTegration level.

If any errors occur, no update to the INTegration level is performed and the developer is notified of the errors that occurred during the update phases.

Module management is performed by XLIB in conjunction with an in-house utility, BUILD. BUILD determines which header files, source code, object files and libraries are out of date and sends this information back to XLIB who then performs the necessary compiles and links to form a new, updated image. BUILD uses "build scripts" when determining what files need to be updated and are portable across all host systems at our company.

### Bug Tracking System (Using SAS(R) Software Products)

We chose to use the SAS/AF(TM) software product to produce an interactive full-screen application that could easily be used by developers on our system. Since the functionality of this bug tracking system is also similar to other tracking systems on other host machines; other host developers can log onto our host machine and report bugs found in the software on our machine.

The primary functions of the bug tracking system are reporting any bugs found in our software, tracking tests that exhibit these bugs, tracking bug fixes and removing bugs from the outstanding bug list.

When a bug is entered, the developer is required to enter certain information about the bug such as: a title and description of the bug, a test program that exhibits the bug's behavior, etc.

The information entered is then added to a master database. It is then the Test Administrator's duty to assign priorities to the bugs reported. Once this information is merged into the master database, SAS(R) is used to manipulate the data and generate various types of reports.

We currently distribute "portable" bug reports and "non-portable" bug reports. The portable bug reports are sent to developers on other host machines who are responsible for bugs found in the software on any host machine so the portable bug can be fixed and the fixed source code transported back over to the VAX. The non-portable bug report is distributed to the developers on the VAX since these types of bugs exist only in the host-layer of the software products and are specific to our host machine.

Furthermore, since this bug tracking system is used across several host machines, "transaction files" containing specific information about bugs entered, fixed or verified are transported (via tape) to other hosts on a periodic basis so their bug tracking systems can be updated to reflect when bugs have been entered, fixed or verified on the other host machines.

### Regression Testing System (Using the VAX DEC/Test Manager)

When looking into obtaining an automated regression testing system, we wanted an application that could be used both by the developers during development as well as become integrated into the source code management system, XLIB, to perform system-wide testing before pushing source code levels. We chose to use the VAX DEC/Test Manager because it was flexible enough to be used for both functions. It also provided us with the ability to have groups of tests pertaining to different needs.

In order to provide the developers as well as the Source Code Manager with a better method for running regression tests at any of the source code levels, we created a command that could be used to invoke DTM. The format of the command is given in Figure 1.

At the development level, developers use the SDSREGRESSS command when testing out their source code fixes or new source code development. The test results help the developer to determine whether or not the source code is ready to be pushed out into the INTegration level.

At the system level, DTM is used to run rigorous regression tests at different time intervals (daily, weekly, monthly, etc.). The SDSREGRESSS command is also used for this purpose when running all test groups that exist in a DTM library at a certain source code level in the form of a batch job. If any tests fail during the execution of the batch job, the Test Administrator is notified by mail so the regression can be investigated on the following day. By running regression tests at regular intervals at different levels of intensity, the Source Code Manager can better determine when a source code level is ready to be pushed to a more stable environment.

### Performance and Coverage (Using PCA)

Performance of our software is a growing concern of ours. We chose the VAX Performance and Coverage Analyzer (PCA) because it enabled us to check the performance of our software and its affect on system resources; to check coverage of our source code in our regression testing; and to generate statistics for a particular release of our all software products.

We currently use PCA to check the software products' affect on important system resources such as CPU usage, I/O operations, and page faulting. This is to help us to determine which parts of our software products require more fine tuning before being shipped out to customers.

We also use PCA to check the coverage of our source code in the tests in our regression testing system to ensure that all source code in our software products have been exercised thoroughly. As a result, we can be assured that our regression testing system is an effective tool in our software development process.

At the end of each software development cycle, we use PCA to generate final performance statistics on all software products. These statistics are compared to the previous release to check for any possible performance regressions.

To make life easier for our developers, we have created a command in-house, SASPCA, that helps them to collect information about the performance and/or coverage of their source code without having to deal with the

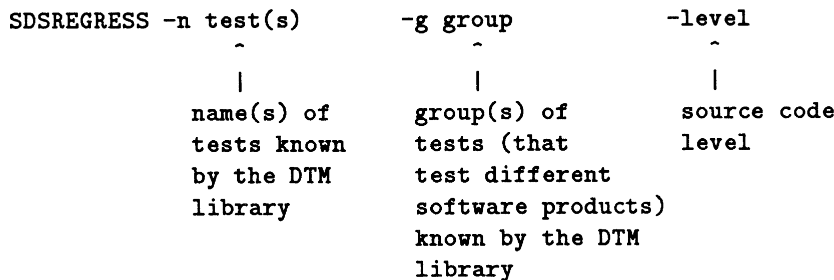



**Figure 1:** Using the SDSREGRESS command

the compiling and linking operations required to use PCA.

We are currently working on adding PCA as an option for the `SDSREGRESSS` command. This command will make it easier for developers who do not have experience with either DTM or PCA to obtain test results and information about the performance and/or coverage of their source code.

## Tool Integration

After getting all automation tools in place, we were able to create an automated system that was able to serve our needs at the development level as well as at the system level. We'll take a look at how we use this system at both levels.

### Development Level Activities

Development level activities are performed by software developers on our system and typically include activities such as:

- reporting a bug
- fixing the source
- pushing the source code into the INTegration level

*Reporting a Bug*

- A user on the system invokes the Bug Tracking System and enters the following information:
  - title describing the bug
  - a description of the bug
  - a test program exhibiting the behavior of the bug
- After the user exits the Bug Tracking System:
  - the information is added to the master data base
  - the test program is copied to the "test bed" area
  - mail is sent to the Test Administrator and assigned support person

*Fixing the Source*

- The assigned support person moves to his/her work area and:
  - Issues the `SDSPULL` command which requests that source code is pulled (INT is searched first, then TEST, then MASTER - to ensure that they get the most recent version of the source). Once the source code is successfully pulled (and not locked by someone else), the source code is copied to the work area and notification of a successful pull is issued. The developer is then responsible for:
  - Running the program (located in the "test bed") to determine where the bug is occurring.
  - After making the source code fix, the developer issues the `SDSBUILD` command which compiles the source code and links the necessary executable images. Once the BUILDing process is completed, copies of the object code and executable images are placed into the work area.
  - The developer then submits the `SDSREGRESSS` command to invoke DTM and run regression tests. After testing is done, the DTM process sends mail back to the developer about the status of the test run.

*Pushing Source*

- The developer then issues the `SDSPUSH` command which involves running the following steps:
  - The source code is run through a filter mechanism to check the syntax of the program and ensures that coding standards are not violated.
  - The source code is compiled with DEBUG so that development debugging is enabled
  - Executable images affected by the source push are linked.
  - The DEC/Test Manager is invoked to run the test collection (determined by the software product class specified in the `SDSPUSH` command).

- The new output results are compared against DTM benchmarks:
  - If successful, XLIB copies source code, object code, and executable images into the INTegration level and mail indicating SUCCESS is sent back to the developer
  - If unsuccessful, XLIB does not perform the push and information about the problems that occurred during update is mailed to the developer.

- Once the push is complete, the developer invokes the bug tracking utility and gets into the FIXED panel and marks the bug as fixed. By entering the bug as fixed, the Bug Tracking System sends mail to the Test Administrator about the fix. It is then the Test Administrator's responsibility to re-run the test (out of the "test bed" area) and verify that the bug is fixed.
  - if the bug is fixed, the Test Administrator creates a new benchmark for the test and inserts it into the DTM library; thus, completing the entire development cycle
  - if the bug is not fixed, the Test Administrator contacts the developer about the results and changes the status from FIXED back to OUTSTANDING.

**System Level Activities**

The Source Code Manager in coordination with the Test Administrator, run the system level activities that affect an entire levels of source code. These activities involve regression testing to check the stability of a level; linking of executable images; pushing one level to the next, etc. and are run at daily, weekly and "pre-change day" intervals.

*Daily activities*

Certain activities are performed daily so pushes to the INTegration level are reflected in the software on the following work day. Running regression tests is essential in order to ensure that all pushes made during the day do not "break" other parts of the software products.

A daily list of activities include:

- Running batch jobs that re-link base images at the INTegration level
- Running all regression tests included in the DTM library
- Comparing all new output results against DTM library benchmarks. If any regression tests failed, the Source Code Manager and Test Administrator are notified so the problems can be investigated on the next day.

*Weekly Activities*

At the end of each week during the development cycle, we attempt to push the INTegration level to the TEST level. This push integrates new development and source code fixes into a more stable environment. Since in-house testing is often performed at this level, we must:

- Run several intensive regression tests on ALL software products at the INTegration level in order to make sure that no regressions have occurred.

  INTegration level will NOT be pushed to TEST if the new output and benchmarks do not match.
- All INTegration level files are then copied to TEST level
- All executable images are relinked
- Regression tests are re-run to ensure that the new files from the INTegration level combined with the existing TEST level source code have not produced any new bugs.

  If any problems have occurred and are determined insurmountable, the old TEST level is restored from the archive tapes.

*Pre-Change Day Activities*

Pre-change day activities only occur when a very stable TEST level exists. Before TEST is pushed to the "frozen", MASTER level, the DTM library test suites are executed once again to determine whether or not the push should be performed. If tests are successful:

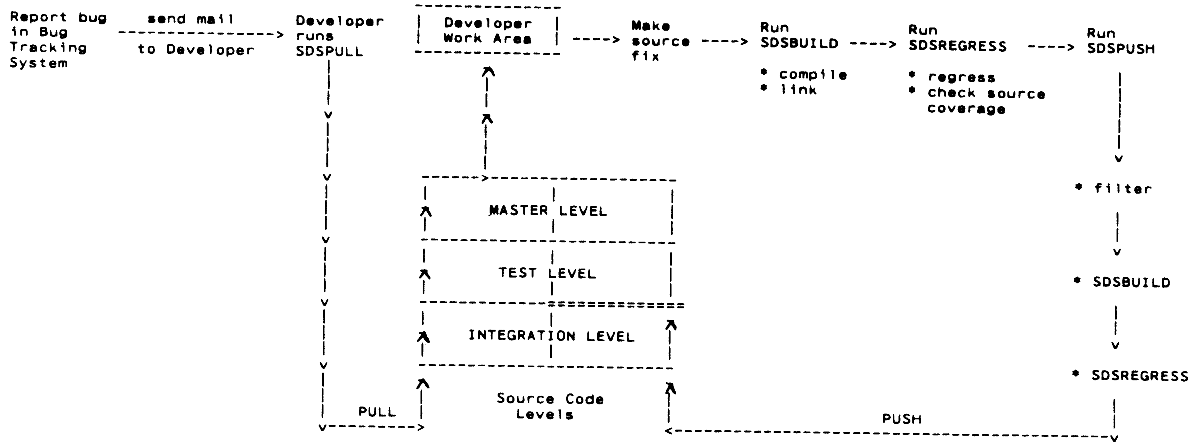
- MASTER level is archived
- TEST level is copied to MASTER Level, then:
- All source is compiled without the DEBUG option.
- All executable images are re-linked.
- DTM is invoked again to run all test suites to check the stability of the MASTER level to see if any problems have surfaced from compiling all source without /DEBUG.

  If any regressions occur, the Test Administrator and Source Code Manager are notified about the problem(s).
- Batch jobs are then submitted to generate PCA statistics regarding our software's performance and effect on system resources. These statistics are later compared against the previous release's statistics in order to check for any performance regressions. Once we are finished with these statistics they are archived for later use.

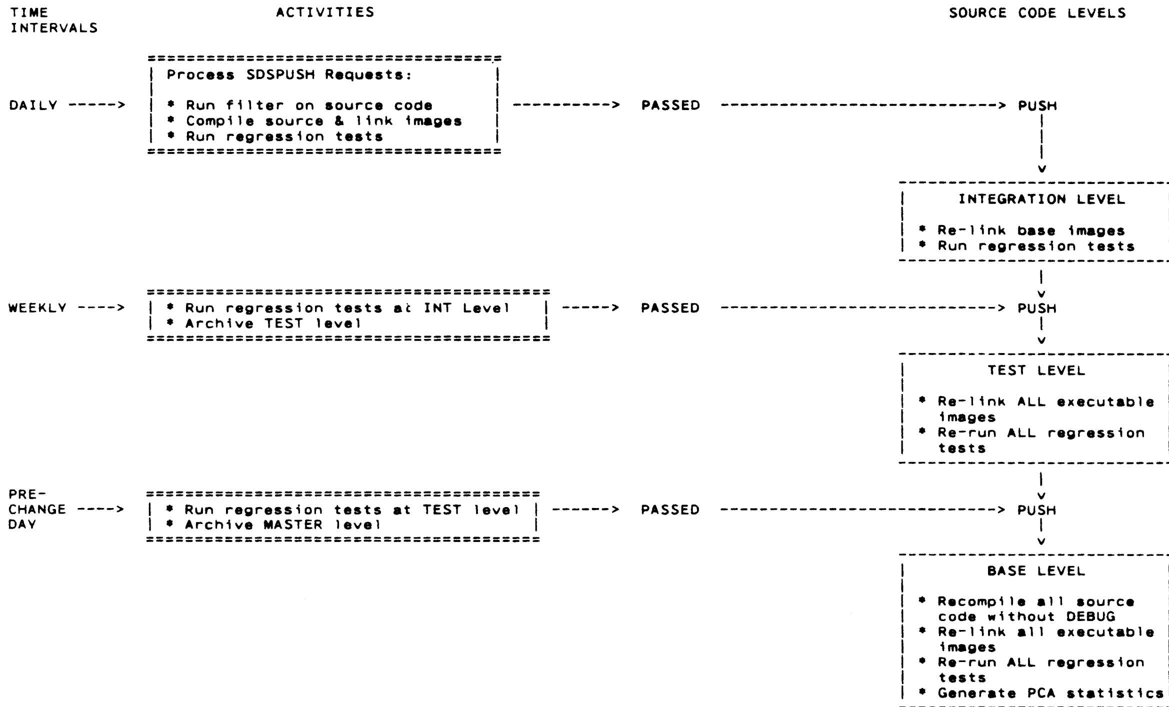
## SUMMARY

We have successfully achieved system integrity by ensuring the completeness of our source code management, bug tracking, regression testing and performance and coverage analysis system. Through integration of new and improved automation tools, we have been able to develop and maintain a more efficient software development cycle. Our company not only benefits through the use of our automated environment, but also ensures that our customers receive reliable software products consisting of better quality and performance.

TOOL INTEGRATION
AT THE DEVELOPMENT LEVEL




TOOL INTEGRATION
AT THE SYSTEM LEVEL



# LARGE SYSTEMS SIG

# The Internet Domain Name System

**S. Robert Austein**
Massachusetts Institute of Technology Laboratory for Computer Science
Cambridge, Massachusetts

## Abstract

The rapid growth of packet switched computer networks in recent years has created an environment where the mere task of locating a particular network host (computer) or user presents a significant challenge. On a large network such as the INTERNET it is no longer practical to maintain a single comprehensive table of all the hosts on the network, let alone all the users.

The *Domain Name System* (DNS) is a collection of concepts, algorithms, and protocol specifications which address this problem. The DNS is part of the TCP/IP protocol suite, and the concepts and architecture of the DNS are gaining a foothold in other network communities such as BITNET and USENET.

This paper presents an overview of the DNS, along with notes on some lessons learned in developing an implementation of the DNS.

### The problems that the DNS attempts to solve

Back in the early days of computer networks it was a relatively straightforward task to maintain a comprehensive table of all the hosts on a network. In an environment of, at most, a few hundred hosts, a table consisting of one entry per host was quite tractable, and weeks or even months could go by without anyone needing to make a change to the table.[1]

Since the adoption of the TCP/IP[2] protocol suite, however, it has been possible to hook an entire *network* onto the INTERNET with little more effort than it used to take to hook up a single host. This ease of connection, combined with the trend towards large numbers of minicomputers and workstations instead of a small number of mainframes, has caused the INTERNET to grow at a rate surprising even to its designers. It is no longer feasible to maintain a comprehensive host table for the INTERNET.[3] Quite apart from the human suffering involved in attempting to maintain such a table, the scale of the problem effectively guarantees that, by the time the central authority can verify all the entries in the table, enough entries will have become obsolete to render their efforts largely useless. To make matters even worse, mathematical analysis of the way in which people use host tables indicates that an inordinate amount of CPU time and network bandwidth will be consumed merely attempting to retrieve and examine a comprehensive host table for a network of this size; see [RFC-1034] for details.

### Other problems of scale

While the problem of host table maintenance and distribution was the driving force behind the development of the DNS (in the sense that it was clear that something would have to be done about the host table problem in relatively short order), it was by no means the only problem of scale that the INTERNET was facing. One particularly noticeable[4] problem was the increasing complexity of the addresses used in electronic mail messages. From its humble origin as an amusing gimmick of an obscure operating system at the Dawn of Timesharing, electronic mail has grown to become a major network and social phenomenon. Indeed, to many users, the "network" consists of those sites reachable by electronic mail; other network services, while useful, pale in significance beside this single service.

Just as different network communities have developed differing low-level protocols, different electronic mail communities have developed differing electronic mail protocols and addressing schemes. Unlike lower-level protocols, how-

[1] In prehistoric times, when the *ARPANET* still used the NCP protocol suite (pre-1982), it was even possible (or at least conceivable) to keep a list of all the authorized *users* on the network.

[2] "TCP" and "IP" stand for "Transmission Control Protocol" and "Internet Protocol," respectively. See [DDN 85] for more information on TCP/IP.

[3] The staff of the DDN Network Information Center (NIC) attempt to do so anyway for the benefit of hosts which have not yet been taught how to use the DNS, but it is a losing battle. As this article goes to press, the NIC host table has ≈ 6500 entries; MIT alone has ≈ 1700, many of which are not in the NIC table.

[4] Noticeable, that is, to the "users."

ever, there is no conceptual difficulty involved in having a host that sits on two different networks relay electronic mail messages across the boundary. In practice, for the sake of mutual sanity, the electronic mail message header formats on the various networks that can communicate via electronic mail relays have tended to converge towards a *de facto* standard which closely resembles but is not identical to the prevailing INTERNET standard. The principle problem here was the difficulty involved in constructing and deciphering the "address" fields, those fields which name the originator and recipient(s) of the message. As the "network" of sites reachable by electronic mail grew, the complexity of these addresses placed them beyond the grasp of the non-specialist.[5] Remedying this situation called for a naming convention which could be used on both sides of the relay boundaries and a protocol that could be used to hide the complexity involved in using the relays from the end users.

Yet another problem on the horizon, related in that it involved problems of transparently re-routing electronic mail, was the growth of environments containing large numbers of workstations with little or no provision for filesystem backup. The user of such a workstation might legitimately want to participate in electronic mail exchange but presumably would not want her mail to be stored on a filesystem with poor backup service; rather, she would like to have any mail addressed to her workstation re-routed to some central repository. Some mechanism was needed to allow the various electronic mail agents to handle such cases intelligently, again without forcing the typical user to pay much attention to the details of how this happens.

### What these problems have in common

While the problems of excessive host table size and transparent electronic mail routing may, at first, seem unrelated, they have common features. Both are essentially the problem of constructing and managing a large-scale database, with some special constraints:

- The primary database key (the "name") must be simple enough for users to remember, but must contain enough information for the database software to answer queries both promptly and correctly.
- The names used should hide unnecessary detail from the users.
- The database should look like a single name space from the users' point of view.
- The database must be highly available: failure of a handful of computers should not cripple the entire network.
- The system should be reasonably reliable; it is permissible for there to be transitory inconsistancies while updates propagate, but the database should converge towards a consistant state.
- The system must scale up well, to meet the needs of a rapidly growing network community.
- The system must allow for extension to meet new needs.
- The system must allow administrative control of portions of the database to be delegated to the administrative entities best suited to maintaining them. No single administrative entity should have the impossible task of maintaining a comprehensive table.

### How the DNS addresses these problems

The DNS is the result of several years of development and experience with one of several schemes that were proposed to address the problems discussed in the preceding section; see [RFC-1034] for more information on the antecedents of the current specification. It is neither possible nor desirable to completely describe the DNS in a paper of this length; if you want all the details, see [RFC-1034], which discusses the general design of the system, and [RFC-1035], the protocol specification. For the present purpose, it will suffice to present the broad operation of the system and describe how it simplifies life on the network for users and administrators.

### Structure of Domain Names

The DNS uses little-endian,[6] tree-structured names, with "." characters delimiting the fields within the name. For example, "`Loki.CC.Miskatonic.EDU`" names a machine locally known as "`Loki`" at the Computer Center of Miskatonic University, which is an educational institution.[7]

There are several noteworthy points about domain names. First, all domain names have an invisible suffix: properly speaking, there are five fields in the name "`Loki.CC.Miskatonic.EDU`," the last being the invisible field (to the right of "`EDU`") denoting the root of the naming

[5] The tendency has been to use some form of explicit routing, whether in the (unofficial) INTERNET "`%`" syntax such as "`user%host1%host2@host3`" or in the USENET "`!`" convention as in "`host3!host2!host1!user`." The inherent difficulty of this scheme arises when one attempts to parse an address such as "`name1!name2%name3@name4`." Depending on the rules used to parse such an address this might be interpreted as mail for user **name2** at host **name1** via **name4** and **name3**, or as mail for user **name2** at host **name3** via **name1** and **name3**. Clearly, more perverse examples can be constructed readily.

[6] The term "little-endian" is borrowed from the terminology used to label byte-ordering schemes. It refers to the semantics of domain names—the leftmost field of the name is the least significant field of the name while the rightmost field is the most significant. In other words, the name starts at the "little end," hence the term "little-endian."

[7] See [RFC-920, RFC-1032, RFC-1033] for a discussion of the administrative constraints on choice of domain names; see [RFC-1035, pp. 7–8] for the technical considerations.

tree.[8] Second, being tree-structured, this naming scheme is recursive: "EDU" has a child node "Miskatonic.EDU" which in turn has a child node "CC.Miskatonic.EDU," and so forth. Last, the choice of names follows *organizational* boundaries; the names hide irrelevant detail while emphasizing detail that may aid users in remembering the names. The assumption is that the mythical "average user" is more likely to remember an electronic mail correspondent as

> Fred Dorf, who works at at the Totally Winning Software Company

than as

> Fred Dorf, whose home machine MUMBLE can be reached via relay by host BUMBLE on the FROTZNET.

The choice of organizational boundaries in the naming scheme also has implications for the delegation mechanism, discussed below.

### Delegation of Authority in the DNS

As previously mentioned, it is absolutely vital that the DNS have a mechanism for breaking the namespace up into reasonable sized administrative chunks. These administrative chunks are referred to as "zones of authority," or just "zones." A zone is a subtree of the naming tree, delimited:

- At its base (root of the subtree) by a node containing a special record called a "Start Of Authority" record.
- At its leaf nodes.
- At branches containing a record delegating authority for the name at that branch to some other zone.

The administrator of a zone is said to have *authority* over the name of the zone and any children thereof.[9] The zone administrator can, at her option, create and destroy records associated with names that are children of the zone name, and create and destroy database records associated with those names. One particular kind of database record that the administrator can create is a record which *delegates* authority for the name at which the delegation appears to another zone.[10] Such a delegation passes authority from the parent zone to the child: the parent cannot claim to be authoritative for any name within the child zone.

The zone is the basic unit of administration in the DNS. Since the DNS was developed on the INTERNET, original authority rests with the DDN Network Information Center (NIC), by fiat; the NIC is authoritative for the root zone. When the NIC receives a registration request from an organization that is willing and able to administer its own portion of the tree, the NIC delegates that portion of the tree to the requesting organization; for example, the NIC might delegate authority for the Miskatonic.EDU subtree to the Telecommunications office of Miskatonic University, thus giving administrative control for the name "Miskatonic.EDU" and any children thereof to the organization most likely to know what data should in the portion of the database representing Miskatonic University. Note, however, that the NIC has the option of revoking the zone delegation at a later date if, in the NIC's opinion, the delegatees are not administering their zone properly; in practice this would only happen if more polite attempts by the NIC to remedy the situation were unsuccessful.

The process of delegation need not stop at one level. To continue the example, Miskatonic University is a large organization in its own right, containing many organizations (departments, laboratories, and so forth) which may or may not be large enough to present an administrative challenge. Some of these organizations, such as the computer center, may want to administer their own portions of the database; some, such as the payroll office, may prefer to leave this task to the Miskatonic.EDU zone administrators; some, such as a fraternity with a collection of personal computers on the campus network, may not be considered technically competent or responsible enough to act as administers. The Miskatonic.EDU zone administrators can chose to delegate authority in the first case while retain control in the second and third cases. Thus, the Miskatonic.EDU zone administrators can delegate the name CC.Miskatonic.EDU to the Miskatonic University Computer Center, allowing the Computer Center to create names like Thor.CC.Miskatonic.EDU and Loki.CC.Miskatonic.EDU constrained only to their own judgment, retain control of the name Payroll.Miskatonic.EDU, insuring that only Telecommunications Office staff can modify the database records describing the host Payroll.Miskatonic.EDU, and retain control of the name Frat.Miskatonic.EDU, forcing the members of the $\Phi\Omega\Omega$ fraternity to obtain the approval of the Telecommunications Office before creating names like Beer.Frat.Miskatonic.EDU and Brew.Frat.Miskatonic.EDU. This process of breaking administration down by organization considerations goes on at each level, until things finally come to rest with, we hope, everyone except the downtrodden $\Phi\Omega\Omega$ fraternity reasonably happy.

Thus the DNS provides a flexible mechanism for parceling out control of the database as needed, while retaining a "chain-of-command" structure useful in handling

[8] The DNS, as specified, is not an "explicitly-rooted" naming scheme. This turns out to be somewhat inconvenient when attempting to design a friendly user-interface. An explicitly-rooted syntax is used in certain parts of the DNS ("Loki.CC.Miskatonic.EDU" becomes "Loki.CC.Miskatonic.EDU."), but the canonical form of domain names in electronic mail messages is not the explicitly-rooted form.

[9] This is not quite true: see discussion of the CLASS attribute, below, for details.

[10] Properly speaking, the delegation specify the *hosts* that provide authoritative name service for the designated zone.

the occasional incompetent or malicious administrator. It is important to remember that this division of the database into zones is only visible to the administrators; it is, and should be, of no concern to the naive user.

### Resource Records, Name servers, and Resolvers

Conceptually, there are three major components of the DNS[11]:

- The database, a set of *Resource Records* (RRs). RRs are organized into zones, as described in the preceding section, but this is usually invisible to the end user; the user sees a single, comprehensive database.

- *Resolvers*, programs which search the database for RRs, on behalf of users. The user communicates to the resolver (via a local user interface outside the scope of the DNS specification) what RRs the user wants to find; the resolver initiates one or more transactions with one or more name servers in order to retrieve the desired information or determine that the query cannot be answered with the information currently available.

- *Name Servers*, programs responsible for dispensing RRs, in response to queries from resolvers, on behalf of zone administrators. Name servers are passive; they answer queries put to them, if possible, but do not initiate transactions other than occasional maintenance operations. Zone administrators make the information in their zones available by loading it into name servers running on the hosts to which the parent zone has delegated authority; this is the only way that new information enters the database.

The conceptual distinction between resolver and name server is sometimes blurred in practice; implementors may chose to combine both functions into a single program, and there are some optional features in the DNS specification that may make it difficult to keep track of which program is currently acting in what role. Nevertheless, the basic relation between these three components is simple: resolvers obtain resource records from name servers. Each of these components are discussed in slightly more detail, below.

*Resource Records*

A RR is a data structure with several attributes. One of these, the NAME, is already familiar. The next two are small integers called CLASS and TYPE. TYPE specifies the kind of resource described by this RR, such as one network address of a host or the name of a host providing name service to a zone. CLASS provides a mechanism for creating parallel trees in the DNS in cases where differing assumptions about data formats for RRs with the same NAME and TYPE need to be made explicit, such as the network address of a host under different protocol suites.[12]

The triplet (NAME, CLASS, TYPE) completely specifies a RR; the queries sent out by a resolver consist of such a triplet and not much else. RRs may (and do) exist which have identical NAME, CLASS, and TYPE values but differ in the data portion of the RR, but, since they match the same triplet, if any one of them is returned by a query, they will all be returned by a query. The DNS uses this to represent "one-or-more" values such as host addresses or members of a mailing list.

The remaining attribute common to all RRs is the "Time-To-Live" (TTL). Given the scale on which the DNS is intended to operate, tracking down a particular set of RRs will probably be a fairly expensive operation in the general case, so a caching mechanism is provided. When a name server sends an RR to a resolver in answer to a query, the resolver has the option of keeping a copy of that RR for the period of time specified by the TTL attribute and using that cached copy to answer user requests rather than undertaking another conversation with the name server. Note that the value of the TTL attribute, like the rest of the RR, is under the control of the administrator of the zone whence it came; thus, the zone administrator has some amount of control over how often resolvers think it is necessary to query her zone's name servers.

The rest of the RR is the resource data (RDATA). This is the payload that the rest of the system transports. The format of the RDATA portion is dependent on the CLASS and TYPE values; it may consist of zero or more integers, domain names, text strings, or bitvectors. In the message format used in conversations between resolvers and name servers, the RDATA field is preceded by a byte count. A resolver need not understand the format of the RDATA attribute of every RR it handles; it can preserve RRs of unknown CLASS and TYPE unchanged.

For a detailed description of the existing RR TYPEs and formats, see [RFC-1035]. For this presentation it will suffice to list the general kinds of data that are currently defined:

- *Zone information:* RRs describing the zone structure, listing electronic mail addresses of administrators, parameters used in automatic name server maintenance operations.

- *Host information:* RRs describing a host's network addresses, hardware/software type, and supported network protocols.

- *Mail Agent information:* RRs describing how to send mail to a specific "mail agent," essentially a virtual

[11] The presentation in this section closely follows [RFC-1034, pp. 6–7], *q.v.*

[12] Readers interested in further details should see [RFC-1034, p. 12] and [RFC-1035, p. 13], but should not expect that that doing so will make everything crystal-clear. A more detailed discussion of then-current ideas on CLASS was present in [RFC-883, pp. 8–10], concluding "[the] concepts of CLASS, recursive servers and other mechanisms are intended as tools for acquiring experience and not as final solutions."

hostname that may or may not correspond to an a network host by the same name.

- *MailBox information:* RRs describing how to send mail to a particular "mailbox," such as a user or a mailing list. The current specification of mailbox RRs is known to be inadequate and will almost certainly need revision before it can be used effectively.[13]
- *User information:* The *Hesiod* ([Dyer 87]) system uses the DNS to support distribution of publicly-readable user information, such as the encrypted form of a user's password, in an highly distributed workstation environment.

Experimentation with storing new kinds of information in the DNS is actively encouraged by its designers, provided that such experimentation takes the correct steps to avoid distressing other users of the system. The listing above should be viewed as a snapshot of known uses the DNS as this article goes to press, with the understanding that new TYPE and CLASS parameters may be added in the future; the TYPE and CLASS parameters used by *Hesiod*, for example, are recent additions to the DNS protocols.

*Name Servers*

Name servers exist to answer queries from resolvers. There are several steps a name server might take in composing its answer:

1. The name server might look in its locally loaded database, if any.
2. The name server might ask a resolver running on the same machine as the name server to look in its cache for the answer.
3. The name server might ask a resolver running on the same machine as the name server to attempt to resolve the query.
4. The name server might decide that it is unable to answer this query.

Implementations may fold together several of these steps. Support for the first and last steps are required in all name server implementations to insure that the name server will be able to answer any query it receives, at least to the extent of an explicit reply indicating that it is unable to find a useful answer. Step 2 is optional: RRs obtained this way are not considered authoritative. Step 3 is also optional, and is referred to as a *recursive query*.

Recursive queries allow the implementation of very simple resolvers for applications such as a memory-bound personal computer; the resolver on the deficient machine simply sends all its requests off to one or more local name servers known to support recursion, and takes whatever answers these servers return as the final answer to the query. There are some restrictions on the use of recursive queries: of particular interest, the resolver acting on behalf of the name server that received the recursive query is *not* allowed to issue another recursive query. Without this restriction, a single recursive query could trigger an exponential progression of subsequent recursive queries until the network ground to a halt due to lack of resources.

Besides answering queries, there are some maintenance operations associated with running a name server. Depending on the implementation, these operations may or may not be handled by the same program that handles queries. Currently, maintenance operations consist of obtaining complete copies of zones and discarding old copies of zones that have expired.[14] This mechanism allows a zone administrator to configure one of her name servers as the "primary" server, and the others as "secondary" servers: with such a configuration the administrator need only update the primary server, because the secondary servers will periodically ask the primary if the zone has changed, and, if so, the secondaries will obtain and install the new version of the zone. The parameters controlling how often these checks and updates happen are attributes of the Start-Of-Authority RR that defines the base of the zone, in keeping with the philosophy that all the "magic numbers" in the DNS should be under the control of the appropriate zone administrators.

*Resolvers*

Resolvers are the programs that attempt to track down RRs in the DNS in response to user requests. Resolver implementations can vary widely in complexity. The simplest implementation, mentioned above, translates the user's request into a recursive query, and sends it off to some name server which (we hope) offers recursive service. Such a resolver is sometimes referred to as a "stub resolver;" we shall refer to a resolver capable of processing requests without having to resort to a recursive query as a "full-service resolver."

A full-service resolver is a fairly complex program. Fortunately (for the reader, not the implementor), much of the complexity involves internal bookkeeping and the handling of obscure error conditions. The basic operation of a full-service resolver is straightforward.

1. The resolver receives a user request; we assume that some user interface package has already translated the user's original request into a (NAME, CLASS, TYPE) triplet.
2. The resolver looks in its internal database (loaded zone files and cached responses from previous queries) to see if it can answer the request; if so, it does.

[13] This is due to the same problem that resulted in the **MD** and **MF** TYPEs being replaced by the **MX** TYPE; see [RFC-973, p. 4] for the reasoning behind that decision.

[14] There has been some discussion of extending the DNS protocols to support partial zone updates, but nothing concrete has emerged to date.

3. The resolver looks in its internal database to determine which of the name server(s) it knows about is most likely to be able to answer the query. Conceptually, the lookup starts at the root of the global database and proceeds down towards the target name.

4. The resolver attempts to contact one or more of these name servers and pose the query to it. If too long goes by without an answer, the resolver gives up and returns a timeout error to the user.

5. If the name server definitely answered the user's question, either successfully or with an error indicating that the requested RRs do not exist, the resolver returns the answer or error to the user. If analysis of the name server's answer detects an error (for example, a name loop caused by a circular chain of nickname pointers) has occurred, the resolver returns an error to the user.

6. The name server may have been able to supply a referral to other name servers that the responding name server thinks might be able to answer to the query. If so, and if the resolver agrees that this is a useful referral, the resolver changes its list of name servers to reflect this referral, and loops back to step 4.

7. If steps 5 and 6 do not apply to the response the name server sent, the resolver removes this name server from its list of relevant name servers, and loops back to step 4. If the resolver reaches this step with no name servers left to contact, there is something seriously wrong with either the resolver or the name servers it has been talking to, so the resolver returns an error to the user.

The reader is now in a position to understand why nonrecursive queries are also known as *iterative* queries.

Several points about the algorithm for iterative query processing are worth mentioning.

- Assuming no blatant errors in the programs or database, this algorithm will be able to locate any desired RRs, if they exist, by walking down from the root of the tree. One piece of information conspicuous by its absence in the above discussion is just how the resolver goes about locating the root. The answer is that the addresses of the root name servers (or the addresses of name servers that can supply the addresses of the root name servers) must be supplied to the resolver from a configuration file or some other source outside the DNS itself, when the resolver boots.

- Any software that attempts to make intelligent use of the DNS must be prepared to handle a soft error due to a query timeout. It may be that a particular application can safely treat soft and hard error identically, but is by no means a foregone conclusion. Unfortunately, many existing network programs make the implicit assumption that only success and hard error are possible. This problem will presumably cure itself in the long run.

- This algorithm can take a long time to run to completion, and there is a tradeoff involved in selecting the timeout. If the timeout is too short, some queries will become unanswerable because the servers involved will never be able to respond quickly enough to suit this impatient resolver. If the timeout is too long, applications (and their users) will waste a lot of time waiting for the resolver to give up on a query that is "obviously" a lost cause. See the section on the CHIVES system, below, for one way of handling this problem.

### Transport protocols

The DNS is intended to be useful in more than just the TCP/IP world of the INTERNET. The message protocols can be used over any reliable 8-bit byte-stream protocol. They can also be used over any (possibly unreliable) datagram protocol. On the INTERNET, the DNS protocols operate over the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP). All transactions consist of a single query message followed by one or more response messages. When using UDP, messages correspond to one datagram packet; when using a byte-stream protocol such as TCP, the first two bytes of the message are a 16-bit integer indicating how many bytes follow.

Implementing the DNS protocols over any reliable byte-stream protocol should be straightforward. Implementing the protocols over a message-based protocol such as is used on the BITNET would probably be possible; the principle difficulty would be the long round-trip time between BITNET messages, not any intrinsic difficulty with the protocols.

## Electronic mail support in the DNS

As mentioned briefly, above, the DNS has two kinds of RRs that support electronic mail. Only one of these is currently in use; this is *mail agent* level electronic mail forwarding. It works as follows: when the program responsible for actually sending electronic mail messages out over the network (the "mailer daemon") receives a message with an address such as "`Cthulhu@Miskatonic.EDU`," it looks for a mail agent RR with name "`Miskatonic.EDU`;" if it fails to find such an RR, it proceeds in the usual fashion, looking for a network address for host "`Miskatonic.EDU`" and attempting to send the message there. If, however, the mailer daemon does find a mail agent RR, it sends the message to the host named in the RDATA portion of the RR.[15] The RDATA portion of the mail agent RR also contains a small integer, used to rank these RRs if more than one exists for the same name.

This simple mechanism provides several benefits:

[15] See [RFC-974] for a more detailed discussion of the algorithm.

- It allows the use of DNS style hostnames for hosts that are not on the network. Suppose that **Loki.CC.Miskatonic.EDU** is not connected to the network but exchanges mail with **Odin.CC.Miskatonic.EDU** via a dialup protocol. If the **CC.Miskatonic.EDU** zone has a mail agent RR named **Loki.CC.Miskatonic.EDU** with an RDATA field indicating **Odin.CC.Miskatonic.EDU**, electronic mail from the outside world to **Loki.CC.Miskatonic.EDU** will automaticly be routed to **Odin.CC.Miskatonic.EDU** for eventual forwarding to **Loki.CC.Miskatonic.EDU** with no special effort on the part of the users.

- It allows the creation of "maildrops," a sort of virtual host. There may not be any physical host named "**CC.Miskatonic.EDU**," but the appearance of such a host can be created, with the mail forwarded to whatever physical host seems appropriate at the moment. A user can make long term assumptions about what his electronic mail address will be, without having to worry that the particular machine on which he currently reads his mail will be decommisioned a week after he places an order for a three year supply of business cards. Further, the choice of physical host can be changed quickly and (to the outside world) invisibly; readers who have had the experience of decommisioning a network host supports a heavy electronic mail traffic volume will appreciate how useful this can be.

- It allows the specification of a series of machines that will accept electronic mail for a particular host when that host is down. On the INTERNET, where long-haul links are a scarce resource, much of the electronic mail traffic moves when the network load is light, that is, in the wee hours. If **Odin.CC.Miskatonic.EDU** happens to be down some morning, but **Thor.CC.Miskatonic.EDU** is up, **Thor.CC.Miskatonic.EDU** can accept mail bound for **Odin.CC.Miskatonic.EDU** and spool it until **Odin.CC.Miskatonic.EDU** comes up. This can also prove useful if **Odin.CC.Miskatonic.EDU** turns out to be down for longer than anyone anticipated.

Electronic mailers that use the mail agent RRs have started appearing on the INTERNET in the last year or so. The author is not aware of any mailer daemons that handle the last case, but this will almost certainly change.

The DNS includes support for *mailbox* RRs, although these are not currently in widespread use. A mailbox is conventionally written (on the INTERNET, anyway) in the form "**user@host**;" in order to usefully represent mailboxes in the DNS, they need to be encoded. This is trivial: an electronic mail address such as "**Cthulhu@Miskatonic.EDU**" is encoded as the domain name "**Cthulhu.Miskatonic.EDU**." The existing mailbox RRs support mail forwarding and mailing lists. The details of exactly how these RRs should be used have yet to be worked out.

### The User Interface

Up to this point we have ignored an important part of any implementation of the DNS, the user interface package. This is the set of library routines or operating system calls responsible for translating between the kinds of requests users and their programs might want to ask and the kinds of queries that a resolver can attempt to answer. The DNS protocol specification does not address this aspect of the system, viewing it as implementation dependent. This view, while convenient for the implementors, has proven somewhat unpopular among the users forced to use the minimalist user interfaces available with the first implementations of the DNS protocols. The new version of the DNS specification ([RFC-1034]) presents an outline of a minimal set of user interface functions:

- A function which translates a host name to an INTERNET address.
- A function which translates an INTERNET address to a host name.
- A function that returns arbitrary RRs.

More sophisticated user interfaces have begun to appear; see the section on the CHIVES system, below, for a brief description of such an interface.

### Reverse translations

The alert reader may be puzzled as to exactly how the address-to-name function, mentioned in the previous, is provided. One unfortunate property of a distributed tree-structured database such as the DNS is that, while it is relatively straightforward to locate a RR given its name, locating a particular RR without its name is impractical. The difficulty is that the later operation requires a tree traversal, which is a hideously expensive operation in the unlikely chance that it is possible at all. The only situation where such an operation (called an *inverse query*) is practical is if a resolver has good reason to believe that a particular name server will have the information in its local database. This is not reliable enough to use on a large scale.

To work around this problem, the DNS uses the use of a secondary tree of address-to-name translations. An INTERNET address of the form "a.b.c.d" is encoded as "**d.c.b.a.IN-ADDR.ARPA**," that is, to find out the name of the host at INTERNET address 11.22.33.44, the user interface asks the resolver to look an RR with a special TYPE and a name of "**44.33.22.11.IN-ADDR.ARPA**." The RDATA portion of the RR returned by such a query consists of the host name associated with the address. A similar scheme is used to encode information about the network addresses of of the Internet Protocol gateways connecting the various networks that make up the INTERNET.[16] The ordering of the components ("octets") of the

[16] See [RFC-1035, pp. 22–23] for details.

address is reversed because INTERNET network numbers are big-endian (the most significant bits in the address are in the leftmost octet) while the DNS naming structure is little-endian, as mentioned previously.

Several points about this scheme are worth noting:

- It does not do anything about the inverse translation problem in general; it simply provides a work-around for INTERNET addresses, which were felt to be the most critical case.

- It depends on the hierarchical nature of INTERNET addresses.

- It duplicates data from the "normal" part of the naming tree. Whenever such a duplication exists, the opportunity exists for administrators to create an inconsistent state by incorrectly duplicating the data. For this reason, use of the inverse naming tree should be restricted to those applications where it really is the only possible solution. Unfortunately, much existing software makes the implicit assumption that name-to-address and address-to-name will always be each other's inverse functions, and that if one function is available the other one will also be available.

## Lessons learned implementing the CHIVES system

The author of this paper recently released a beta-test version of an implementation of the DNS protocols, called CHIVES. While a lengthy discussion of this effort is inappropriate for the present forum, some of the lessons learned from this project may be of interest, and are presented here.

### Distribution of data in the DNS naming tree

In the several years since the DNS first became an operational system, some patterns have emerged in the way administrators name RRs. One interesting feature is the branching factor at each node of the tree, something that an implementor would like to know when choosing the internal data representation for a resolver or name server. The distribution tends to be strongly bi-modal: either a node has at most a handful of children or it has a lot of children, in the 50–100 range or even greater. This poses an interesting problem, since there is no way to know in advance how much the branching factor will vary from one node to the next in a dynamic representation of part of the DNS tree; the particularly hard case is representing the resolver's cache efficiently. Hash tables that work well for small branching factors will not work well for large ones, and vice versa. Re-hashing when the branching factor exceeds a certain limit would work, but the cost of the re-hashing operation will probably be high.

The CHIVES system uses a height-balanced binary tree (specifically, an AVL tree) for its internal representation. This works well for insertions and lookups, since the worst case time for these operations is $O(\log n)$. This representation does not, however, work well for deletions, because maintaining the AVL property during deletions is expensive; in practice this is not a serious limitation, because some kind of garbage collector is needed to recover from memory fragmentation, and, depending on the design of the garbage collector, the need for balanced deletions may vanish entirely.[17]

### Timesharing, Locking, and the "Shared Database"

As mentioned above, the conceptual distinction between resolver and name server is not present in all implementations. An implementor might want to combine the two programs (and the name server maintenance functions) into a single program, or leave the programs separate but have them use a database in shared memory. Both of these approaches have disadvantages. A single program is subject to availability problems: if it is engaged in a long transaction, such as transferring an entire zone, it will probably be unable to service user requests until the transfer is complete. The shared memory approach requires that the various programs be able to lock all or part of the shared database to maintain consistency; this is easy to say but hard to do correctly, and an error can be disastrous, particularly if the implementation directly involves the operating system via special system calls that depend on correct operation of the locking mechanism.

CHIVES uses the multiple process approach, but does not attempt to share memory between the processes. The resolver, zone transfer client, UDP name server, and TCP name server are all separate processes. Zones are represented as simple text files, using a slightly extended syntax to allow, for example, the zone transfer client to record when it obtains a zone so that other programs (including a later instance of the zone transfer client) will know when the zone becomes obsolete. In the few cases where direct communication between processes is required, CHIVES uses a simple lock-step protocol that can be safely aborted from either side at any time. Thus, both locking and scheduling tasks are accomplished by built-in mechanisms in the operating system: the filesystem code and the timesharing scheduler, respectively.

As additional benefits, CHIVES is more portable than a system using the shared memory scheme, and is (potentially) more robust than either of the other schemes could make it: failure or corruption of one component of CHIVES does not necessarily imply failure or corruption of any other component.

Obviously, there is a price for all this, which is that restarting any of the processes is fairly expensive, since it may have to load a number of files into its internal database. This can be partially alleviated by various techniques, such as compiling the text files into a relocatable

[17] In fairness to previous implementors who have chosen simpler data representations, the author freely admits that, to this day, attempting to visualize the entirety of the in-memory cache representation makes the author's head spin. Drawing a picture of it requires at least four dimensions....

binary format that can be loaded quickly, or by making a "dumped" version of each of the process with the appropriate data already loaded into the process's image file. In any case, the price seems low compared to the benefits of this approach.

### Bootstrap information

Previously, we mentioned that, in order to know to find the root of the global name tree, a resolver needs to have some configuration information supplied to it when it boots. This can take one of two forms:

- A file of RRs to load into the resolver's cache.
- A list of default name servers to use when nothing else seems appropriate.

While cache preloading is occasionally useful, it is a mistake to use it for this purpose, for several reasons. First, the bootstrap information is in the form of RRs, which have to have TTL fields, and, since the resolver would be crippled if those RRs ever time out the TTLs must be set to some unreasonably large value. Second, if the resolver shares memory with or is the same process as a name server, this information is now available to the name server to distribute: this is not a good thing, since the local configuration file may be badly out of date, causing the name server to distribute obsolete information for zones for which it has no trace of authority. Exterminating erroneous RRs (sometimes called "bogons") from the database is very difficult; the only way that bogons ever disappear is when caches are flushed or when the TTLs expire; due to an unfortunate bit of history, the "unreasonably large value" most often used is 99999999 seconds, which works out to something over 3 years.

CHIVES, as might be guessed from the preceding paragraph, uses the second method for locating the root servers. This information can never time out, and is not part of the database proper, so it can never be distributed. If this configuration information is incorrect, it will, at worst, cripple the resolver that uses it to boot; it will not corrupt the rest of the database. The default servers need not be root servers, merely servers that this particular resolver trusts to either answer correctly or refrain from answering. Furthermore, since these default servers are used only as a last resort, the resolver can tell that it is using bootstrap information and take steps to remedy the situation, rather than continuing to believe possibly obsolete RRs.

There is, at this point, no excuse for writing new DNS programs that depend on preloading the cache: any existing programs that do depend on this method should be fixed.

### Timeouts

One difficultly in designing a resolver is selecting an appropriate timeout interval. Not only are there drawbacks (described previously) to timeouts that are too long or too short, but the problems involved may overlap in the middle, that is, some timeout intervals may simultaneously be both too long and too short. CHIVES sidesteps this problem by using two different timeouts. User queries return to the user with a timeout error relatively quickly, but the resolver will continue to attempt to find an answer for a short while after this happens. The assumption is that the user may well try again in a little while; this is particularly true if the "user" is the local host's electronic mail daemon. If the query is answered during the period between the two timeouts, the answer will go into the resolver's cache, and should be immediately accessible when the user tries again.

### Suspicion

Another issue in resolver design is suspicion. A resolver can not assume that everything it gets in an answer is true; bogons can and do show up. The CHIVES resolver makes several checks before caching any RRs it receives:

- Does this message look like it really pertains to a currently outstanding query? The format of an answer message contains an echo of the question, as well as an echo of some arbitrary identification bits. This step flushes totally spurious messages.
- Does the RR pertain to the root? If so, it is ignored unless the question was about the root. Name servers will usually supply addresses of servers when answering a query with a referral; this is often helpful, but experience has shown that it is not safe to blindly accept new addresses for the root name servers. The CHIVES resolver only accepts new addresses for the root from its trusted default servers, or servers to which the trusted default serves referred it.
- Does the TTL attribute of the RR look "reasonable?" The resolver is skeptical about RRs with TTLs that are too large, on the theory that such RRs are probably escaped bogons from somebody's cache preload file. The resolver will either reduce the TTL to some more reasonable value, or discard (refuse to cache) the RR, depending on a flag in the configuration file. The value of "too large" can, of course, also be changed.

### User interface

The CHIVES system has a more sophisticated user interface than the minimal approach described in [RFC-1034]. Some of the functions are general purpose, some are geared specifically towards supporting an intelligent mailer daemon. Most of these functions return the "canonical name" of the target object: that is, they tell the user at what name the target RRs were found, after following any nickname pointers. The functions are:

- *Primary name and address:* Returns the address of a host in a particular protocol suite (that is, for a particular value of the CLASS attribute). Returns canonical name of the target host.

- *Mail Agent:* Returns a vector of host names, sorted in descending order by the preference portion of the RDATA, for a mail agent lookup in a particular CLASS. Returns canonical name of the target host.
- *Validate Name:* Checks for the existence of RRs that would demonstrate the existence of some object not directly represented in the DNS. For example, a maildrop exists if either a host address or a mail agent RR exists. Some applications, such as an electronic mail composition program, don't care about how the maildrop is represented; they just want to know if a particular name is valid. Returns canonical name of the target host.
- *Authenticate Address:* As previously mentioned, using the inverse mapping tree (**IN-ADDR.ARPA**) may cause consistency problems. Furthermore, not all transport protocols can be represented in this way. Some of the programs that ask for an address-to-name translation are really trying to answer the question "is address X really the address of host Y," for example, an electronic mail listener process that wishes to determine whether the entity on the other end of the network connection is lying about the name of its host due to malice or confusion. In such cases, the program requesting the address-to-name translation already has a host name on hand, and the question can be rephrased as "does host Y have an address X." This completely avoids using the inverse naming tree, and answers the question equally well from the user's standpoint.

### Search paths

In the entire preceding discussion, we have made the implicit assumption that the user has been specifying complete domain names. In practice, a user on **Odin.CC.Miskatonic.EDU** does not want to have to type the complete name "**Loki.CC.Miskatonic.EDU**;" since both are within **CC.Miskatonic.EDU**, the user wants to type "**Loki**" and expects some piece of software to do the "obvious" thing. This is another case of something easy to say and hard to do. What suffixes are the "obvious" ones to try? How long is the user willing to wait for an answer? (The answers are, of course, that all the hosts this particular user wants to talk to are obvious, the other 99% of the hosts in existance don't matter, and the user is willing to wait perhaps a few seconds.)

We define a *search path* to be a list of suffixes to append to a name when attempting to resolve it. The CHIVES resolver supplies two kinds of search paths: the *local* search path and the *remote* search path. The local search path is only used when looking in locally-loaded zones; the remote search path also searches the cache and causes actual resolving via the network. Thus a site can use search paths for names in zones they talk to often, while limiting the amount of effort spent attempting to resolve possibly non-existent remote names. A name that is terminated with a trailing dot, such as "**Loki.CC.Miskatonic.EDU.**," is looked up exactly as it stands except for removal of the trailing dot, without any use of either of the search paths; this is appropriate for any application that for any reason knows that it already has a completely specified name.

### For more information...

The DNS is a relatively young protocol, as such things go. Both the existing implementations and the concepts underlying them are still undergoing active development. There is an electronic mail forum for developers and interested bystanders, the INTERNET mailing list **Namedroppers@SRI-NIC.ARPA**. If you are on the INTERNET or have electronic mail access to it, and wish to be added to the list, send a message to **Namedroppers-Request@SRI-NIC.ARPA**.[18] All of the documents with the notation "RFC" (short for "*Request For Comments*") are available online in the **NETINFO:** directory on **SRI-NIC.ARPA**. If you are not on the INTERNET, or you prefer your documentation in hardcopy form you can obtain any of the documents listed in the bibliography of this article except for [Dyer 87] by contacting:

The DDN Network Information Center
SRI International
333 Ravenswood Avenue, Room EJ291
Menlo Park, CA 94025

CHIVES is a free software package, currently implemented on the TOPS-20 operating system but very likely to be ported to other systems in the near future. Volunteers interested in working on the project are welcome. For more information on the CHIVES system, send electronic mail to **Bug-CHIVES@XX.LCS.MIT.EDU**, or write to:

S. Robert Austein
Massachusetts Institute of Technology
Laboratory for Computer Science, NE43-503
545 Technology Square
Cambridge, MA 02139

### References


[DDN 85] *DDN Protocol Handbook*, Menlo Park, CA, DDN Network Information Center, December 1985.

[Dyer 87] Dyer, S., and F. Hsu, "Hesiod", *Project Athena Technical Plan—Name Service*, April 1987, version 1.9.

[RFC-883] Mockapetris, P., "Domain Names—Implementations and Specifications," RFC-883, University of Southern California Information Sciences Institute, November 1983.


[18] Do *not* send a subscription request directly to **Namedroppers**; doing so will get you a free lesson in network etiquette from a couple hundred strangers, in varying degrees of politeness.

[RFC-920] Postel, J., and Reynolds, J., "Domain Requirements," RFC-920, University of Southern California Information Sciences Institute, October 1984.

[RFC-973] Mockapetris, P., "Domain System Changes and Observations," RFC-973, University of Southern California Information Sciences Institute, January 1986.

[RFC-974] Partridge, C., "Mail Routing and the Domain System," RFC-974, CSNET CIC BBN Labs, January 1986.

[RFC-1032] Stahl, M., "Domain Administrators Guide," RFC-1032, SRI International, November 1987.

[RFC-1033] Lottor, M., "Domain Administrators Operations," RFC-1033, SRI International, November 1987.

[RFC-1034] Mockapetris, P., "Domain Names—Concepts and Facilities," RFC-1034, University of Southern California Information Sciences Institute, November 1987.

[RFC-1035] Mockapetris, P., "Domain Names—Implementations and Specifications," RFC-1035, University of Southern California Information Sciences Institute, November 1987.

# NETWORKS SIG

# An Architectural Perspective of a Common Distributed Heterogeneous Message Bus

**Howard Kilman** and **Glen Macko**
DIGITAL EQUIPMENT CORPORATION
MERIDEN, CONNECTICUT

## Abstract

Heterogeneous networked computing over the last few years has gained popularity in the press and has started to emerge as product. This paper describes a common heterogeneous message bus abstraction that functions as a bridge between execution units on a single processor or on distributed processors. The message bus is used for integrated peer-to-peer communications. The bus is integrated because it blends **all** inputs into a single message queue. Applications and value-added network services can be built using this peer-to-peer platform. In addition, this message bus provides: a common application interface, simple and consistent operations, an efficient implementation, a host of support utilities, and a flexible choice of underlying networks (with the ability to easily integrate user specific networking schemes). The bus is heterogeneous because it has been implemented on VAX/VMS, RT, VAXeln, MS-DOS, RSX, and ULTRIX.

The conclusion is that a Message Bus messaging sub-system is more effective and efficient than traditional peer-to-peer communications systems.

## INTRODUCTION

This paper discusses the concept of a Message Bus. The basic rationale for a Message Bus is outlined with generic requirements for distributed applications. An implementation of a Message Bus, called PAMS, is detailed. The basic PAMS functionality is outlined. Additional services such as selective broadcasting, non-volatile recoverable queues, and journaling are discussed. Comparisons are made to other networking approaches such as custom developed networking systems and Remote Procedure Calls. Finally, the paper concludes with some thoughts about the cost effectiveness of a Message Bus architecture as the peer-to-peer networking platform for distributed application developments.

## RATIONALE

Applications over the last 35 years of automation have become increasingly complex and distributed. From the centralized batch oriented applications of the 1950s and 60s, today's applications tend to be distributed in both control and data.

The basic reasons for this trend for distribution lies with increased demand for expanded application functionality and performance. In addition, the application hosts have become increasingly smaller and cheaper, while maintaining relative performance. Finally, the emergence of interconnecting networking technologies has solidified this trend towards applications distribution.

As systems have become more distributed, the complexity of the solutions have risen proportionally. These distributed implementations compared to their centralized counterparts are more complex in: software architecture, hardware architecture, and networking architecture. The software architecture is more challenging due to the basic asynchronous nature of distributed processing and the need to coordinate the access of common resources. The software designed is typically multi-threaded[1]. Multi-threaded designed applications are difficult to implement and maintain. The hardware of these distributed solutions is often based on multiple heterogeneous platforms. Users of systems in such networks are often frustrated by the fact that they can't get those systems to work cooperatively. In addition, each hardware platform invariably supports unique development tools, rules, and methods. Finally, the networking subsystem that provides the means of distribution tends to be hybrid [2].

[1] Multi-threaded in the sense that each module accepts inputs from multiple sources and has distinct processing control loops

[2] Hybrid in the sense that it uses multiple access methods and communication protocols

The traditional software development methodologies and tools are generally ineffective or inefficient when used with distributed application systems. Applications developers require the automation of software production. Buzz words like: re-usability, portability, network monitoring, defect removal, application prototyping, software design simplification have emerged. Application developers are demanding comprehensive off-the-shelf networking tool kits. They require networking platforms that have: a standard interface for accessing the network, a standard set of networking features, support for multiple host environments, support for multiple networking backbones, simplification of coding by supporting single-threaded designs, existence of built-in productivity tools for simulation and test, and is "future safe" in that it can accommodate new emerging environments and networks.

A standard interface for accessing the network implies that the procedural or message based interface to the network should have well defined and consistent syntax and semantics. These rules should apply equally, independent of the hardware or system software backbones.

In choosing the physical network to use in a application design many networking characteristics are mapped into application requirements. Features such as circuit based or logical link delivery (one-way), broadcast or multi-cast delivery (N-way), or connectionless delivery (any-way) must be selected in addition to transport speed, processing overhead, message length restrictions, and distance limitations. The characteristics of the network sometimes determine the methodology in which the software must be developed. Application developers should produce code that is independent of the characteristics of the underlying network. They should use an interface that is common across many different networking schemes.

Single-threaded designs are possible if all inputs and outputs to a process are concentrated into a single work queue. Processes then have a single control loop in which each item of work (e.g. message) is processed to completion, required output or actions are generated, and the next work item is then extracted. Hence, this is an example of one control loop or single-threaded. The items can be placed on a process's input work queue in a FIFO or in a priority based manner. Items may be extracted from the queue either sequentially or by some selection criteria (i.e. sender, message class, etc). Single-threaded designs are possible if **ALL** events that a process requires are integrated into this work queue. This implies all messages from all partners, all timer events, all events from external devices, and all general networking events. Single-threaded design ideally fit the Client/Server model of network application design.

The Client/Server model defines some processes as "servers". These are continuously running programs which wait for client requests (packaged as messages) to be placed in their input queue, processes them as they arrive, and send replies when required. This allows the user to develop applications with tremendous implicit parallelism. Distributed application design using this model has proven to be extremely cost effective and popular.

As distributed networking systems are developed, the need for enhanced productivity tools has become apparent. Typical gaps in tool sets are:

- message simulators
- message capturing
- a message replay facility
- a general scripting capability

Scripting should allow the encoding and the ability to control the flow of simulated message traffic. The effective control of an operational network requires yet more tools. An example of such a tool would be a centralized network monitoring facility.

The technologies in which distributed applications are based is constantly changing. New hardware platforms, software languages, networking backbones are consistently being introduced. The ability to integrate these new pieces into an existing distributed network umbrella is highly desirable.

The next section defines a generic **Message Bus** and a DIGITAL implementation of a Message Bus called **PAMS**. The paper will show how the generic characteristics of a Message Bus and PAMS assist in the design, development, test/integration, and control of distributed applications.

## DEFINITIONS

### Message Bus

A **Message Bus** (see Figure 1) is a data highway in which:

- all network events and work units (data) are packaged into messages,
- messages can be of variable size and can be categorized by user definable classes and types[3],
- processes have a single attachment point to the bus where all communication (i.e. messages) to other processes is funneled,
- the highway uses a simple logical bus topology[4] (as compared to a star, point-to-point, hierarchical, etc) that spans both inter-cpu and intra-cpu,
- the implementation is host and network backbone independent, and
- processes attached to the message bus can communicate to any other PAMS attached process, without a formal connection sequence required for each partner.

[3]The messages preserve the "write" (i.e. record) boundaries of the sending process.

[4]A bus topology is inherently simpler to attach and control. This makes peer-to-peer communication simple and efficient.

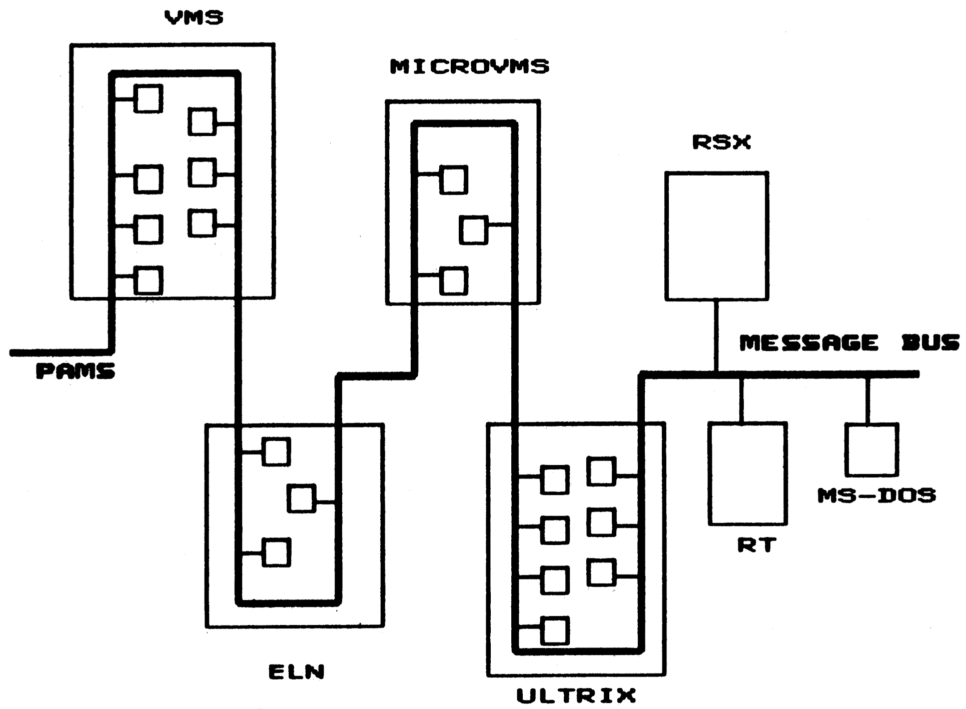



**Figure 1:** PAMS Message Bus

## PAMS

PAMS is a heterogeneous implementation of the Message Bus architecture. **PAMS** is a ***heterogeneous***, ***integrating***, ***extensible***, ***N-way***, ***full-duplex***, communication message facility that is ***network independent***.

PAMS is *heterogeneous* in that it is supported on the following environments: VAX/VMS, VAX/ULTRIX, VAXeln, RSX-11, RT-11, and MS-DOS. The VAXeln and VAX/VMS implementations are full functioned while the others are end-node implementations.

PAMS is *integrating* because it blends all the classic input/output that a networking task requires. As shown in Figure 2, the user application has a single point of focus for all external work units (input or output). All units are formed into messages and are presented on this single queue in either a FIFO or a priority based fashion. PAMS also supports the capability to selectively receive messages based on the source network address of the message. Messages can be received from any other PAMS attached process, on any PAMS connected network, and can include simulated messages from PAMS SCRIPTS[5], and user specified timer messages.

PAMS is *extensible* because it is packaged with a defined external interface. This interface allows the transparent integration of user specific networks and/or devices. The input/output to these special networks and devices occur simultaneously with normal PAMS messaging.

PAMS supports an *N-way* communication mechanism. This implies that any process can communicate to any other attached process without a formally established connection between them. This is sometimes referred to as a "connectionless" environment. In addition, PAMS has the capability of performing one-way (circuit based), and any-way (broadcast/multi-cast based) messaging.

PAMS is *full duplex* in that messages can be simultaneously sent and received by all PAMS attached processes.

PAMS is *network independent* because its transparently supports (with the identical interface): DECnet, direct Ethernet, direct DDCMP, and LU6.2 communications. All of the semantics and idiosyncrasies of each network are masked from the user's application. PAMS provides a set of standard interface procedure calls. These calls are valid over all PAMS supported network backbones and all supported PAMS host environments.

The common PAMS interface is invoked procedurally, but the basic control of the network is message based. Message based interfaces tend to be more extensible, migratable, and portable than procedure call based interfaces. Classical procedural interfaces are totally synchronous. The PAMS interface supports both synchronous and asynchronous operations. However, the default is asynchronous operations. All the synchronous calls have an optional timer value attached. This feature can eliminate the deadly "wait forever" condition.

[5]PAMSCRIPT is a tool that allows the symbolic simulation, capture and replay capability of PAMS' messages

Finally, the PAMS Message Bus is *stateless*. After a process is attached to the bus, messages can flow freely between any partners, regardless of any explicit or implied network/application state [6].

## ARCHITECTURE

PAMS fits within the ISO model for communication networks as shown in Figure 3. The PAMS interface routines are positioned in the application layer (layer 7). The characteristics and robustness of the various networking backbones that PAMS supports determine the specific ISO layer coverage. When using PAMS in its standard form (e.g. DECnet), there is support for all ISO layers through session (i.e. 1 through 5). When used in conjunction with direct Ethernet, PAMS supports only physical, data link, and a minimal transport.

Any good peer-to-peer system should be capable of setting up communications and transmitting and receiving messages with a minimum of code and hassles. As previously noted, the basic architecture of PAMS is a Message Bus. The explicit savings of a bus architecture is shown in Figure 4. In a bus based network, process A can send to any other process (e.g. B, C, or D) over a single attachment point (e.g. PAMS queue). In the classic point-to-point network scheme, process A must maintain separate links to each of its communication partners. Hence A has multiple control points and must internalize the prioritization and processing of input work units.

Figure 5 clearly shows how a Message Bus architecture simplifies the logical structure of a distributed application. In the physical layout, processes A through K are hosted on various hardware and possibly systems software platforms. Processes A1, A2, and A3 are on a single host. The network that interconnects these processes is a hybrid configuration (noted N1 through N5). There is some indirect connectivity between all potential network partners. However, the mechanics and rules for peer-to-peer communication between any two partners most likely will be different. In addition, each partner would most likely require a separate message port for each active connection (see Figure 4). Therefore, using traditional networking techniques, each network process, in the physical application layout, would require the intelligence to maintain and control multiple network access points and adjust to the different network interface specifications. This same application developed using PAMS uses a single standard network interface and controls all network data flow through a single attachment point.

## MESSAGE BUS EXTENSIONS

Since the first PAMS paper (??) there have been significant extensions to the PAMS Message Bus. These extensions fall into four categories: Heterogeneous System

[6]LU6.2 peer-to-peer connections are Statefull. Only defined set of operations are possible per application state

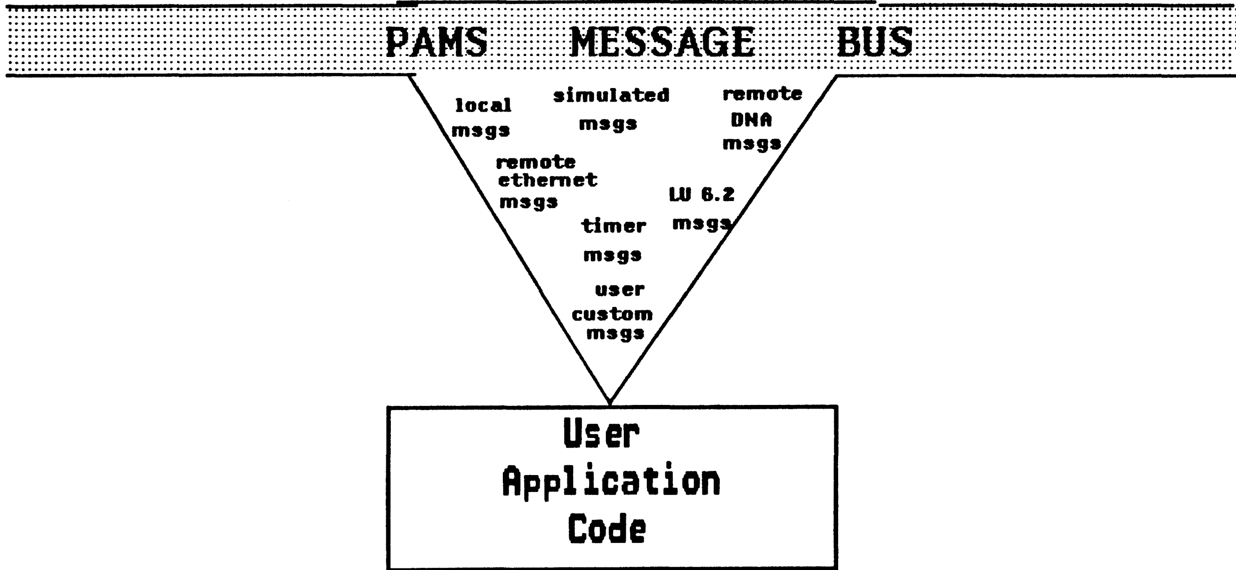



Figure 2: PAMS Integrated Message Bus

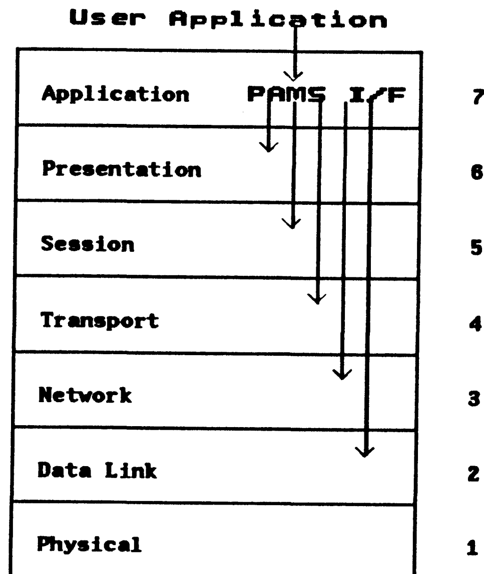



Figure 3: PAMS ISO Architecture

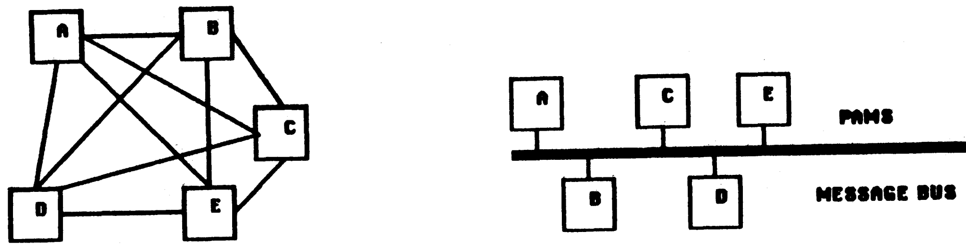
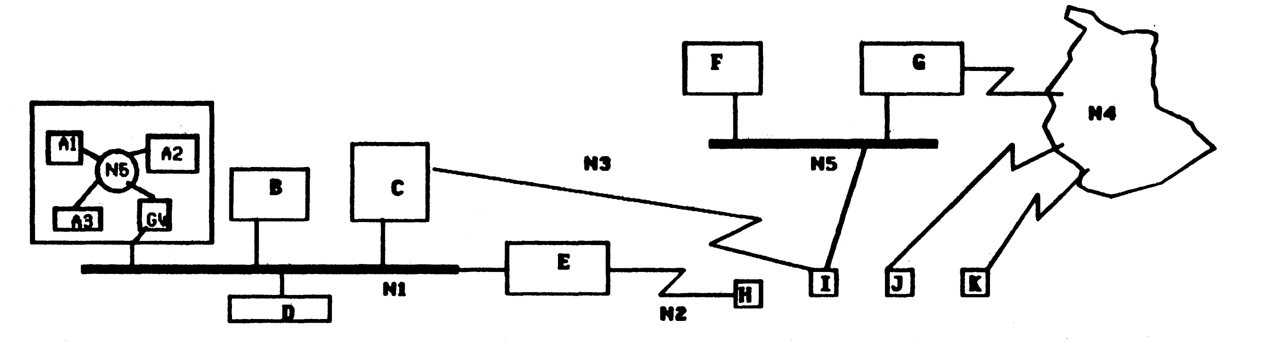



**Figure 4:** Point-to-point vs. Message Bus

Physical Layout of a Networked Application




Logical Layout of a Networked Application using PAMS

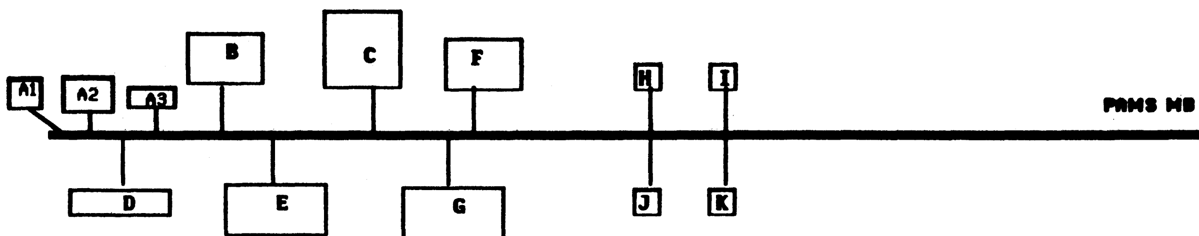



**Figure 5:** Networked Application

Support, Message Recovery Services, Selective Broadcast Services, and support for additional network backbones.

### Heterogeneous System Support

A typical distributed networked application spans multiple hosts and multiple host environments. PC to mini, PC to mainframe, mini to mainframe, are examples of typical implementations. The PAMS Message Bus has been implemented on a range of DIGITAL computational platforms.

**VAX-PAMS** is the original platform for the PAMS Message Bus. It supports the full PAMS feature set. Local messaging is done using VMS global sections with the added value of protected mode access. VAX-PAMS has a message routing capability for directing PAMS messages across PAMS Groups[7]. VAX-PAMS also supports the Selective Broadcast Services (SBS) and Message Recovery Services (MRS) which will be discussed later. VAX-PAMS supports the following networking backbones: DECnet, direct DDCMP, direct Ethernet, and LU6.2.

**ELN-PAMS** is the other full functioned implementation of PAMS. The basic communication tools of VAXeln typically involve local messaging being done with VAXeln datagrams and remote messaging being done with VAXeln DECnet circuits. PAMS extends these tools and presents a single network transparent interface to the user with all the Message Bus features. ELN-PAMS supports DECnet and direct DDCMP for its remote messaging and datagram Message Objects for its local messaging.

**PC-PAMS** is a MS-DOS, end-node implementation of the PAMS Message Bus. As an end-node, it requires a PAMS routing host. It supports automatic connection to a routing PAMS host and also automatic fail-over to a secondary router in case of line failure. Limited MRS and SBS functionality is supported along with a full message capture facility. PC-PAMS is built using DIGITAL'S DECnet-DOS product. DECnet-DOS supports Ethernet and asynchronous DDCMP connections. PC-PAMS supports these networks and all PCs, XTs, ATs, and VAXmates using PC(MS)-DOS 3.1 through 3.3. As in all PAMS implementations it supports the PAMS interface standard and a PAMS based applications can be ported readily to a PC.

**ULTRIX-PAMS** is currently another end-node implementation of PAMS. It resides with DECnet-ULTRIX. It operates over DECnet using Ethernet or point-to-point links. All the features of PC-PAMS are supported in ULTRIX-PAMS. A routing version of ULTRIX-PAMS is under investigation.

**RT-PAMS** is an end-node, direct Ethernet only version of PAMS. RT-PAMS is typically used as a bridge between an older existing RT application and a newer supervisory VAX application.

[7] A PAMS Group is a collection of processes that are utilizing the PAMS Message Bus on a single host. There can be multiple PAMS Groups per host. PAMS routing is required to send messages between PAMS Groups.

**RSX-PAMS** is another end-node implementation that allows a RSX task to have N-way connections to a VAX processor. RSX-PAMS supports direct Ethernet and DECnet backbone networks.

### Message Recovery Services

Maintaining consistency of data is a key element in building a data processing system which can withstand failures of its hardware and software components. VAX/VMS provides standard database products to maintain consistent data which resides in the file system, such as RMS, DBMS, and RDB journaling.

Message Recovery Services (MRS) for the PAMS Message Bus are intended to extend data recovery to the level of pending messages. Using Message Recovery, the sender is relieved of the responsibility of tracking the progress of a message through the next level of computing.

Message Recovery Services increase the robustness of PAMS Message Bus delivery by providing applications with the ability to recover from message delivery failures due to:

- application task abort, or
- communication line failure, or a
- system crash.

In addition, MRS provides a mechanism to allow a target process to inform the sender of the outcome of message processing. Applications no longer need to exchange reply messages for the purpose of communicating a simple "success/failure" status.

MRS implements these features by using:

- enhanced delivery services utilizing message queues recorded on disk,
- recovery services for undeliverable messages, and
- journal services for maintaining message traffic logs.

There is an extra cost associated with making messages recoverable via non-volatile disk storage. The amount of extra processing is enabled on a message basis. This allows applications to selectively incur the additional processing imposed by MRS for just those messages which are not easily recovered.

Figure 6 details an example of an application that fails and subsequently recovers while using MRS. Task A is sending messages to Task B. The communication link between the processes is broken. In this example it is due to a system failure, but it could be for any reason. For all messages subsequently sent during this failure mode, MRS can be instructed to take one of the following actions:

- discard the message, or
- return the message to task A as undeliverable, or

- archive the message in the source recovery queue for later delivery.

The action taken per item is determined by the sender of the message (i.e. Task A).

When the link between system A and B is re-established, the messages in the source recovery queue can either be automatically delivered to Task B or can requested by Task B manually. The method of recovery processing is determined by user definable MRS recovery characteristics. In addition, there also exists a destination recovery queue that is used for failure/recovery on the destination side of a transaction. These are for messages that arrived at the destination node, but were unable to be delivered to Task B.

### Selective Broadcast Services

Distributed applications frequently require the network to support the broadcasting or multi-casting of a single message to multiple recipients. PAMS supports Selective Broadcast Services (SBS). SBS is a facility that allows the selective distribution of a broadcast message. The distribution can be inter or intra cpu.

A process becomes selectable by SBS via a registration message. After registration, the processing of SBS is totally transparent to the application processes. The primary selection criteria for a broadcast message is the target address. PAMS pre-defines a range of addresses that are reserved for exclusive SBS use. These addresses are called Multi-point Outbound Targets (MOT) and they denote broadcasting events in the network. When an application process generates a message targeted to a MOT, it directs the SBS server to process the message as a selective broadcast event. All matching registrants are delivered a copy of this message[8].

There are two event types supported by SBS:

- **Private** - MOTs that fall within the Private range indicate that all SBS processing will occur on the SBS Server where the event was declared. Only this SBS Server's registration tables are scanned and processed for selection/distribution.
- **Universal** - PAMS supports a range of MOT addresses that have global scope. These MOT events, when declared, are distributed to all SBS Servers in the PAMS network. The actions performed is the union of all the SBS Servers' registration databases.

Figure 7 shows the difference between Private and Universal MOT events. Processes A1 through A15 reside on a VAX-PAMS host as a single PAMS Group Process B1 through B6 is a second PAMS Group on a ELN-PAMS system. If a Private MOT were declared in PAMS Group A only the registration tables of Group A's SBS Server would be processed. If, however, a Universal event were declared on Group A, then all groups that support SBS functionality would be notified of the declaration and all tables would be processed for candidates for distribution (i.e. Group A and B's SBS Servers).

SBS distribution locally requires the replication and transmittal of a "clone" message to all selected recipients. Inter-group distribution depends on the physical backbone network that PAMS is configured. On a DECnet network, Universal messages are replicated and distributed. But, on an Ethernet network, the SBS servers use the multi-casting capability of Ethernet to optimize the distribution phase. Therefore, only one distribution message would be sent on the network and only those PAMS groups with registered interest in the particular MOT event would enable the reception of this message. Hence, there is a savings in the CPU demand on both the sending and receiving partners.

### Networking

The default inter-node network used by PAMS is DECnet Phase IV. PAMS utilizes DECnet non-transparent task-to-task to extend the message bus architecture to remote CPUs. In addition, PAMS provides the capability to integrate other networks under the Message Bus umbrella. The following are some of the networks that have be added: direct DDCMP, direct Ethernet, and LU6.2.

The direct Ethernet allows PAMS access to the data-link layer of Ethernet. In addition, PAMS provides a scaled down transport layer that provides: circuit establishment, large packet disassembly/reassembly, flow control, error free transfer, and link loss detection. VAX-PAMS, RSX-PAMS, and RT-PAMS support direct Ethernet.

The PAMS LU6.2 connection provides a general-purpose interface to DIGITAL's DECnet/SNA APPC/LU6.2 application programming interface. This allows the user to develop "LU6.2 Servers" that meet the needs of specific applications, while enjoying all the productivity benefits that PAMS provides. VAX-PAMS is the only PAMS product that currently directly supports the LU6.2/PAMS network integration. However, all PAMS implementations can share access to LU6.2 resources by requesting services through a VAX-PAMS LU6.2 Server.

Figure 8 shows a hybrid network that includes Ethernet, direct DDCMP, and LU 6.2 connectivity. A PAMS based process can access any other PAMS based process regardless of the physical network connecting them. For example, the PC with PC-PAMS connected to the VAX over a dial-up DDCMP circuit has transparent and equal access to any process on VAX-PAMS or to a PC-PAMS computer connected on the Ethernet. Also, any PAMS process can use the PAMS/LU6.2 facility to transfer files to or from an IBM based CICS application.

## COMPETITIVE APPROACHES

Two common approaches used to resolve an application's requirement for peer-to-peer messaging are:

[8] The details of SBS processing can be found in reference ??

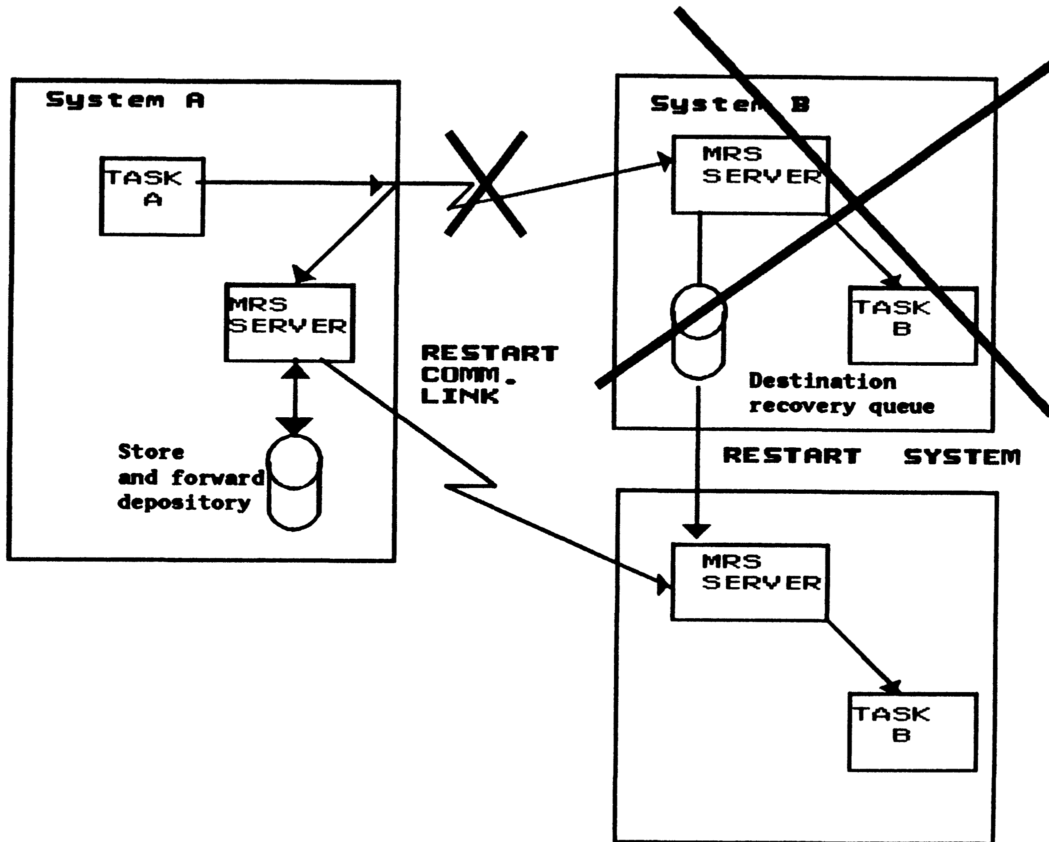



Figure 6: Message Recovery Services

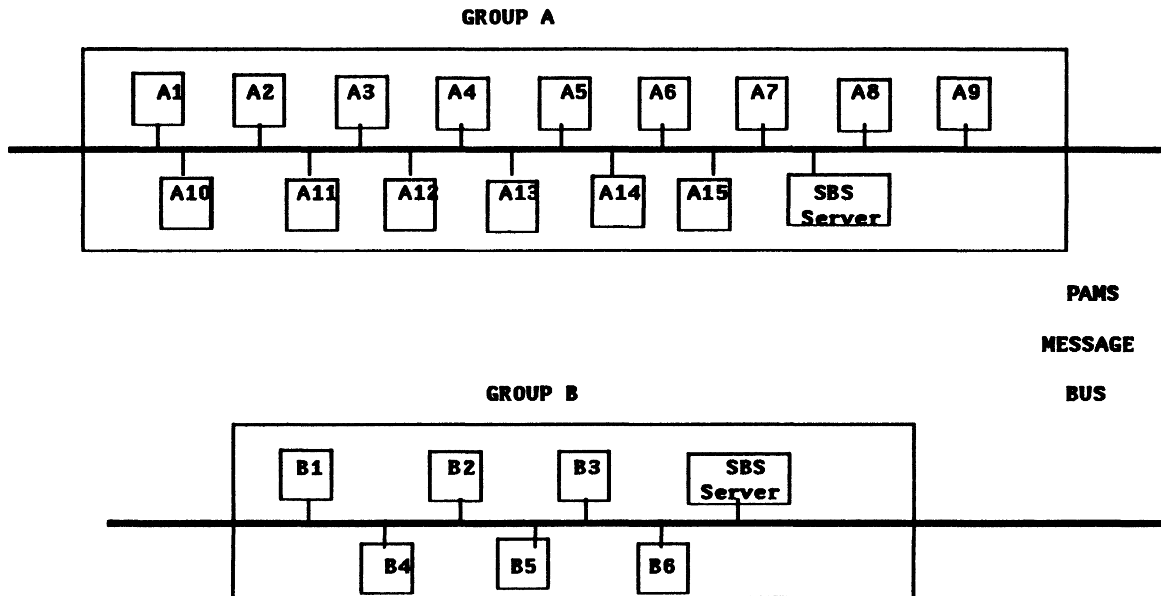



Figure 7: Selective Broadcast Services

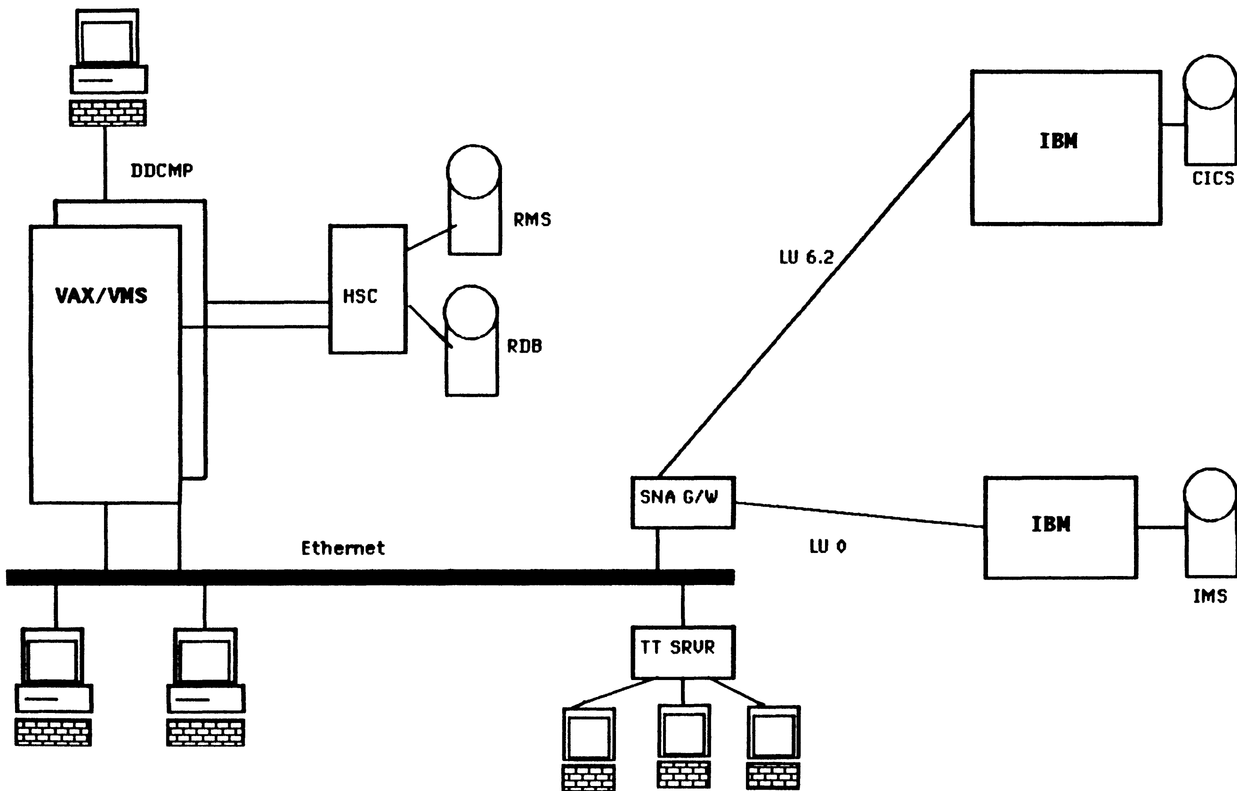



**Figure 8:** PAMS Networked Application

1. To build a callable layer for augmenting the basic networking facilities schemes available on the target host(s), or
2. To purchase an off-the-shelf message facility that meets the application's core requirements.

The PAMS Message Bus is a combination of 1 and 2. It is an off-the-shelf facility that is layered on standard DIGITAL supplied networking backbones (i.e. DECnet, Ethernet, SNA Gateway, Global Sections, Message Objects, etc). PAMS provides the added value of a Message Bus construct.

Figure 9 compares the features of the PAMS Message Bus against a set of off-the-shelf networking packages.

### Custom Implementations

Software designers using VAX/VMS, ULTRIX, VAXeln, or MS-DOS typically read the supplied documentation and develop a set of callable procedures to implement the peer-to-peer specific messaging facility they require. These facilities isolate the application writers from the details of the physical network. For example on VAX/VMS, applications would use Mailboxes for local messaging and DECnet Task-to-Task for remote messaging. DECnet and Mailboxes have different feature sets and interfaces. DECnet is available on all DIGITAL supplied operating systems, while the Mailbox facility is specific to VAX/VMS. Therefore, messaging on environments other than VAX/VMS would require yet another communication facility.

The DECnet Task-to-Task facility provides logical link (virtual circuit) capability between any two cooperating DECnet processes. The connections are point-to-point and messages are delivered to a unique communication port per partner. Therefore, if multiple partners are required then the application must establish individual sessions (see Figure 2) with each partner and either: poll the set of input queues, or write the applications so that it is multi-threaded[9]. In addition, networking events (i.e. link lost, partner disconnect, etc) are delivered to yet a different collection point.

Although DECnet is supported on all DIGITAL operating environments, the interface to non-transparent Task-to-Task facility differs on each implementation. Therefore, both the portability and development efficiency suffers. Traditionally, it requires a seasoned system developer to design and implement non-transparent Task-to-Task based applications. Also, DECnet does not provide the capability to integrate external events (i.e. timer events, device events, etc) into a single blended input queue. Finally, DECnet does not have any facility to perform message simulation and capture or message tracing.

Custom implementations for peer-to-peer communications have the advantage of the users developing the specific package that meets their requirements. However, this is at the expense of the time/effort/cost of designing, building, and supporting a custom package. Finally, it is unlikely that a custom implementation would have all the features of a mature off-the-shelf product.

[9] Multi-threaded in VAX/VMS context means using ASTs, in ULTRIX it means signals

### Remote Procedure Call

In Fall 1987, DIGITAL announced a program for a Remote Procedure Call (RPC) Facility. The details of this program are not public at this writing, therefore we will compare the PAMS Message Bus against the current most popular RPC facility.

SUN Microsystems has designed, specified, and introduced an open standard set of protocols that collectively are called Remote Procedure Call. RPC is a value added networking facility that is primarily based on the Client/Server model. The transactions between the clients and the server are strictly synchronous. As seen in Figure 10, the client process is blocked until the server process has received the message, processed the request, formulated the response, and sent it back to the client. This tight synchrony can be achieved in PAMS using Message Recovery Services. But, sender processes employed in PAMS usually continues execution after the message is either accepted by the local PAMS engine or queued to the target's message input queue.

In RPC's Client/Server model, the client is the master of the connection. The roles of the client and server can be altered using a RPC feature called "Callback". PAMS processes do not have defined roles and any process can send a message to any other PAMS attached process. As previously noted, PAMS is well suited to the Client/Server based applications.

RPC defines three interface layers for applications. PAMS provides a single interface definition. RPC also uses a protocol called XDR which handles the differences of data encoding in various RPC environments. XDR can also handle user definable arbitrary data structure definitions. PAMS has no such capability.

RPC makes no attempt to provide a blended input queue, message simulation, selective reception, message capture, or priority (e.g. out-of-band) messaging.

## FUTURE DIRECTIONS

As PAMS evolves, customers request new host and networking platforms, and suggest new built-in features to the existing product set. Some of these potential extensions include: OS/2 PAMS, full functioned ULTRIX-PAMS, OSI network, TCP/IP, and Hyperchannel support, and the use of DECnet Name Server for PAMS address resolution.

## CONCLUSIONS

In this paper we have outlined some general considerations that an application developer must address when contemplating a distributed application. A distributed application is any system that requires multiple processes

| | PAMS Message Bus | Custom Implement- | VAX/VMS Mailbox | DECnet Task-to- | Ethernet | MAP | Remote Procedure |
|---|---|---|---|---|---|---|---|
| Available on VMS | X | ? | X | X | X | X | Future |
| Available on non-VMS O/S | X | ? | | X | X | X | SUN |
| Tools for Custom Networks | X | ? | | | | | |
| Full Duplex Connections | X | ? | X | X | X | X | |
| Support Local Messaging | X | ? | X | X | | X | X |
| Portable CALL interface | X | ? | | | | | X |
| Network Transparent | X | ? | | X | | X | X |
| Session-less connections | X | ? | | | | | X |
| RCV from single queue | X | ? | X | | | | |
| Message Recovery | X | ? | | | | | |
| Broadcast/Mulicast Messages | X | ? | | | X | | X |
| High Priority Queueing | X | ? | | | | | |
| Selective Reception | X | ? | | | | | |
| Multiple Readers of a Queue | X | ? | | | | | |
| Message Capture | X | ? | | | | | |
| Message Simulation | X | ? | | | | | |
| Message Replay | X | ? | | | | | |
| Monitor Pending Messages | X | ? | | | | | |
| Integrated Timer Events | X | ? | | | | | |
| Connectivity to LU6.2 | X | ? | | | | | |
| Data Encoding/Decoding | | | | | | X | X |

**Figure 9:** Messaging Features Comparison

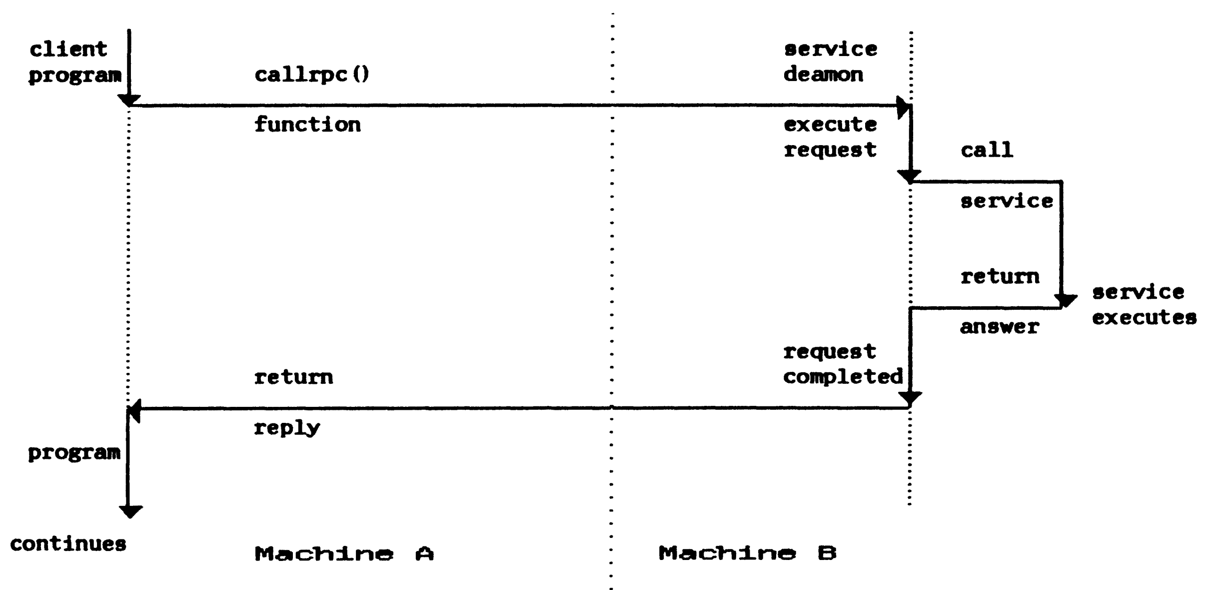



**Figure 10:** SUN RPC Transaction

to execute application transactions. These application's processes require some base inter-process (peer-to-peer) communication in order to exchange control and/or data. Also, these processes can reside on a single or multiple computational engines. A simple example of a distributed application would be two processes residing on a single cpu. One process could be providing an application user interface while the second process could be servicing an application database. A complicated example would be the applications detailed in Figures 5 and 8.

We have defined an architecture called a Message Bus and have mapped its features against these general distributed application requirements. It was shown how a Message Bus simplifies the design and implementation of peer-to-peer applications. Also, it was noted that the PAMS implementation of a Message Bus provides tools that enhance test and integration of such systems. The Message Bus architecture was then compared to distributed application developments using traditional DECnet and to the newer Remote Procedure Call facilities.

In conclusion, effective and efficient integration of heterogeneous peer-to-peer communications systems is indeed feasible and practical using the approach of a common message bus architecture spread over the diverse computing environments. Using such a sub-system is cost effective in the design and development due to intrinsically simpler single-threaded structure of the application software. The test, integration, and support phases of software development are vastly enhanced due to the existence of applicable PAMS tools.

The productivity savings of using a Message Bus exist for the spectrum of distributed applications. The savings are, however, greater in larger and more complex applications. But, all distributed applications benefit from the simpler, integrated, consistent, and testable networking characteristics of a Message Bus. Finally, applications developed using the Message Bus specification can be ported to other platforms and can integrate new networks as requirements dictate. The re-usability, simplicity, and adaptability of a Message Bus based application derives great savings in an application's net cost and will extend the system's expected life.

## ACKNOWLEDGMENTS

There are many individuals who contributed to the ideas, analysis, specification, and implementation of PAMS. The authors would like to take the time to explicitly thank: Martin Michelsen, Randy Skelding, Mick Konrad, Steve Judd, and Walter Stutzman for their technical and inspirational contributions.

## References

[1] H. Kilman, E. Berelian, L. Farmer, F. Schoen, P. Wang, J. Vij, "VMS, Xenix, Unix, and MS-DOS Transparent Resource Sharing", Proceedings from 1987 Spring DECUS

[2] G. Macko, "Developing a Message Bus for Integrating VMS High Speed Task to Task Communications", Proceedings from 1986 Fall DECUS

[3] H. Kilman, "A Fast Inter-Process Communication Facility for VMS", Proceedings from 1984 Fall DECUS

[4] R.S. Divakaruni and H. Kilman, "Futuristic Trends in an Integrated Office Environment", May 11, 1985 proceedings from VENCOM '85

[5] A. S. Tanenbaum and Robbert Van Renesse, "Distributed Operating Systems", Computing Surveys, vol. 17, no. 4, December 1985.

[6] "Selective Broadcast Services Users Guide", Digital Equipment Corporation

[7] "Remote Procedure Call Protocol Specification", Sun Microsystem, Inc.

[8] "External Data Representation Protocol Specification", Sun Microsystem, Inc.

# A Local Area Network for a Multivendor Environment

**Roger G. Ruckert**
Software Analyst
Medtronic, Inc.
7000 Central Ave., Mail Stop B100
Minneapolis, MN 55432

**Abstract**

In an effort to provide easier access to our different computer systems, reduce terminal and printer costs associated with these different systems, and provide for future growth, Medtronic has implemented a local area network. This paper will detail our hardware selections and describe what software interfaces were developed. Specific topics include broadband cable issues, protocol conversion, programmable terminals, and shared printer driver software.

## Company Background

Medtronic, Inc. is the world's largest manufacturer of biomedical devices, including cardiac pacemakers and heart valves. Medtronic employs about 5,000 people throughout the world and does business in over 75 countries. International production facilities are located in Brazil, Canada, France, Puerto Rico, and Holland. Medtronic corporate headquarters are located in suburban Minneapolis, MN.

## Computer Systems Background

During the years, some of our different departments procured different software packages in order to better perform their jobs. Often these decisions necessitated buying different hardware. An example of this was the decision to buy Express, a decision support tool. At the time, it only ran on IBM mainframes and Prime systems. We opted for the Prime version, which meant we had to buy Prime hardware. Eventually, we had hardware from IBM, DEC, HP, Prime, and Sperry. Some users who had signons on different machines were literally having 2 terminals in their offices. We considered buying duplicate terminals and printers as not a very wise or necessary use of our corporation's resources.

## LAN Requirements and Goals

Based on our particular situation, we decided that a Local Area Network (LAN) would meet all of our requirements. These requirements are as follows (See [1], p. 119):

- reduce corporate investment in vendor-specific terminals, printers, and associated hardware;
- reduce terminal population in cases where multiple terminals were in use in the same work area;
- reduce the printer footprint so printers could be located close to the users (one shared printer takes up less room than a cluster of vendor specific printers);
- utilize terminals and printers better (for instance, our vendor specific terminals were often idle due to insufficient demand);
- allow users to access multiple technologies through a single low-cost, multi-function workstation;
- create an open-ended system for future growth;
- guarantee high performance without the risk of degradation in a heavily loaded system;
- distribute the system fully (this requirement would minimize the down time caused by single component failures);
- support any-point to any-point communications with a wide range of hardware on both ends;
- increase the utilization of the nodes attached to the host computers by means of port contention;
- lower the cost per station by reducing the number of vendors involved, thus opening the door for volume discounts; and
- maximize the use of available resources.

## Broadband Cable System

Earlier than the LAN issues surfaced, we had installed a broadband cable system during a major addition to our

Minneapolis campus. At that time, we needed a transmission medium that was economical, provided a large bandwidth for both data and video, and would allow us to inexpensively move terminals from one location to another. This latter requirement is important as departments are constantly moving and the cost to the company in both time and materials to move dedicated point-to-host terminals would be prohibitive.

With regard to using both data and voice, we actually installed 2 cables in our campus. This was done not only for the convenience of having data on one cable and video on another, but also for redundancy reasons if one of the cables failed. The video cable is used for events that the whole company may wish to see, such as the annual shareholder's meeting, special events with visiting guest lecturers, and special broadcasts of UNITE, an educational program with the University of Minnesota.

The actual hardware components of the broadband network are as follows (See [5], p. 343):

- 1/2 inch broadband cable: this is standard CATV industry cable and offers excellent resistance to noise pickup;
- amplifiers: these can be located wherever necessary, and provide the signal boost necessary to offset the losses which occur during the pathing of the signal;
- power inserters: these deliver power to the amplifiers;
- splitters: when the cable needs to be trunked or branched, splitters are used;
- taps: these are inserted into the cable to "tap off" signals which are then transmitted to equipment using flexible drop cables such as RG6; we have found that one 4-port tap for every 2,000 square feet of office or lab space is a good rule of thumb; and
- broadband modems: these are similar to standard modems, except that they operate at a much higher frequency range (radio frequency or RF); these are especially useful for synchronous terminal connections between host computers and multiplexers or terminal controllers.

As this paper will not deal any further with the hardware issues involved in our network, let it just be added that our broadband system is very reliable: We have yet to experience a system-wide failure in over 5 years of operation.

Since we already owned a broadband cable system, one additional LAN requirement was:

- the LAN must utilize the current broadband cable system.

### LAN Details

The LAN we selected was LAN/1 from 3M (now called VistaLAN/1 and sold by Allen-Bradley). A major feature is its 2.5 megabit data transmission rate. Another is the ease which the configuration can be changed. All traffic between any 2 points on the network passes through a pair of network interface units (NIUs), similar in concept to the way a pair of modems works. There is an NIU associated with each group of ports on the network. Each NIU can accommodate 4 or 8 ports. Each NIU may be programmed for chaining to another port if the called port is busy, accommodating different baud rates, and recognizing different flow-control options. Their flexibility of programming is excellent.

To start a session with the VAX, for example, one would first try to establish a virtual circuit. This may be done in either a general way ("CV" for "call the VAX") or by specific port ("C40,3" for "call node 40, port 3"). If that node is busy, the call will be rotored to the next port on the chain if that port is so programmed. If not, or if all ports are currently busy, a busy message will appear to the user. This possibility of running out of access ports to a specific machine is the largest risk for this type of setup. A general rule we have followed is to allocate 1 NIU port for every 4 expected users of the host system.

While the physical medium is a single cable (typical of CSMA/CD transmissions like Ethernet, for example), the actual method of transmission is token-ring. This is achieved by defining a "head end" at one point on the cable, having all transmissions from it at a high frequency, and all transmissions to it at a low frequency. At the head end, then, a remodulator folds the low frequency signal into a high frequency signal to achieve this ring effect.

The major restriction of VistaLAN/1 is that it currently supports only asynchronous devices. This actually made some tasks easier, such as shared printer driver discussed below. In general, however, this is not as desirable as synchronous protocol support. The major disadvantage is that it is difficult to determine an inactive user who is not using his port currently in order to log him out and give someone else access to the resource. We currently put the onus of this responsibility on the individual host (e. g., "Watchdog" on the VAX).

### Equipment Selection

Since our group would be responsible for maintaining all equipment, we wished to standardize the equipment selection for the following reasons:

- fewer different hardware components meant that we would need to learn how to fix and maintain a more limited set of hardware;
- reasonable list prices, coupled with the possibility of volume discounts, would reduce the corporation's outside expenses;

- established vendors only, preferably with authorized service and sales forces in the Minneapolis area;
- size, appearance, and other ergonomic considerations;
- on-site tests of actual functionality and performance;
- keyboard programmability for terminals; and
- ANSI 3.64 (VT100) compatible terminal equipment.

Our final equipment selections were:

- terminal: TeleVideo 970 (note: the GA970E chip set is needed for VAX compatibility)
- medium speed printer: Genicom 3404
- low speed printer: Epson FX 80/100
- modem: Rixon R212A Intelligent Modem
- plotter: HP7475A 6 pen plotter.

In general, we have been pleased with our choices. The hardest things from a user acceptance point of view were the Rixon modems (the users would prefer the old acoustic coupler modems) and the programmable terminals. This problem with the terminals was caused by user frustration when response was slow or keys were incorrectly programmed. The users would then hit combinations of keys in an effort to extricate themselves from their problems which, unfortunately, usually only exacerbated the problem.

Now three of the particularly interesting features of this setup will be addressed: protocol conversion, programmable terminals, and shared printers.

### Protocol Conversion

As was mentioned earlier, a restriction of the LAN is that it only supports asynchronous devices. Since we had to connect 2 of our synchronous hosts to the network (Sperry 1100/72 and IBM System/3x family), protocol converters became the logical solution. A protocol converter appears to the host as a native synchronous terminal (or group of terminals) while appearing to the terminal as an asynchronous host. On the host side, it is the protocol converter's responsibility to handle tasks usually associated with synchronous terminals: answering polls; buffering the sent and received data; and actually sending and receiving the data. On the terminal's side, the protocol converter handles cursor control; screen attributes such as highlighting, reverse video, and blinking characters; and producing the visual effect of being a synchronous terminal. If the setup is working correctly, the protocol converter should be totally transparent to both the host and the terminal.

In addition to the advantage of connecting synchronous hosts with devices, the other major advantage is the ease of network expansions. Since it is the protocol converters that are configured on the host, changes to the network itself are transparent to the host. Some terminal parameters, such as the Sperry's terminal address (RID and SID), are not accessible to the end user but only to the protocol converter, thus providing a modest increase in protection against malicious people or faulty synchronous terminals. Finally, there is the obvious advantage of the price of asynchronous over the synchronous terminal.

Along with these advantages are some limitations. Some signals to the host, such as carrier detect (CD) and data terminal ready (DTR), can be filtered out by the protocol converter and are never passed from the terminal. Another disadvantage is the delay between keystroke and screen display. Command keys are usually emulated by a programmed sequence of control and/or escape sequences. This means that what the user types is not necessarily what gets echoed back to the screen. On a public data network, there may be additional packet charges involved due to this. The delay of echoing is more of a nuisance than anything else.

### Programmable Terminals

The TeleVideo 970 is, to our knowledge, one of the most completely programmable terminals on the market. (Though it is not manufactured any more, it can still be obtained in the used marketplace.) Except for the central keys with the numbers and letters, all other keys may be programmed to a predefined sequence of keystrokes. (Of course, they all have factory defaults if they are not programmed.) In this way, the feel of the terminal may be made to simulate any given terminal, be it a Sperry UTS20, DEC VT100, or IBM 5251. To facilitate key loading, an "LKEYS" routine, complete with menus, was developed. It resides on our Prime systems and allows the users to downline load their own keys. We currently support 6 production LKEYS files. (In addition, users can further customize the keys by local key loading.) The "personal message" area at the bottom of the screen allows one to have any 9 character or less message one chooses. In order to identify the particular key programming that is currently active, we have programmed this area to contain a message, such as "SPERRY" or "S/3xLD".

Due to the ease of programming the keys and their flexibility for emulating different terminals, all departmental support personnel have TeleVideo terminals in their homes. This makes it an easy task to work at home with a variety of different hardware technologies while using only a single, inexpensive terminal.

Our network also has personal computers attached to it. We use VT100 or VT102 emulator packages, such as Kermit and SmarTerm 100 as well as SmarTerm 240, a VT240 emulator.

### Shared Printers

One of the many benefits of our LAN setup, and to my mind the most elegant, is the ability to share printers across different systems. Due to the size of our company,

we have users of a particular computer system scattered around. Using a traditional approach, there would be IBM printers for IBM printouts, DEC printers for DEC printouts, etc. This would lead to duplication of hardware and the extra associated costs with that.

Our solution was to place printers throughout the corporation that are only attached to the network, not to a particular machine. The exact method for printing varies by machine, but the principles remain the same for all machines. Each machine has a file containing the node addresses of the different "shared printers", as they are called. In addition, there is a print queue on each machine associated with each printer accessible by that machine. When the machine determines that there is printout queued for a shared printer, the following steps are taken:

1. The host machine determines the queue(s) that have output queued to them.

2. The host tries to make a connection with the port on the network associated with a given printer. If the port is already in use (i. e., another system is printing on it), the next printer that has something queued is checked, etc. If no other queues have output queued, the program goes into a wait state and retries later.

3. At this point, the virtual circuit is established. Based on the specific implementation, the host will either (a) pass control to a spooling routine to do the actual printing (as on the VAX) or (b) output the print images by itself.

4. If a spool routine is performing the printing (e. g., a VAX symbiont), the driver must periodically check to see if the printing is done. Once it determines that is is, it will break the virtual circuit and check for more work to do. If there is none, it waits and retries at a later time. If the program is outputting the print images (e. g., Sperry), it will break the circuit when the output is done and hang up the circuit. It, too, will then look for work to do or wait and retry later.

On the VAX, the implementation is through the use of generic queues, one for each shared printer, and one physical queue (TXA7) associated with a physical port on the VAX's backplane. When there is work to do, a connection is attempted between TXA7 and the desired shared printer. If it is made, the generic queue's output is passed to TXA7 and the printing commences.

One of the disadvantages of this system is that only 1 printer can be active at a time. If a user is printing a long report on one printer, all of the printouts in the other shared queues must wait for that one report to finish before they will print. We have selected an informal limit of 50 pages per printout to guard against this very thing. On the VAX, we can drive up to 16 print symbionts with one driver program. On the Sperry, we utilize 1 driver per symbiont, as the program is the symbiont. Therefore, we currently have 3 drivers running which all consume system resources. However, this is a necessary evil to ensure good turnaround time at the printers.

Another related disadvantage of our system is the sensitivity of each printer in the group. If a job is printing and the paper jams, for example, all other print jobs must wait until the problem is resolved. An interesting note about our VAX implementation is that when there is a jam and one tries to find out the printer that is hung, TXA7 appears as the only active printer since the generic queue is stopped after its output is passed to TXA7. Therefore, we write out the currently active shared printer to a separate file before control is passed to the symbiont so we can locate the offending printer if the need arises.

A final disadvantage is that this system cannot be robustly implemented with all systems. We currently have implementations on our Prime, Sperry, and VAX systems. Especially difficult are the hosts that use protocol converters. (The Sperry does not use these for the shared printer drivers; we use asynchronous communications boards instead.) On the IBM System/38, for example, we have real difficulty in detecting network messages and carrier information. The shared printer driver we had on this system was unreliable at best, but with new levels of protocol converter firmware we plan to make another, more successful, attempt in the future.

Despite these disadvantages, the shared printer setup works very well. It is interesting to see a printer start a printout from the Sperry, then pause, then print a VAX report, pause, and finally print a Prime report. Also, we know that the utilization of this printer is far better than would 3 native vendor printers in the same physical location.

## Conclusions

At Medtronic, we needed a flexible and cost-effective networking tool. A LAN was the preferred solution. By installing our LAN on our existing broadband cable system, we have been able to realize the following major advantages:

- reduced hardware costs by not buying vendor specific terminals and printers (for example, when we ordered our VAX 8500, which was our first VAX, no native terminals or printers were ordered, much to the surprise of our sales representative);

- better utilization of existing equipment;

- any-point to any-point communications ease the task of moving equipment around the corporate facilities; and

- by using common asynchronous communication, many types of systems can be accessed using a single terminal, thus reducing the terminal footprint on a user's desk; in addition, shared printers attached to the network can be conveniently located close to the users, reducing the printer footprint companywide.

All in all, we are very pleased with our network.

## References


[1] Dean, John. "Local Area Networking in a Multi-Vendor Environment," *Proceedings of the National Prime Users Group Conference,* June 1985 (St. Louis), vol. 1, p. 117.

[2] Digital Equipment Corporation. *Networks and Communications Buyer's Guide.* July—September, 1987.

[3] Digital Equipment Corporation. *Networking: The Competitive Edge.* 1985.

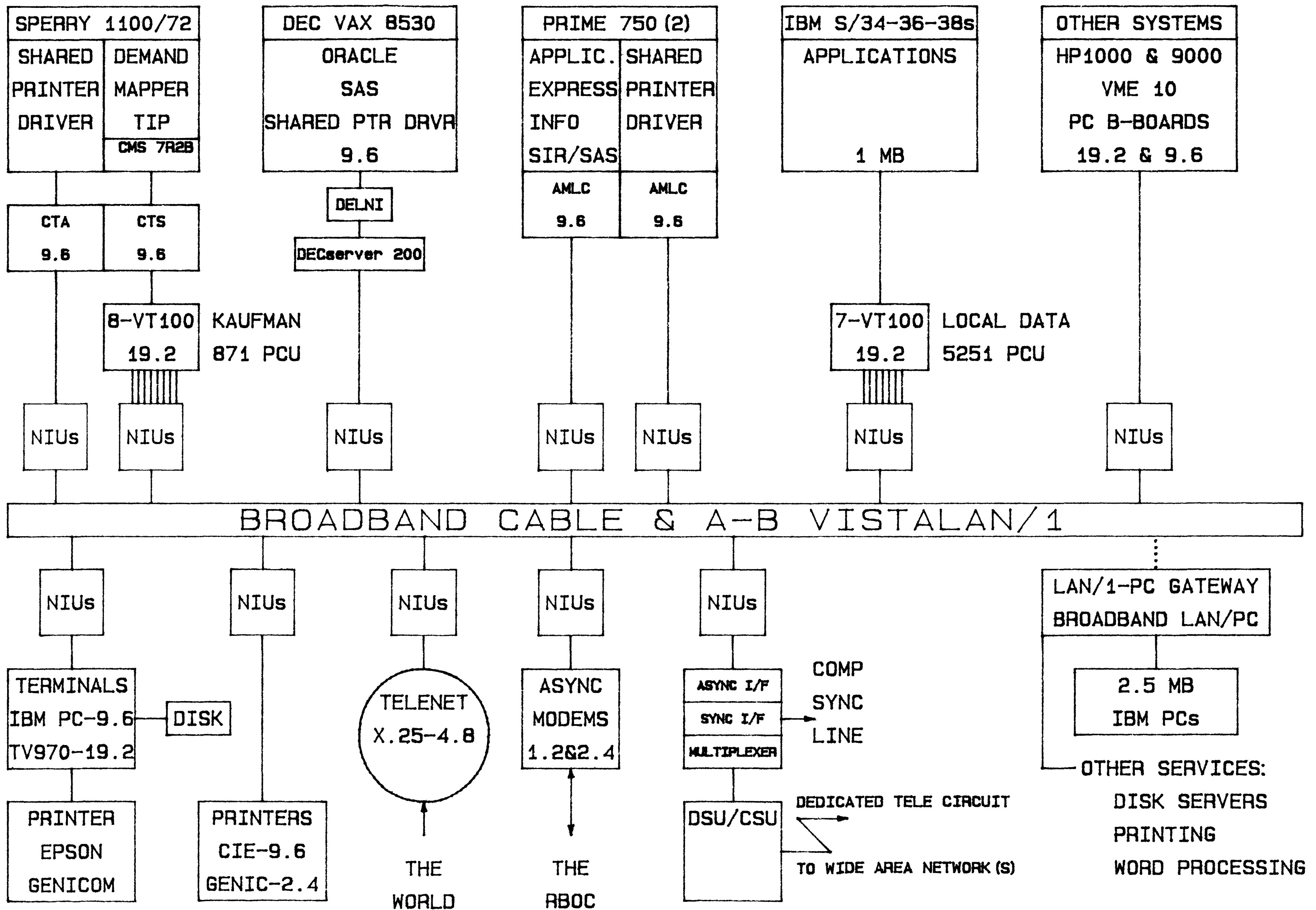
[4] Doll, Dixon. *Local Area Networks.* Session held at Battelle Columbus Laboratories, November 10th, 1985.

[5] Gammel, John. "Local Area Networking in a Multi-Vendor Environment," *Technical Papers of the National Univac (USE) Conference,* April 1984 (Dallas), vol. 1, p. 343.

[6] Gammel, John. "Local Area Networking in a Multi-Vendor Environment - An Update," *Technical Papers of the National Univac (USE) Conference,* October 1984 (San Diego), vol. 1, p. 389.

[7] Micom, Inc. *Recognizing, Evaluating, and Choosing Solutions to Data Communications Application Problems.*

[8] Tanenbaum, Andrew. *Computer Networks.* Englewood Cliffs: Prentice Hall, 1981.


MEDTRONIC, INC. LOCAL AREA NETWORK
SPERRY 1100/72
SHARED PRINTER DRIVER
DEMAND MAPPER TIP
CMS 7R2B
CTA 9.6
CTS 9.6
8-VT100 19.2
KAUFMAN 871 PCU
DEC VAX 8530
ORACLE
SAS
SHARED PTR DRVR
9.6
DELNI
DECserver 200
PRIME 750 (2)
APPLIC. EXPRESS INFO SIR/SAS
SHARED PRINTER DRIVER
AMLC 9.6
AMLC 9.6
IBM S/34-36-38s
APPLICATIONS
1 MB
7-VT100 19.2
LOCAL DATA 5251 PCU
OTHER SYSTEMS
HP1000 & 9000
VME 10
PC B-BOARDS
19.2 & 9.6
NIUs
BROADBAND CABLE & A-B VISTALAN/1
TERMINALS
IBM PC-9.6
TV970-19.2
DISK
PRINTER EPSON GENICOM
PRINTERS CIE-9.6 GENIC-2.4
TELENET X.25-4.8
THE WORLD
ASYNC MODEMS 1.2&2.4
THE RBOC
ASYNC I/F
SYNC I/F
MULTIPLEXER
COMP SYNC LINE
DSU/CSU
DEDICATED TELE CIRCUIT
TO WIDE AREA NETWORK (S)
LAN/1-PC GATEWAY
BROADBAND LAN/PC
2.5 MB IBM PCs
OTHER SERVICES:
DISK SERVERS
PRINTING
WORD PROCESSING

# OFFICE AUTOMATION SIG

## PERSONNEL COMPUTERS AND ALL-IN-1:

## DOCUMENT TRANSFER AND TRANSLATION

**C. F. Stanland**
E. I. du Pont de Nemours and Company
Savannah River Plant
Aiken, SC

ABSTRACT

The Savannah River site is currently utilizing a multi-vendor approach to provide Office Automation. Personal computer workstations include the Apple Macintosh, IBM PC, DEC Rainbow and DEC VAXmate. Users of these workstations typically employ a variety of word processing and graphics packages to produce documents, many of which need to be exchanged in reviseable form with users of different word processing packages.

Since most of these same users are utilizing terminal emulators to access the departmental ALL-IN-1 systems, the need/desire to translate and/or transfer workstation-created documents via electronic mail naturally arose.

We will discuss our enhancements to document transfer capabilities of ALL-IN-1 to support host-initiated file transfers to and from any of the above mentioned workstations. Documents, including attachments, may be uploaded to, or downloaded from, ALL-IN-1 in their native format or with conversion. Document conversions are performed "transparently" using ALL-IN-1 DSAB's or third party packages with attributes such as rulers, soft returns, bolding, and underscoring preserved. Support for translation independent of transfer is also provided and a "Get From Personal Computer" option has been added to the editor menus.

Included in our discussion will be a summary of the supported terminal emulator file transfer packages, the various workstation to VAX hardware connections utilized, and the capabilities of the file conversion packages that are currently available.

### INTRODUCTION

The phone just rang; it was your boss. His superintendent had just called and said that the report originally due next week must be ready in one hour. Your group has been working on the individual sections, but nothing has been pulled together yet. Jane has a great graphic that she's done on the Macintosh, while Bill has his data analysis prepared in a WPS-Plus document under ALL-IN-1. Joe is just finishing up his conclusions using Displaywrite 3 on the IBM PC, while you've been using Word Perfect to write the introduction. To top it all off, your boss, who's office is 20 miles away, wants to add a last minute management overview section and he's a Macintosh Microsoft Word user.

As personal computer and local area networks increase in numbers, scenarios like the one described above become increasingly frequent occurrences. This paper will discuss the approach taken by the Savannah River Plant in addressing the challenges of document transfer and translation.

### BACKGROUND

The Savannah River Plant (SRP) is a Department of Energy facility located on a 350-square-mile reservation outside of Aiken, South Carolina. The plant, which is operated for the government by E. I. du Pont de Nemours and Co. Inc., produces special nuclear materials, primarily for use in the nation's defense programs. The site currently employs about 13,000 workers in technical, production, clerical, construction, and security roles.

In January of 1985, Du Pont management commissioned a study of how the long-term information needs

of the site could be met. The result of this study was a 10-year plan calling for the implementation of a sitewide information system which would help solve problems like the one described above. The final system would include:

- Extensive local area networks
- Personal computers in all work locations
- Direct workstation to workstation and workstation to central computing facility (CCF) access
- Office Automation and CAD/CAM computers

The Computer Projects Department (CPD) was subsequently formed to address the needs of the information system plan. Since that time, one of the major tasks of CPD has been to procure and install personal and office computers and the networks necessary to tie them together.

Currently, local area networks are being installed at major plant facilities. These networks are interconnected via Applitek bridges to the large sitewide broadband system. (There are 41 miles of broadband cable at the site.) Twelve departmental office automation systems, consisting of Digital VAX 8550 and MicroVAX II computers, are in place. ALL-IN-1 is the primary user interface for information system access and approximately 1,800 of the projected 4,400 users currently have accounts.

Objectives

There are a variety of word processors and personal computers in use at the Savannah River Plant. Currently Apple, IBM, and Digital personal computers (pc's) comprise the 4,000 workstations in use at the plant. Of these workstations, approximately 2,000 are IBM (XT's AT's or PS/2's), 1,500 are Apple Macintoshes, and 500 are Digital Rainbow or VAXmates. On these workstations, a variety of word processors are used and require interchange. Additionally, a variety of other applications are in use including graphics, spreadsheets, and databases.

The goal of the document transfer and conversion effort was to allow users to electronically exchange reviseable form documents throughout the site. This was desirable in order to eliminate the existing "foot-net" method of transferring documents - a method impractical when facilities are as much as 15 miles apart. It was also desirable to eliminate the re-typing of documents which occurred whenever two users of dissimilar word processors wanted to exchange documents.

Given the multivendor environment present at the site, another of the objectives of the document transfer/translation application was to allow users to continue to utilize their existing workstations and word processors, whenever possible. It would be impossible to replace all of the existing workstations at once, much less with a single vendor solution. Additionally, it was felt that the training and support workload would be smaller, and automation efforts would be more openly received, if users retained as much of their existing environment as possible.

A final objective was that any document translation solution be an integral component of the site ALL-IN-1 Information System.

## DOCUMENT TRANSLATION

Requirements

As stated above, there were several word processors in use at SRP which require support. These are listed in Table 1:

**Table 1. Word Processors in Use**

| Apple | Digital | IBM |
|---|---|---|
| Macwrite<br>Microsoft Word | WPS-Plus | Displaywrite<br>Wordstar<br>Word Perfect<br>Multimate<br>Microsoft Word |

There needed to be a way to convert a document created using these word processors into any other format, while retaining as many attributes as possible, especially character attributes (bold, underline, etc.) and rulers.

Technical Approaches

There are two approaches which can be taken in order to meet the above requirements. One can either:

(1) Enhance ALL-IN-1 so that it can read/write/convert these document types, or

(2) Develop software outside of ALL-IN-1 that performs the translations, using ALL-IN-1 just to mail the documents.

The first option is the more desirable solution because:

- It would not be necessary to translate the entire document in order to read a portion of it - with ALL-IN-1, translation occurs a line at a time

- ALL-IN-1 has an implicit conversion capability, allowing translation to be performed by a simple copy operation

- Append, merge, list, and other ALL-IN-1 functions would work on any document.

There are disadvantages to option (1) however, which include:

- Digital does not supply sufficient technical information for users to develop and integrate the required conversion mechanisms

- The programming effort would be large and time consuming

- CPD would have to support any custom conversion software

- It assumes access to all word processor file formats

The second option, likewise, has it's advantages and disadvantages.

On the positive side:

- CPD already knew of a vendor who produced a personal computer based system which did most of the required conversions.

- The conversion software need not be dependent upon internal knowledge of ALL-IN-1

On the negative side,

- It would not be a true integration with ALL-IN-1, therefore a mechanism would have to be developed to allow ALL-IN-1 to "deal with" the document types, while not interpreting them.

- The existing software was relatively slow

- A translation would require converting the entire document in order to read a single line

The final solution, as might be expected, is a hybrid of both approaches.

Translation Software

The word processing document translation is performed using two pieces of purchased software, ALL-IN-1 and KEYpak, which minimizes the support and development and support required.

KEYpak

The stand-alone translation system which CPD had experience with is produced by Keyword of Calgary, Alberta. To use that system, floppies are inserted into a disk drive unit attached to a personal computer, and software is run to translate between the source and target formats. CPD approached Keyword with the idea that they port the translation software to run in a VAX/VMS environment in order to translate files which had been uploaded from personal computers. Keyword ported the software to VMS and it is now available as their KEYpak product. KEYpak does not support all translations available on the stand-alone system, but the translations CPD desired are available.

To use KEYpak, an image is run and the following information is provided on the command line: source file name and format, target file name and format, conversion log file name, and optionally, a configuration file name. The configuration file contains special translation directives. A status is returned which indicates the success of the translation.

ALL-IN-1

In addition to Keyword, CPD also met with Digital to discuss adding functionality to ALL–IN–1 which would allow it to read additional word processing formats. [This ALL–IN–1 facility is commonly referred to as a Data Set Access Block (DSAB). Standard ALL-IN-1 has DSAB's for WPS-Plus, ASCII, and DEC DX.]

Savannah River first requested that Digital develop a DSAB which would allow ALL-IN-1 to read and write Macwrite files. Macwrite was selected because it was the dominant word processor on the Apple Macintosh and because Keyword did not provide a Macwrite translation routine. Also, the users of Macwrite were typically persons who would want the ability to read a Macwrite document without waiting on a translation to take place. Digital did develop the DSAB, which is currently available as the "Macwrite Handler for ALL-IN-1."

If we were to be able to use ALL-IN-1 to mail documents that were of any file type, we also required a DSAB which deals with files irrespective of their contents. A "foreign" or "binary" DSAB was already under development by Digital and met just those requirements. When a user attempts to read or edit a document which has a DSAB type of "BINARY," a message appears which reads: "This file cannot be read by ALL-IN-1." Whenever binary files are copied using ALL-IN-1, the copy is done in a block-mode fashion and no interpretation of the contents is attempted.

Supported Translations

Using KEYpak and ALL-IN-1, translations can be performed between any two of the word processing formats in use at the site. This is possible because both translation packages support DEC DX as an input or output format. Because of this, DX can serve as an intermediate format when no direct translation is possible. To convert from WPS-Plus to Word Perfect, for example, the WPS-Plus file would first be converted to DX using ALL-IN-1, then the DX file would be converted to Word Perfect using KEYpak.

Translations supported by each of the conversion packages are listed in Table 2.

**Table 2. Supported Translations**

| ALL-IN-1 | KEYpak |
|---|---|
| WPS-Plus | Displaywrite |
| ASCII | Wordstar |
| Binary | Word Perfect |
| Macwrite | Multimate |
| | Microsoft Word |
| DEC DX | DEC DX |

**ALL-IN-1 Integration**

The next step which we faced was to integrate the DSAB and Keyword conversion mechanisms into ALL-IN-1. This meant that we would have to incorporate, to some extent, the following support for those document types ("foreign formats") that ALL-IN-1 could not read or convert:

- File cabinet storage
- Use as mail message attachments (e.g., Wordstar attachments to WPS-Plus messages)
- "Gold-G" from the editor, with automatic conversion
- A "Show Document" function to show the document type of documents (and any attachments)
- Convert document support
- Translation during document transfer to or from a workstation

The first task to be performed was to allow ALL-IN-1 to handle and distinguish the various "foreign format" documents. For compatibility with electronic mail, this also means that the format of an attached document must be discernable at the receiving node, provided it is running the same software. To accommodate this, it was necessary to modify ALL-IN-1 (i.e., the tables within OAET.MAR) to define a DSAB for each of the formats to be supported. This required specification of a DSAB name, a default file extension, and the routine to be called to read or write the file. Since there weren't any action routines which would allow ALL-IN-1 to process the files, the action routine was specified as being the same as that for the "binary" DSAB. This ensures (1) that the user is told if he tries to read/edit an "unreadable" document, and (2) that copies do not affect the document.

Next several tables were set up as indexed files accessed by ALL-IN-1 entry forms. The document file specifies all document types the DSAB required, whether or not the file was intrinsically supported by ALL-IN-1, and the KEYword translation support and attributes. A conversion file specifies an action to perform (KEYpak or ALL-IN-1 translation) for each possible conversion. Conversions not directly possible (e.g., WPS-Plus to Wordstar) are handled in a two-step process with DEC DX being the standard intermediate format.

The assignment of a DSAB to each document type ensures proper file cabinet storage and electronic mail transmission. The data tables are used during operations such as "Gold-G," "Convert Document," and "Document Transfer" to determine the proper routine to execute to convert the document to the output format specified by the user.

Document Transfer

Requisite for any VAX-based personal computer file translation is a good file transfer package. At the time CPD was formed, there were already a few terminal emulator/file transfer packages in use. Our task was to

integrate the packages under the ALL-IN-1 document transfer subsystem in as transparent a fashion as possible.

Users are required to provide three pieces of information in their user profiles which are related to document transfer: workstation, emulator, and hardware connection. The workstation field indicates whether the user has an Apple, IBM, or DEC personal computer. The emulator field indicates which terminal emulator/file transfer package is being used, and the hardware connection indicates either an Ethernet or serial connection. These values are used to select the forms and command files necessary to perform the file transfer.

For Apple file uploads, the user is first prompted for the file type (e.g., Macwrite, binary, etc) and ALL-IN-1 document filing information (i.e., destination folder and title). A dialog box then appears and the user indicates which file is to be transferred through standard point and click mechanisms. The file is transferred, translated if necessary, and the ALL-IN-1 document created. For MS-DOS workstations, the procedure is similar, except that instead of a dialog box, the user is asked to input the full file specification.

For file transfers to the workstation, the procedure is roughly the reverse of the above. The user first selects a document to download. He then indicates the desired format of the document that is to be created on the workstation. For the Macintosh, he is presented with a dialog box and indicates the output file and directory. For the MS-DOS systems, the user specifies a full target file specification. If the document has attachments, the user is asked to specify which combination of original document and attachments are to be transferred. Additionally, he is given the option of specifying a single or multiple output files.

The desirable characteristics for a standard terminal emulator were those which would allow the most seamless integration with ALL-IN-1. These included:

- Supports host-initiated file transfers
- Allows specification of host output file characteristics
- Efficient as well as fast
- Robust in a variety of network connection environments
- Inexpensive
- Available on all CPD-supported workstations
- Has or plans REGIS graphics support
- Has or plans Ethernet support

After having investigated many packages, we chose pcLINK (from Pacer software) as the emulator we recommend to new users, although our document transfer enhancements (except for Ethernet support) work with several terminal emulators, as shown in Table 3.

**Table 3. Supported Terminal Emulators**

| | Apple | Digital | IBM |
|---|---|---|---|
| Serial: | pcLINK<br>MACterminal | pcLINK<br>Poly-COMM | pcLINK<br>VTERM |
| Ethernet: | pcLINK | pcLINK | pcLINK |

## CONCLUSION

Now, let's re-visit the scenario that we started with.

Jane uploaded her graphics from the Macintosh into a binary document in ALL-IN-1 and mailed it to you as an attachment to her message. Bill mailed you his WPS-Plus conclusions and Joe sent his Displaywrite 3 document. You file all the attachments in their original format and then begin composing a mail message to your boss. When it's done, you attach your introduction, Bill's analysis, Joe's conclusions, and Jane's graphics. You then send the message to your boss.

After your boss reads your message, he performs two document transfer operations. For the first, he downloads all of the text files into a single Microsoft Word document on his Macintosh. Then, he downloads the graphic. He adds his overview to the report and then prints the report and graphic on his laser printer. With 5 minutes to spare, he carries the document upstairs to his boss.

---

The information contained in this article was developed during the course of work under Contract No. DE-AC09-76SR00001 with the U. S. Department of Energy.

# Basic Networking for Office Automation

Robert Gary Mauler
and
Valerie Cabral Mauler

Westinghouse Electric Corporation
Baltimore, Maryland

**ABSTRACT**

**This paper will take the mystery out of networking Workstations for non technical people. Based on experience gained in networking over 250 PC and VAX Workstations, guidelines and recommendations will be made to help simplify the planning, installation, and check out of an Ethernet LAN. The goal of this paper is to help the poor soul who is told to "...automate our office and network our PC's while you are at it."**

After working with Ethernet since about 1982, and attending a lot of DECUS Networking sessions, we are surprised to find that there are still a lot of people out there that either don't understand or feel comfortable when designing or working with a network system based on Ethernet. The goal of this paper is to help the not so technical person become more familiar with Ethernet wiring systems, thus realizing that there is no magic going on. If you know and follow a few basic rules you too can successfully network your office.

Networking an office should be approached in an organized manner. Hopefully the network you install will be around for a long time, and so it is important to build a solid foundation upon which you can add. The first thing that needs to be done is what you are doing right now, that is learning about the topic and planning your network. Secondly, you must either install the network hardware yourself or at least supervise the installation to make sure it is installed per your design (you don't need surprises later on when you find out that an installer took a short cut). Thirdly, after installation of the networking hardware, the cabling and other active devices must be checked out to verify proper operation. And finally, you need to implement a network management system both to provide diagnostics capability as well as to provide statistics for planning future growth of the network.

This paper will deal exclusively with the subject of office networking using Ethernet. That is, we will discuss how to install a network in an office environment using Ethernet protocol and RG-58 50 ohm cable between the desktop workstation and the wiring closet. One of the first confusing things that you may encounter is the many terms used by vendors to say what we just said. You will hear terms such as ThinWire, ThinEthernet, Cheapernet, and even 10BASE2 used to describe Ethernet using RG-58 cable. In order to be consistent in this paper we will use the term "ThinWire", which is Digital's terminology. Another way of looking at ThinWire is to compare it to "Thick Ethernet" or the standard Ethernet RG-225 trunk cable. The main difference is that ThinWire is smaller in diameter, much easier to pull, and cheaper (Cheapernet). The size is the major factor that makes it practical to wire Ethernet right to the workstation on the office worker's desk. The other major difference between the two cabling schemes is the difference in cable connectors. ThinWire uses what is called a "BNC" type connector while the RG-225 cable uses an "N" style connector. BNC connectors are pictured in Figure 1. The BNC connector is what you might call a "twist & lock" since the connector is pushed on and then rotated 90 degrees to make the connection. On the other hand the "N" style of connector is threaded on. When it is necessary to connect a computer to the network there is a difference again based on the type of connectors. Typically

when using the RG-225 cable a piercing type of tap is made on the cable and then a device called a transceiver is clamped onto the cable. In the case of ThinWire, BNC type connectors are used. A BNC "T" connector allows two segments to be connected together and also provides a male connector to connect to the workstation's onboard transceiver.

Once you understand the terminology, it time to start learning the rules of networking. One source of these rules is a document from Digital Equipment Corp. entitled "Networks and Communications Buyer's Guide". Although it is an excellent source of information be aware that it only represents networking products from Digital. The same will be true of almost any document that you receive from a vendor. You need to shop around to get the big picture; there's more than one way to network a building.

When it comes to office networking there are only a few rules that you need to remember. First, the total length of a ThinWire segment is 185 meters or 606 feet maximum per segment. Although that may not seem like a whole lot, in actual practice it is more than enough. The second rule is that there can be no more that 30 nodes or devices maximum per segment. The 30 node rule also implies that there should be a maximum of 60 connectors per segment (30 nodes X 2 connectors per node). Here again, with the use of multi-port repeaters, as you will see later on, your network will probably never reach the maximum of 30 nodes on a segment. Third, ThinWire segments are to be connected together with BNC "T" connectors which are then directly attached to the workstation. In other words there should NOT be a length of cable between the workstation and the BNC "T" connector. In addition, if a workstation is removed permanently then it is preferable to replace the BNC "T" connector with a BNC barrel connector. The fourth rule is that both ends of an Ethernet segment of wire must be terminated with a 50 ohm terminator. In addition, one end and only one end should be grounded. All other connectors along the segment should be isolated from ground by using the plastic shields provided with your network interface card. In the case where you are using a multi-port repeater from Digital Equipment Corporation (DEMPR), then the requirements for 50 ohm termination and a ground on one end of the segment are provided for by the DEMPR. One last note before we leave these rules is that Ethernet is a multi-drop configuration and not a ring type wiring scheme. Therefore you must insure that you don't form a ring when wiring with ThinWire.

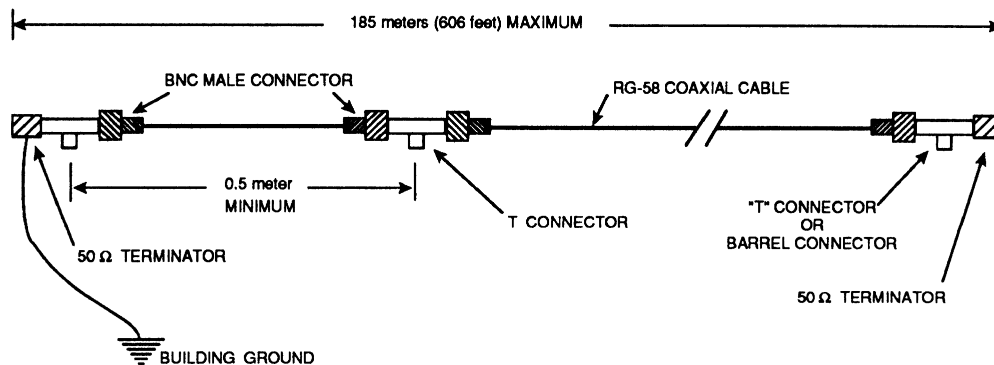



Figure 1

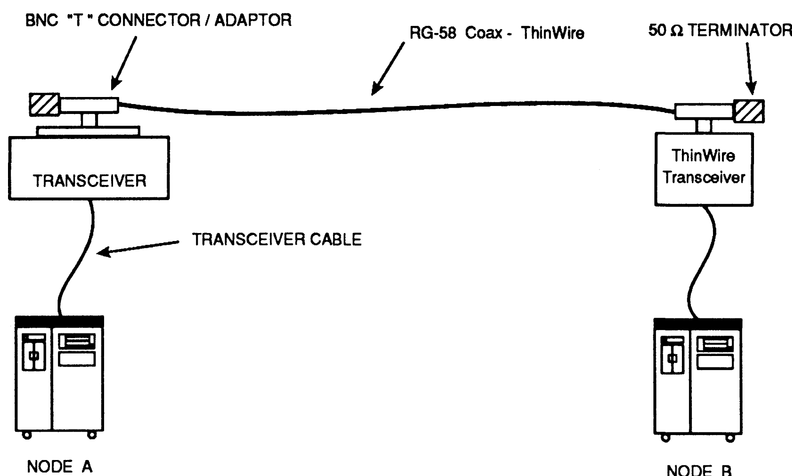



Figure 2

To form the simplest network it is going to take two nodes. There could be a few deviations such as one node being a terminal server and one node a VAX, but for this discussion we will assume that both are MicroVAX systems (see Figure 2). In the case of a MicroVAX, and for that matter most minicomputers, you will find on the back of the cabinet a 15 pin miniature D connector (looks like a small RS-232 connector) labeled Ethernet. The first item that you will need is a transceiver cable. The purpose of the transceiver cable is to provide DC power and digital data signals to a Ethernet transceiver. Here is where the real fun starts. The "transceiver" has had many flavors over the years. Back in the beginning it was just called a transceiver, but then DEC, Intel, and XEROX got together and came out with Ethernet Version 2. So then you had two types of transceivers, a Version 1 and a Version 2. To further complicate matters, another group of engineers got together and now we have an 802.3 standard. The main difference between these three types of transceivers is the "heartbeat test" or SQE (Signal Quality Error signal). Version 1 and 802.3 transceivers do not use the SQE signal while the Version 2 transceiver does. It is important to use the correct type of transceiver for a particular Ethernet device or controller. In some cases when you connect the wrong transceiver it just won't work. But things can really get bad if you connect a transceiver with heartbeat to a controller that is not expecting the heartbeat and mistakes it for a real packet collision. In this case the device will sort of work but you will wonder why the network is so slow.

One area of confusion that we have noticed is in how to mix Thick and Thin types of transceivers on a network. In Figure 2 there are two types of transceivers shown. The transceiver on the left represents a Thick transceiver that has an "adapter" to make it compatible with ThinWire wiring. There are several ways to make this transition. One very expensive scheme used by DEC is to employ what they refer to as a "Loop Back Transceiver". This method requires that the Loop Back Transceiver be modified by removing it's 50 ohm terminators and replacing them with "N" barrel and "N to BNC female" adapters. For those who are budget conscious there is another approach. AMP Corporation, the makers of most of the media attachment devices for DEC and other Ethernet vendors, offers an adapter that simply replaces the Thick cable clamp/tap assembly with a new assembly that provides a BNC "T" connector for ThinWire cabling. A big advantage to the AMP method is that the number of connectors in the cable path is reduced by four. Not only is this an obvious cost savings but there are less electrical RF joints to attenuate the signal and otherwise cause trouble.

The Transceiver on the right in Figure 2 is an example of a ThinWire Transceiver. A ThinWire Transceiver is one that is designed and built to connect directly to ThinWire cabling without the need for any adapter. If the network you are installing is in an office environment then you should take advantage of these smaller and lower cost devices.

As people see the advantages of networked workstations, the network will quickly grow beyond a single work group. A device called a Multi-Port Repeater is used to connect a building Ethernet Trunk cable to the ThinWire segments running to desks (see

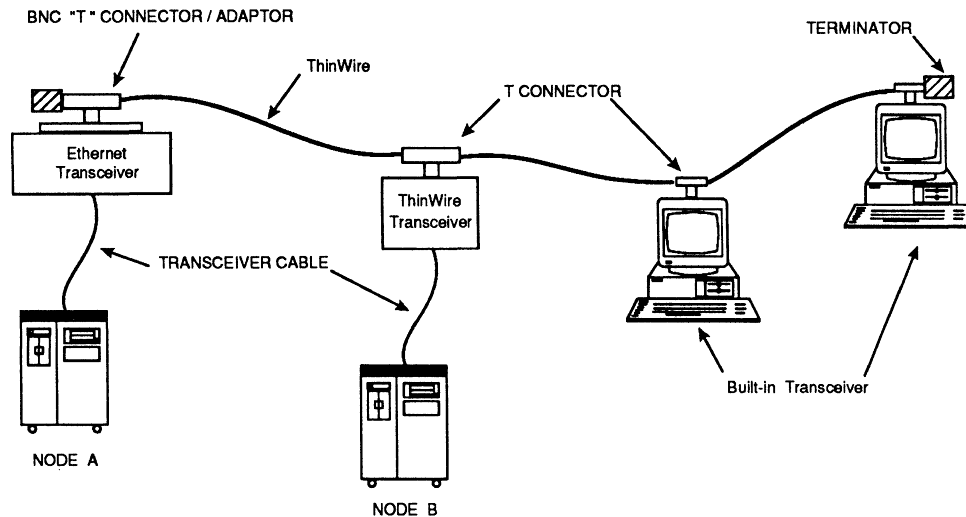



Figure 3

The network that is shown in Figure 3 takes our basic network one step further. We have now added two workstations. It is apparent from the illustration that there are no transceivers associated with the Workstations. But there must be transceivers, since any Ethernet connection is through a transceiver. In this case the workstation has its transceiver electronics built onto the Ethernet controller board or in some cases the CPU, memory, I/O, and network controller are all on one large motherboard. The workstation will have a BNC female connector protruding from the rear of the cabinet to which a BNC "T" connector is attached. If the Workstation is physically the last node on a segment then one port of the "T" connector is terminated with a 50 ohm terminator. Otherwise the "T" connector is placed in a "series" fashion.

Figure 4). Digital Equipment Corporation calls their device a DEMPR which stands for Digital Ethernet Multiple Port Repeater. This type of device is available from other vendors besides DEC, but for simplicity I will use the term DEMPR in this paper. The DEMPR should be placed in a wiring closet located such that the farthest workstation is within a 600 foot radius of it. A DEMPR has 8 BNC female connectors to which 8 ThinWire segments can be attached. If a port is not connected to an active segment then a 50 ohm terminator should be installed. An 802.3 type transceiver (non heartbeat) and cable is used to connect a DEMPR to the Ethernet Trunk cable. The transceiver used for this application would be the piercing type. A maximum configuration for a DEMPR will allow up to 232 nodes in a 600 foot radius to be connected.

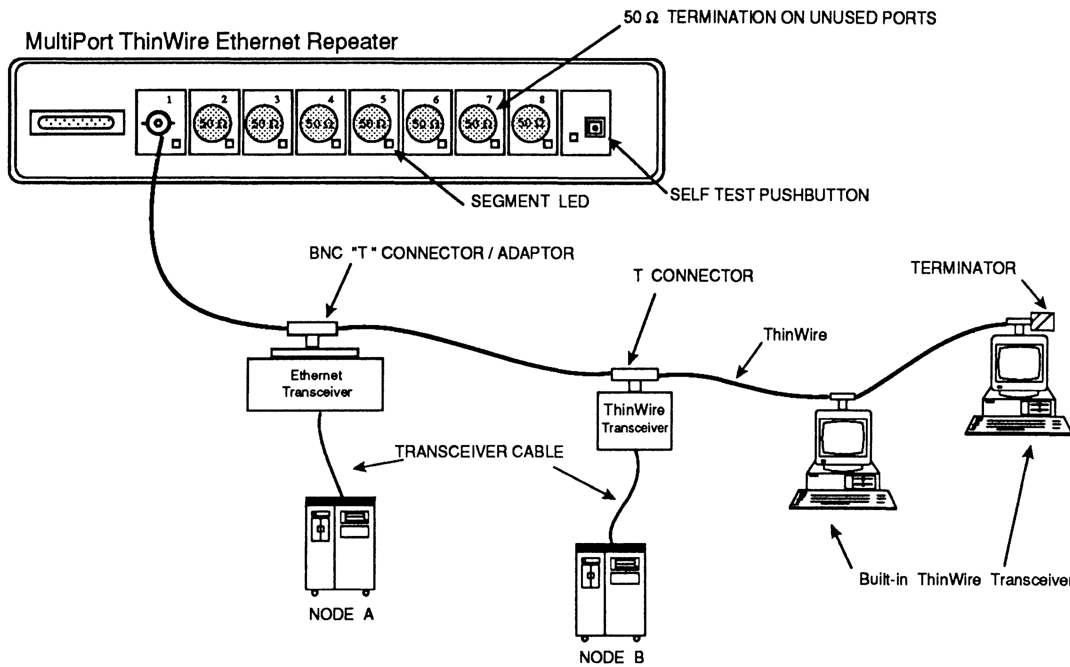



Figure 4

The DEMPR can also be used in a "local" configuration which means that it is not connected to an Ethernet Trunk cable. This type of configuration is useful in small buildings. For example, in a small three story building the wiring closet could be placed centrally on the second floor. From the second floor wiring closet ThinWire segments would be pulled to all three floors. If one DEMPR is not enough, up to eight can be connected using DEC's DELNI (Digital Ethernet Local Network Interconnect) or a similar type device. An example of this type of configuration is shown in Figure 5. With this type of configuration it is possible to connect up to 1,856 nodes.

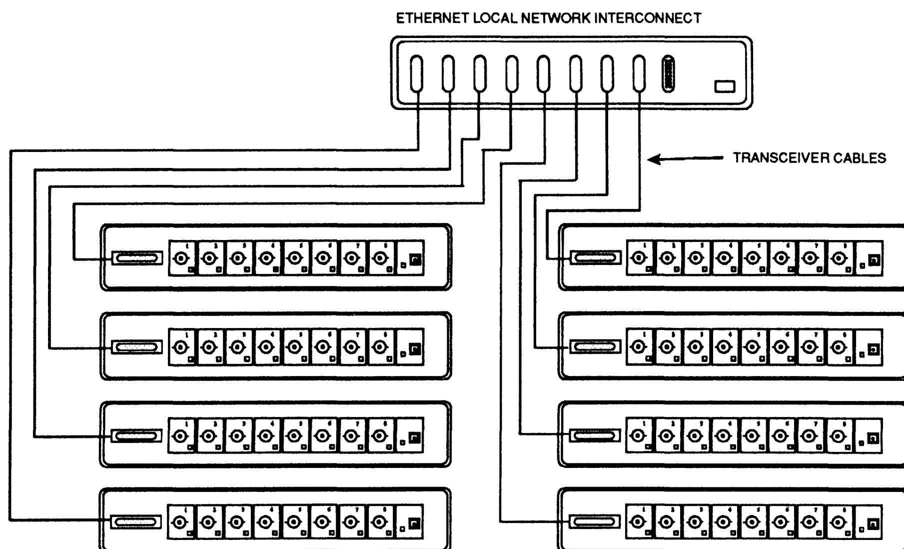



Figure 5

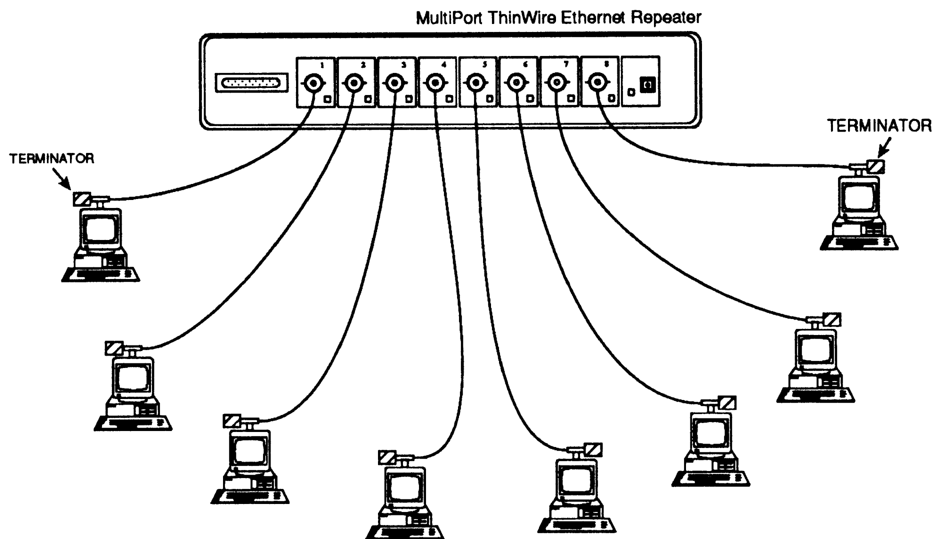



Figure 6

Up to now we have been talking about workstations that are connected in a serial fashion. Recently there have been announcements by DEC, 3COM, and SynOptics in the area of twisted pair wiring plans for Ethernet. The motivation for this is based on the fact that most buildings already have telephones installed that use twisted pairs. Therefore, there are usually twisted pairs already available at a site. In addition, the twisted pair technology is mature and well understood by installers and maintenance personnel. An advantage of the twisted pair wiring scheme is that it is a point-to-point connection. That is, the workstation is connected directly to the network through a repeater. This type of wiring makes it very easy to troubleshoot problems because there are only three pieces to worry about: the workstation, the copper wire, and the network repeater. In Figure 6, we show how a point-to-point configuration can be connected using ThinWire as the media. The picture would look almost the same if you were to use twisted pair media except that it would be necessary to add two adapters to convert from ThinWire to Twisted Pair and back again. These adapters would be placed between the workstation and the DEMPR.

As usual there are trade-offs to be made when deciding between serial and point-to-point wiring plans. But it usually comes down to what your organization is willing to pay for. It should be easy to see that in the point-to-point wiring scheme, the cost of the DEMPR and its transceiver is only divided between 8 workstations. Whereas for the serial wiring plan the cost of the DEMPR, etc. can be divided among as many as 232 Workstations.

Now it is time to look at a practical application for all of the Ethernet devices we have been talking about so far. Figure 7 shows the layout of a typical office that you might be required to network. This example will allow us to explore ideas for placing computer equipment around the office, as well as several of the wiring schemes that are possible. Starting at the lower left corner of the floor plan you see a computer room. Although it is not necessary to have a computer room for a small MicroVAX, it does provide a central location not only for the computer but also for the user manuals, software distribution media, and storage of backup tapes. Another advantage is that the Server is out of high traffic areas where people might have a tendency to "push buttons". The next area along the bottom is a cubicle set aside for shared printers. This is definitely a necessity since printers require supplies of paper

as well as a table to hold print-outs waiting to be picked up. If your printers are within approximately 100 feet of the Server, then they can be hooked up using standard RS-232 wiring. If the distance is much farther or the VAX server does not have RS-232 ports then an Ethernet Terminal Server such as the DECserver-200 will be required to make the connection.

there are no workstations currently installed but a drop is installed for future growth. From there segment B is routed through the ceilings and down the walls of the offices along the top of the floor plan. In each office the ThinWire cable exits the wall through a face plate and attaches to a BNC "T" connector on the workstation. From there it continues on to the next office.

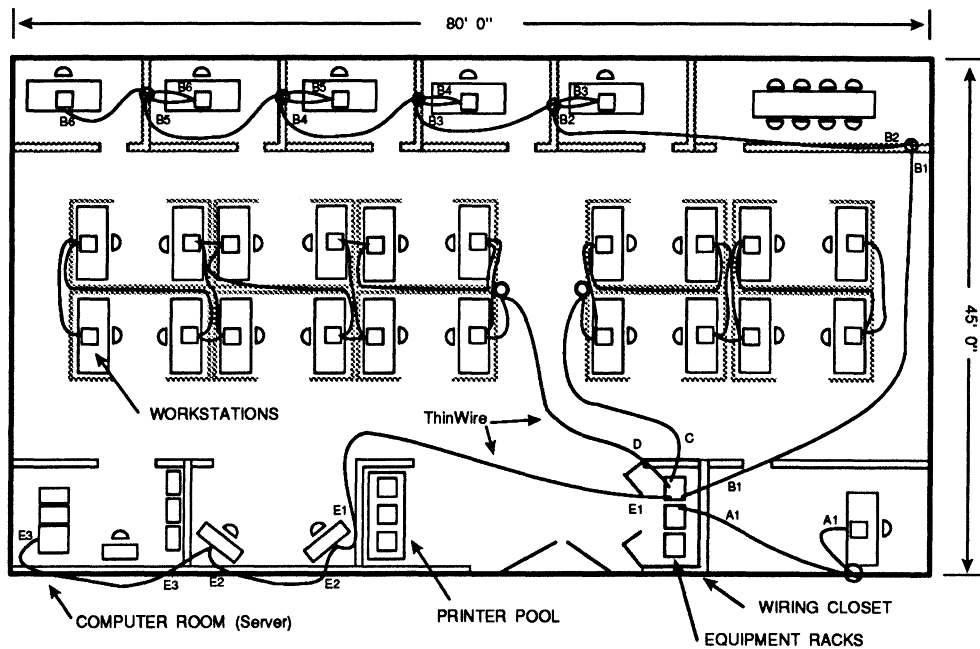



Figure 7

The next area that is shown is a wiring closet. We feel that all communications wiring should be pulled back to a wiring closet. The wiring closet should be large enough to handle both data and voice patch panels as well as data networking equipment such as modems, repeaters, bridges, and terminal servers. In addition the closet should be well ventilated and have independent power outlets. The wiring closet becomes even more important when you want to share twisted pairs with the telephone people so that you can run Ethernet over their unused pairs.

All ThinWire segments originate from the wiring closet. The first segment in our example is labeled A. This is wired as a point-to-point segment since it was convenient and there were no other nodes close by. Segment B starts out by going to a conference room where

The middle of the floor plan contains desks divided into cubicles using free standing partitions. In most cases the ThinWire is dropped down from the ceiling through some type of conduit to floor level. Then if you are lucky you will be able to route the ThinWire through data wiring troughs built into the cubicle partitions. If your partitions do not have a built in wire trough then you will have to improvise another way to keep the data cable off of the floor so that it will not be eaten by the cleaning crew's vacuum cleaners or floor polishers. As before, the workstations are connected in a serial fashion using BNC "T" connectors.

In this example, you will notice that the segments on the average have about 8 workstations in series. In addition, the typical total segment length is about 300 feet. In both cases we are not anywhere near the design

limits for ThinWire. By using DEMPR's to limit the number of workstations in series, it is possible to have multiple segments that are electrically isolated from one another. That is, if one of the office ThinWire segments gets cut by a vacuum cleaner, at worst it will only take out 5 to 10 workstations. It is true that it will take a little longer to isolate the problem since there are 10 to 20 pieces of coax per segment to check, but the coax is readily accessible in the wiring trough in the office partitions. Based on our experience over the past five years the failure rate is very low and once a problem is reported the failing part can usually be located within 15 minutes. By including short serial segments in an overall point-to-point scheme we have found the best compromise between the economy of serial wiring and the convenience of point-to-point.

obstacles such as fire walls, air conditioner ducts, etc. At the conclusion of the installation the design drawing and the as built drawing should be compared so that an accurate record is produced and filed for future reference. We would also suggest that a form such as the one in Figure 8 be used to build a data base on each segment that is installed. The information on this form will be invaluable to the person who may one day want to expand the network or who just wants to locate a problem with one of the workstations. As your network grows to the point where you want to use network management tools such as a LAN Analyzer or DEC's EtherNIM, then this table will allow you to easily find and identify nodes on the Ethernet cable.

Wiring Closet Location ____________________ Sheet ________ of ________

Multi-Port Repeater ID ____________________ Date ____________________

| SEGMENT ID | LENGTH | RUNNING TOTAL LENGTH | # CONNECTORS | RUNNING TOTAL # CONNECTORS | NODE NAME | NODE ADDRESS | ETHERNET ADDRESS | CPU TYPE | USER NAME | PHONE # | WORKSTATION LOCATION |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |

Figure 8

Keeping good records of the physical layout of your network cannot be over emphasized. There are three phases to this process. First, as you do your network design, estimates of cable lengths and placements must be made and recorded on the wiring plan drawing. Then as the cable is pulled the installer should record any modifications that have to be made because of unexpected

This brings us to the subject of network trouble shooting. What if it doesn't work? We believe that the best approach is to be positive, assume that it will work, and try to load your networking software. Nine times out of ten everything will be fine. But if the network does not come up, pay attention to any error messages you get on your terminal for possible software configuration errors.

Although programmers may wish to disagree, we have found that it is usually a software problem. If you happen to experience one of the rare occasions where the hardware does fail then it is time to take a look at the hardware. Fortunately, the engineers that designed a lot of the networking hardware on the market were very generous with LED's. Just about every active network device that I have used has had it share of status indicators. We recommend reading the instruction manual that comes with your particular set of equipment to see just what kind of information can be determined by observing the LED's.

These simple techniques to trouble shooting will resolve most hard failures. If you are experiencing intermittent or network performance types of problems, then you will probably need to use a device called a LAN Network Protocol Analyzer. For example, if it takes a lot longer to transfer files than you think it should at 10MB/sec then a LAN Analyzer can help by showing you statistics on network traffic. For example a good analyzer will allow you to determine the number of collisions per second, lost packets, jabber nodes, average packet sizes, etc.

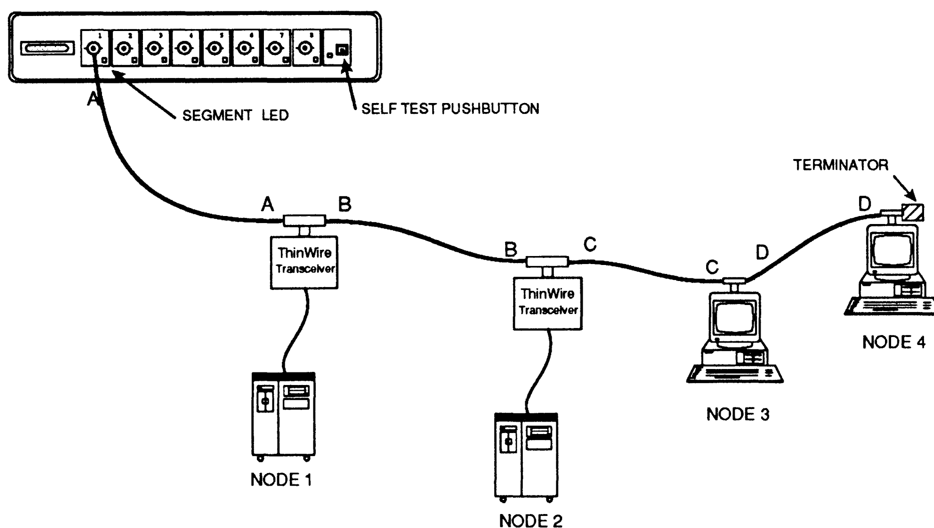



Figure 9

The "process of elimination" is a technique that will work in cases where there is a hard failure. By this we mean that nothing is going across the segment and a SEGMENT LED status indicator is probably lit on a DEMPR. Refer to Figure 9 to follow along with this process. The first step is to disconnect segment B from NODE 1 and replace it with a 50 ohm terminator. Then reset the SEGMENT LED by depressing the SELF TEST PUSHBUTTON and then letting it go. If the SEGMENT LED is on then segment A must be the problem. On the other hand if the SEGMENT LED remains off then that segment is working and you did not locate the problem. Then you must reconnect segment B and repeat this procedure on Node B and so on down the line until you locate the problem.

Although it is difficult if not impossible to record on paper everything that we have learned about networking in the last five years, we hope that by pointing out a few areas where there is potential for confusion, newcomers to networking will feel comfortable with Ethernet wiring in an office environment.

# Office Automation Security: Closing the Doors to Your Computer System

**Robert A. Clyde**
Clyde Digital Systems
Orem, Utah
January 6, 1988

**Abstract**

Since today's organizations store information assets on computer systems, it is essential that an organization safeguard its computer systems. Eighty-seven percent of security problems comes from current or former employees, while only 13% comes from outsiders. This implies that an effective security system must solve the problem for insiders as well as outsiders. There are numerous common oversights that leave open doors to a computer system. The ways to correct these oversights include having sufficient physical and media security, properly managing passwords, controlling unattended logged-in terminals, using sufficient access controls, immediately applying software updates, defending against Trojan horse attacks, and continually monitoring the system for security problems.

## Introduction

Information is an asset. The destruction, alteration, or disclosure of information can be very damaging to an organization. In the last twenty years, offices have become much more automated, and as a result most information is now stored on computer systems. Without the proper safeguards for these systems, an organization places its information asset at risk.

Interestingly, most security breaches do not come from outside of an organization; they come from inside the organization. A recent survey [1] indicated that 81% of the security problems comes from employees and 6% comes from former employees. Only 13% comes from non-employees. This is particularly important because many sites have the mistaken impression that if they can just protect the system from outsiders they will have solved the security problem. In reality such a single-minded approach to security ignores the largest part of the problem.

Many problems are honest mistakes on the part of users. Other problems represent overt abuse. Both mistakes and abuse can be security problems and represent substantial damage. The various types of security problems that can occur fall into the following categories [2]:

- Denial of Service—The computer system becomes inoperative or no longer responds in a normal fashion.
- Information Loss—Information is lost.
- Integrity Loss—Information or programs are modified or corrupted.
- Information Compromise—Sensitive data or programs are stolen or improperly disclosed.
- Resource Exploitation—The computer system is used to achieve objectives outside the authorized purpose for the system. For example, an employee without authorization uses an organization's system to perform computations for outside consulting.

## The Open Doors

Most computer systems come with some security built in. Adding additional security hardware and software can often increase this security. However, in many cases security problems occur because of common oversights which leave *open doors* to the computer system. Therefore, adding additional security capability to a system must be complimented with a knowledge of what to do with the increased capability. The purpose of this paper is to describe some of the common open doors, listed below, and then to explain how to close these doors.

- Insufficient physical and media security
- Improper password management and control
- Unattended logged-in terminals
- Insufficient access controls
- Failure to immediately apply updates
- Vulnerability to a Trojan horse attack
- Lack of security monitoring

### Insufficient Physical and Media Security

The first place to start with security is to look at the physical security of the system and media (e.g., magnetic tapes and disks). If the physical security is inadequate, then unless the data is encrypted, it is probably at risk. Doing the following things will help close this open door:

- Close and lock the real doors to your computer. With certain office computers (e.g., PCs and work stations) this can be difficult. In these cases it would not be wise to keep sensitive or vital data on such systems unless it is encrypted.
- Keep a written log of people entering the computer room. This log will be essential in the event of a security problem.
- Insure that the console is secured. If the console is located outside of the computer room, be sure that it is secured. For example, on a VAX this would involve setting the console disable switch. Otherwise, someone could interrupt the system and insert harmful logic into the running operating system.
- Store disks and tapes securely. Since off-line media contain information assets, the storage of such data should be secure. This may involve locking media in a safe or vault room (preferably fire-proof).

### Improper Password Management and Control

One of the most common open doors is improper password management and control. In fact some systems do not require passwords at all. On other systems the passwords are particularly easy to guess or derive. Also, many users are never trained in password control. As a result they write passwords down, rarely change passwords, or choose obvious passwords. This section describes methods for proper password management and control.

#### Change the default passwords

Once a new system is installed, it often has a number of default accounts and passwords. For example, versions of VMS prior to 4.6 had default passwords for three system accounts. The first thing a system manager should do is disable these accounts or change the passwords. If a system is a turn-key system, it may very likely contain default passwords for other accounts. On a VMS system these accounts can be found by running AUTHORIZE and executing the SHOW */BRIEF command [3].

#### Make the initial password pre-expired

Whenever a new user account is added, the system manager should make the initial password pre-expired. This forces the user to choose a new password. Then the system manager should *tell* the new user the initial password. This password should not be written down or mailed via electronic mail. If it is, others may see the password. Also, the initial password should not be predictable. In other words, it should not be the same or follow the same pattern for all new users. As soon as the user logs in, the user will have to change the password since it is pre-expired

#### Require users to change passwords periodically

The system should require users to change their passwords periodically. On a VAX/VMS system this can be done via the AUTHORIZE program [3]. The site is able to select the period of time between forced changes. However, VAX/VMS does not prevent a user from changing a password to something new and then changing it back to the old password. Nevertheless, programs can be written to check for such an occurrence.

#### Choose secure passwords

Many users make poor password choices. The ideal password should be easy to remember and impossible to guess. Users can follow the simple guidelines below to select good passwords.

- Never use names for passwords.
- Never use single words.
- Random strings of characters are secure but hard to remember.
- Nonsense words are secure and easier to remember. This is the type of password that VMS's password generator makes [3].
- Multiple words are relatively secure, and they are easy to remember. However, be careful not to choose common phrases.

#### Explain password management to users

As part of new user orientation, the system manager or system security administrator should explain password management. This explanation should include the following guidelines:

- Change passwords periodically.
- Choose secure passwords.
- Do not use the same password on multiple systems. Otherwise, if someone cracks your password on one system, your password is compromised for other systems.
- Do not write a password down.
- Do not tell others your password.
- Do not store a password in a terminal's answerback or in a PC's terminal emulation program.

### Unattended Logged-in Terminals

One of the easiest ways to break in to a system is simply to take a stroll around the office and find an unattended terminal which has been left logged in. Now the intruder does not even have to know the password in order to access files. This is particularly dangerous if the unattended terminal is logged in to a privileged account.

Users are normally instructed to log out when a terminal is not in use. However, often a user must leave a terminal while a program or application is running. Since it may be very difficult and wasteful for the user to exit the program or application, the user leaves the terminal logged in. For example, a user who is executing a report program that has been running for a half an hour and still has 10 minutes to run before completion, will probably not abort the program and then log out before leaving the terminal.

#### A terminal locking mechanism

A site needs controls to handle users who leave terminals logged in. One method is to have a program or device which detects idle terminals and stops the user's process (i.e., logs the user out). This works, but it tends to upset users, particularly those who are in the middle of a program or application. A better solution is to have a program or device which can lock the terminal even if it is currently running an application [4]. Such software or hardware should be able to provide the following features:

- The user can initiate the lock with just one or two keystrokes. If more keystrokes are necessary, many users will not take the time to lock the terminal.
- Once locked, the only way to unlock the terminal is to enter a valid password.
- No terminal input or output can occur while the terminal is locked.
- Terminal output can be buffered in memory and displayed on the terminal when the terminal is unlocked. If this is not done, users will be reluctant to invoke the lock for fear of missing an important message from the computer.
- The unlock password is encrypted using a one-way encryption algorithm.
- The unlock password can be the user's original log-in password, provided that it is properly encrypted.
- The same locking technique works for all terminals.

#### Automatic idle terminal locking

Once a locking mechanism is available, a program to detect idle terminals can lock those terminals rather than log them out. This pleases users since the worst thing that can happen to them is that they may have to enter the unlock password before continuing on. Also, since idle terminals are locked rather than logged out, the time between execution cycles for the program can be decreased substantially.

### Insufficient Access Controls

Many sites fail to use access controls sufficiently. Often access controls are available, but are improperly used or not used at all. This leaves a door wide-open, since users or would-be users can access information for which they were not authorized.

#### File protection

The first thing to do is to properly protect the files, memory structures, and devices on the system. On VAX/VMS this is done through file protections and access control lists [3]. A system manager or system security administrator needs to set up the default protections and access control lists so that the system will tend to stay in a secure state as new files are created.

#### Dial-in and network access controls

One of the ways that an outsider or ex-employee can access a system is through dial-in lines. Although VAX/VMS does provide an additional dial-in password, it is often useful to place additional access controls on dial-in lines so that such individuals do not even get an opportunity to log in. For example, using dial-back hardware or software can be helpful in assuring that a user is dialing in from a known location. It can also be effective in maintaining an audit trail of telephone numbers and users that have dialed in to the system.

Another troublesome area is controlling network access. In some environments there may even be systems on the network which are not under the direct control of the site. It is particularly important that the DECnet proxy logins be properly set up [3]. Otherwise, it may be possible for a user on one system to gain unauthorized access to another system.

#### Access control by function

On VAX/VMS, it is possible to control access by function rather than simply through file protections, access control lists and privileges. This is done by using captive accounts and other software. For example, an operator needs READALL privilege to backup all of the files on the system. However, most sites would prefer that the operator not be able to use this privilege to print any file on the system. In order to restrict the operator to just the backup function, a captive account must be used.

*A key principle in controlling access to the system is to have one user per account.* If multiple users are allowed to use the same account, then the site loses accountability for each of those users. There is no way for the site to tell which user is actually logged in to the account. Moreover,

password control on such an account is usually very poor since the multiple users continually share the password amongst themselves.

### Failure to Immediately Apply Updates

Often software (particularly operating system software) updates correct security problems. If an update does correct a security problem, the site should apply it immediately. If the site fails to do so, it runs a significantly higher security risk. This results from the fact that once the update becomes public, everyone knows that there is a security problem with a particular program and some enterprising individual could use the update to find the security hole and exploit it.

### Vulnerability to a Trojan Horse Attack

A Trojan horse is a seemingly innocent program which contains covert logic to perform some damaging action. An example of a Trojan horse is an enticing game program which also attempts to copy the salary file to an unsecured location. A Trojan horse can be particularly difficult to defend against and is an open door at most sites. Nevertheless, some methods that can be used to defend against such an attack are listed below [4]:

- Be cautious of free software—it may contain a Trojan horse.
- Acquire software from reputable sources.
- Use mandatory access controls [5,6].
- Check integrity of programs and files periodically [7].
- Use security monitoring to detect abnormal behavior.

### Lack of Security Monitoring

Many systems lack any type of monitoring for security purposes. This means that if a security problem occurs on such a system, there will be no record of the fact nor any evidence of who did it and how it was done. This leaves the door wide open for system abuse to occur, especially from the inside. On the other hand, when a system contains proper security monitoring it can provide the site with the following benefits:

- Deterrence—users are deterred from performing unauthorized activities.
- Evidence—information is collected which can be used as a basis for discipline or prosecution.
- Recovery—enough information is available to enable the site to recover from both accidental and deliberate damage or loss.
- Assurance—sufficient information is available to provide the assurance that a system is being used appropriately.
- Accountability—every user is accountable for his actions on the system.

The VMS audit trails and accounting logs [2] provide an excellent starting point for security monitoring. In addition, it is necessary to monitor terminal interaction in order to collect sufficient information to achieve the benefits listed above. Also, since a large amount of information is being collected, it certainly helps to have a facility which performs automated analysis and reporting from this data [8–13]. Such a facility can assist the site in pinpointing potential security problems without having to manually scan all of the monitored data.

### Conclusion

Since a company's information assets are generally stored on computer systems, it is essential that these systems be protected. In many cases security problems occur because of common oversights that leave doors open to the computer system. By following the guidelines in this paper, it is possible to close many of these doors.

### References


[1] Straub, Detmar. "Computer Abuse and Computer Security: Statistics from a Recent Empirical Study." *The 4th Insider Threat Identification Systems Seminar.* Bethesda, MD: August 25, 1987.

[2] Clyde, Robert. "Investigative Techniques." *The 5th Insider Threat Identification Systems Seminar.* Bethesda, MD: November 24, 1987.

[3] Digital Equipment Corporation. *Guide to VAX/VMS System Security.* Maynard, Massachusetts: July 1985.

[4] Clyde, Robert. "Defending Against Trojan Horses, Viruses, and Worms." *Proceedings of Digital Equipment Computer Users Society.* Nashville, TN: Spring 1987, pp. 381-386.

[5] U.S. Department of Defense. DoD Computer Security Center. *Department of Defense Trusted Computer System Evaluation Criteria*, CSC-STD-001-83 (Aug. 15, 1983).

[6] Blotcky, S., Lynch, K. and Lipner, S. "SE/VMS: Implementing Security in VAX/VMS." *Proceedings of the 9th National Computer Security Conference.* Gaithersburg, MD: September 1986, pp. 47–54.

[7] Young, C. "Taxonomy of Computer Virus Defense Mechanisms." *Proceedings of the 10th National Computer Security Conference.* Baltimore, MD: September 1987, pp. 220-225.

[8] Clyde, Allan. "Insider Threat Identification Systems." *Proceedings of the 10th National Computer Security Conference.* Baltimore, MD: September 1987: pp. 343-356

[9] Denning, Dorothy. "An Intrusion-Detection Model." *Proceedings of the 1986 IEEE Symposium on Security and Privacy.* April 1986, pp. 118–31.

[10] Peters, Bernard. "Automated Audit Trail Analysis." *The 4th Insider Threat Identification Systems Seminar.* Bethesda, MD: August 25, 1987.

[11] Peters, Bernard. "The Audit Trail Analysis Program." *The 5th Insider Threat Identification Systems Seminar.* Bethesda, MD: November 24, 1987.

[12] Gates, James. "Tools for Identifying the Source of Security Breaches." *The 3rd Insider Threat Identification Systems Seminar.* Bethesda, MD: April 2, 1987.

[13] Clyde, Robert. "Suspicious Event Testing and Weighted Scoring for the Analysis of a Surveillance Data Set." *The 3rd Insider Threat Identification Systems Seminar.* Bethesda, MD: April 2, 1987.

# Writers as User Interface Designers

Peter Donahue

Digital Equipment Corporation

Nashua, New Hampshire

## What This Paper is About

The importance of the software user interface is growing as less technical people use computer systems. To use these systems productively, the user interface must be thoughtfully designed. Writers, with their strong interpersonal skills and user knowledge, are in an excellent position to both help define the interface and facilitate the design process. This paper will discuss these points within the framework of GOLD, an actual DEC project.

By effective user interface design, I mean more that just coding for the terminal screen. Design is the long, arduous journey of bringing an idea through a large corporation. The user interface is everything a user sees.

This paper will discuss the writer's role in the design of an effective user interface. It's strictly from a writer's point of view and, like they say in the movies, any resemblance to real people is strictly coincidental!

It's exciting work for me and others in my group. Better user interfaces mean better products because they are perceived to be easier to use and they require less documentation. I hope to show that writers' work in user interface design will produce these results.

This paper will cover these main points:

Brief description of my group, Office Systems Documentation, to set the context of our present user interface work

Description of GOLD, a project where most of our ideas about windowed user interface design came from.

What we as writers discovered about ourselves and our craft and how that knowledge is being used on our current projects.

A look at how this has influenced what we're doing now.

There is a great deal of interest in user interface issues in DEC these days. I will talk about the GOLD project, one of the first to break ground in this area. Along the way, we discovered some things about how writers could use their skills to help the project along.

## A New View of Human Engineering

This quote by Ben Schneiderman sums up how we feel about the importance of user interface design.

*Human engineering, which was seen as the paint put on at the end of a project is now understood to be the steel frame on which the structure is built.*

User interface design was usually a last minute thing. In the past, this was acceptable. A good programmer could figure out the system on her own without relying on the user interface or documentation. But this has changed. Computers are no longer a way of life for most people, but a tool to get their work done.

## Office Systems Documentation (OSD)

Our user interface design work did not come out of the blue. The OSD group has always been concerned with user needs. Members of the group have worked on core documentation, on-line help, CBIs, and on-line documentation. From these ideas, the next logical step was for writers to get involved in the design of the user interface itself.

## What Was GOLD?

GOLD started in response to competition. Several competitors were selling compound document editors. GOLD was to be a compound document processing system that allowed a user to create documents with text in multiple fonts and point sizes, tables, and graphics. The types of graphics included charts, drawings, and scanned images.

It was designed to run on Digital's VAXmate and IBM AT bit-mapped workstations under MS Windows, an object-oriented interface that used a mouse. With the bit-mapped screen, you could see the document as it would appear in final form - proportionally spaced text in multiple fonts and page breaks. Its sophisticated formatter prepared the document for a variety of printers.

As you might assume, GOLD was a large project, requiring several large engineering groups to work together towards a common goal. The difficulties inherent in this arrangement were greatly underestimated and caused problems later on.

To provide a context for the rest of the discussion, here are GOLD's significant milestone dates:

December 85 - The functional spec committee formed. Its goal was to draw up the plans for what was to be built. Our writing supervisor got himself invited to join this committee.

January 86 - According to the old model (Schneiderman's "paint") the user interface was treated as a component (and a small one at that). Documentation, however, recognizing the importance and opportunity of doing primary user interface work, eagerly took responsibility for the user interface section.

February - The first draft of the function spec is published. Only the User Interface section has anything to do with MSwindows.

March - User Interface Design Group formed.

April/May - Second draft of the functional spec published with a very complicated User Interface section.

May/June - Formal, officially sanctioned User Interface Design Group falls into disarray. The writers begin meeting informally with key developers.

July - "Skunk works" begins.

October - Third draft of the functional spec. Contains a User Interface section produced and written by writers. This section becomes the first draft of a separate document.

January, 87 - Second draft of the writer's User Interface document. Project canceled due to a variety of reasons - chief of which was that resources were taken for the DECwindows project.

I'd like to focus on the March and July dates because they represent the most contrasting styles.

## March - User Interface Design Group formed

The User interface Design Group, with representatives from documentation, engineering, software architecture, human factors, and international groups, met three times a week for three hours in what quickly became a "black hole" of productivity. Very little was accomplished in a group that was too big and unfocused. Several times two people would argue a technical point by writing at different ends of a white board at the same time. The group talked at length about how the software should handle technical details like a chapter in a bulleted list while primary user interface questions went unanswered.

Were the right questions being asked by this group? Was it important in a user interface design meeting to discuss how that software would handle putting a chapter into a list? Writers felt the question should be "Does the user need it at all?" A user shouldn't have to worry about what the software does.

The problem of trying to merge large, separate development groups started to become evident. The graphics, text, and tables groups all had separate visions and ways of looking at a problem.

## July - The Skunk Works starts

The documentation 'skunk works' started in July. Skunk works is a term coined by Tom Peters, author of *In Search of Excellence*, to denote a small group of people left on their own to work on a project with little management influence.

To begin the process, writers began meeting informally with key engineers to talk about the user interface. They gathered additional information by getting the principal players into a room and facilitating discussion. Next, a writer wrote a description of one section of the user interface, meeting with an engineer to review the design and make adjustments if needed. The finished section of the user interface was included in the functional spec. The process worked well and it was formalized for other user interface sections.

## What Happened in the Skunk Works?

Some interesting things came out of the skunk works:

Power vs usability - writers tried to focus the design on the 20/80 rule which states that users use 20% of the product 80% of the time. Engineers wanted features, no matter how elaborate or how infrequently the user might need them.

Engineers are highly trained and technical people building a product for novice or intermittent users. Writers were invaluable here in representing the user and keeping the design on track.

An event near the end of the project brought our entire skunk works process into focus. I was in a conference room with another writer and a few engineers talking about how the user should create tables, a difficult problem. We discussed and rejected several designs, until it looked like there might be no solution. Finally one engineer turned to us and said, "Well, how do you want it to look? Tell us and we'll code it that way." This represented the perfect meshing of writer and engineer.

## What Was Our Goal?

Our ultimate aim was to:

Develop a basic design

Prototype it

Let people use the product

Use their feedback and comments as strategies for the next design cycle

Refine the design and sent it back into the loop

Writers did some prototyping by getting software tools from enthusiastic engineers through non-official channels.

## What Worked in GOLD?

Clearly, we made the most progress in the skunk works. With documentation producing sections of the user interface, we provided a focus for design and discussion. The user interface design was often the only representation of the product. Many people said they could only understand the product after reading our User Interface sections.

## What Didn't Work?

Time was the big enemy of this project in two major ways. First, a great deal of time was spent in the large User Interface Group that could not be made up at the end of the cycle. Second, the design of a user interface is an iterative process - you can't build a user interface right the first time. More time should have been spent in this step.

Writers are not user interface experts. We have good instincts, and provided good input, but user interface design is a full time and expensive job and needs to be handled by experts.

Resources were scarce. Even members of our own group warned against taking on too much and reminded us that writing manuals were our primary responsibility.

Finally, and most importantly, the audience definition kept changing from the casual user to the desktop publisher to the casual desktop publisher. (Writers kept to the managerial, intermittent user.)

## Why Do Writers Make Good User Interface Designers?

Writers intercommunicate. Often the only time different groups got together was at a meeting we called. We were the glue that held the project together. We found that the arrangement of people was the most important element.

Writers were facilitators. Once the groups were brought together, we acted as the hub of a wheel for the different groups and helped them talk together. Writers come from diverse backgrounds - teaching, school administration, journalism, social work - and are trained to see things in a larger perspective - globally rather than analytically. As generalists, we complement the technical focus of engineers. Strength of intellect alone does not convince people of a point of view - communication does.

Writers also keep up with what users want through regular visits and phone conversations and translate this knowledge into useful information.

Writers don't suffer from the Not Invented Here syndrome. The first thing we did on GOLD was to go to tradeshows to see what competitors were doing. We recognized good design and incorporated the best into our own work. We also adopted MSwindows and its Style Guide immediately, a move that ran counter to accepted thinking at the time.

Finally, writers use editors in our work. We know what we need and like in these tools.

## How Has This Influenced Current Products?

There is now a corporate emphasis on windowed user interface design with the DECwindows project. Human factors groups, who used to test designs written by others, now do primary design. They are the experts the writers were not.

Writers now have clear roles in the user interface design process in several ways: as expert users; investigators of competitors designs; as watchdogs of the design. Designing is a full-time job and writers need primarily to write books. The writer, engineer, and user interface designer make the best team.

Writers have grown and now think visually as a way to reduce documentation. Why describe a fill pattern, for example, when a user can see it and simply click on it?

We used to say that the books were big because the software was complicated. Now that the user interface is better, we do not have this excuse. We have to "put up or shut up" and reduce the size of the documentation. A good user interface design is inversely proportional to the size of the documentation.

## Conclusion

In this paper, I touched on four main points:

Office Systems Documentation has always been concerned with user concerns and user interface design was its next logical step.

GOLD gave us the chance to put our ideas into action, and took us from the megacommittee to the skunk works.

Writers grew in understanding of their craft and carried that understanding into new projects.

Current and future projects have been influenced by these ideas.

Why should you, as a user, care that writers help design the user interface or that they have grown? Because as a user you will be saying, "This product is great, it's so easy to use." And you must also say, "These books are great, they are easy to use and they're so short." If you don't say this, we've failed as writers.

So, my message is not only to writers and engineers, but to anyone who can influence a user interface. Get involved. It's worth it.

###

# PERSONAL COMPUTER SIG

# Trojan Horse Software

Kenneth A. Stricker
Martin Marietta Electronic and Missile Group
Orlando, Florida

Abstract

This paper addresses a serious problem encountered by users of public domain software on personal computers. Although most public domain software is entertaining or useful, there exists a small group of programs (usually referred to as Trojan Horse Programs, or Trojans since they get into a system under the guise of useful software) that have been written specifically to either irritate the user, or to physically destroy any data stored on the user's computer.

As the number of computer bulletin boards, (where most public domain software is traded) increases, users must be aware of the potential dangers of Trojan Horse Software. Users of PC programs brought into business locations, engineering offices, or manufacturing plants may place local PC's at risk. In fact the use of some Trojans could be construed as an act of sabotage which, when traced to the perpetrators, could lead to legal action, and depending on the installation affected, lead to a fine or imprisonment.

## Why Public Domain Software ?

The main reason that public domain (PD) software is widely used is that it is usually very inexpensive (NOT free), it is easily and quickly obtainable through computer bulletin board systems, and public domain software clearing houses, and it often is targeted toward more specific applications than commercially available software. It is generally written by a computer user, or group of users, to solve a specific problem, or to satisfy a specific need.

The main method of distribution for PC based public domain software is through computer bulletin boards. A bulletin board system allows a PC user to dial in to a "host" PC system via a modem, and leave electronic mail for other users, participate in interactive conversations with other users, post technical or special interest questions to other users, and to upload and download public domain software.

Most bulletin board systems (BBS's) have some form of security, ranging from simple fill in the blank logon questions, to elaborate callback schemes. One such callback scheme (used on at least one BBS that I know of in Orlando,FL) requires a first time user to leave his or her phone number and immediately hang up. The "Host" computer then calls back the user, and produces three sets of one to nine separated by a pause. The user must count the number of tones in each group, and this becomes his three digit logon code, which in conjunction with his username and password allows him access to the BBS.

## What is Trojan Horse Software

A broad definition for a Trojan horse software package (Trojan) is any software program or utility that intentionally produces unwanted, unpleasant, or unexpected results. The key word in this definition is intentionally. Many programs, PD and commercial, will on occasion produce unpleasant results, either due to a "bug" in the code, or misuse by the user. Trojans, on the other hand, are written specifically for this reason.

The problem with Trojans is that they are easy to obtain, and difficult to trace. Your friends can be your worst enemy, as many people will upload a newly acquired software package to a BBS without first running the program. Fortunately many (most?) BBS System Operators (SYSOPS) test all uploaded programs before they are made available for download, or at least tag newly uploaded and untested programs as untested. Even so an occasional Trojan may slip through undetected. especially difficult to detect are Trojans of the mole, time-bomb or virus variety which will be discussed later.

The consequences of a Trojan may vary from minor annoyance at the loss of some data, to the actual destruction of property. If you consider what might happen if Trojan were to "Crash" a hospital computer, or a railway control system, the results could be fatal.

There are basically four types of Trojans: The type that simply annoy the user, but do not directly produce any irreversible side effects (Level 1 Annoy Trojans), those that may or may not cause the loss of some data (Level 2 Annoy Trojans), those that are intended primarily to destroy or corrupt data and storage, and those that affect system user files. Each type will be discussed separately.

### Level 1 "Annoy" Trojans

The level 1 annoy Trojans are generally "prank" type programs that are intended to be run on someone else's system. These Trojans are for the most part harmless (unless the person finds out that you did it), and usually humorous. They can only marginally be referred to as Trojans, and are only included in this paper since they do fall under the broad definition since they produce unexpected results for the PC user. These programs may run in the background, or may present a "fake" prompt screen that waits for input to "strike".

One of the best known programs of this type is DRAIN, which initially presents the unsuspecting "victim" with what appears to be a normal DOS prompt screen, however as soon as he enters a command, he is presented with a formidable looking error message that informs him that his disk drive is full of water. The program then proceeds to inform him that it is draining the drive (accompanied by appropriate sound effects) and then putting it through a "spin dry" cycle (also with sound effects), and then returns the user to DOS.

Another level 1 program (and one of the author's favorites I might add) is called DRIP. Drip runs in the background (i.e. the user is free to run programs, issue commands etc. as if it weren't there) and is inactive until the "Scroll Lock" key is pressed. DRIP then randomly looks at locations on the screen, and if there is a character in that location, it scrolls it down the screen until it either hits another character below it, or "drops off" of the bottom of the screen. this gives the appearance that the characters are "Dripping" off of the screen. This program does not seem to affect the operation of the PC (Except in BASIC).

### Level 2 "Annoy" Trojans

The chief difference between Level 2 Trojans and the Malicious Destroy type Trojans is that level 2 Trojans do not affect physical storage. This is not to say that no data is lost from level 2 Trojans.

Level 2 Trojans may take several forms, and some may only be considered Trojans under specific circumstances. One such program is JULIAN, a program designed to allow a user to "lock" the computer keyboard until he reenters a password. This is a useful program, until someone comes along and locks your keyboard for you.

Other level 2 programs may erase all or part of the system memory, either immediately or from a "background" mode. In background mode the program can be loaded into memory, certain interrupt vectors changed to point at the program (such as timer tick, or keyboard interrupts) and a "Terminate but stay resident" command used to end the program (this is the same method used in memory resident utilities as Sidekick). At some occurrence of the specified interrupt, the program takes control of the system and may either "Hang" the system, re-boot it, or "Crash" it. This could be a disaster if you are in the middle of creating a 200x400 cell spreadsheet. This type of approach is often referred to as a "Time Bomb" program, and may also be used by the next type of Trojan.

### Malicious "Destroy" Trojans

Of all the types of Trojans, the Destroy Trojans are by far the most dangerous, and are therefore the main focus of this paper. These Trojans usually attack the PC's disk storage system in one way or another. They may delete individual files at random, or go for specific files such as the DOS COMMAND.COM file, or they can reformat the disk or render the data on it unrecoverable. If the computer is has only floppy disks, this is a minor problem as only one floppy is lost. However the loss of the data on a 10-40 Megabyte hard disk could mean total disaster.

Trojans that delete individual files are generally the easiest to recover from if they are discovered in time. Since deleting a file generally does not erase any file data, deleted files may often be recovered using file utilities such as "The Norton Utilities". However the user may not immediately notice the loss of something like a quarterly report file until after other files have been stored over the data.

Reformatting destroy Trojans do exactly what the name implies, they perform a total reformat of the disk. Once reformatted, any data that was previously stored on the disk is lost, as all sectors of the disk are written over in the format process.

The previous 2 types of Trojans are relatively easy to understand, since they basically emulate existing DOS Functions. However in order to understand how the rest (and majority) of the destroy Trojans work, I need to explain some basics of the DOS disk filing system.

The first sector on any DOS disk, either floppy or fixed, is referred to as the "Boot sector" e figure 1). This sector contains data about the disk manufacturer, the sector size, the number of sectors per block, the type of disk, and a "bootstrap" routine.

Following this area is a structure known as the File Allocation Table, or simply the FAT. This area is used map directory entries to the corresponding area of storage on the disk (Figure 2). Each FAT entry corresponds to one cluster (one or more sectors) so that the data for a file directory entry that points to the 2nd FAT entry would be in the 2nd cluster on the disk. Each FAT entry in turn contains a 3 Byte value that indicates the next FAT entry/cluster used by the file. In this manner, a file may be allocated as many clusters as necessary with the last FAT entry in this "Chain" containing the hexadecimal value FFF. Unallocated clusters are represented by a value of 000, which tells DOS that that cluster is available for use. (Please note that FAT entries 0 and 1 are always reserved by DOS.) Other reserved FAT entry values include FF0-FF6 for a reserved cluster, and FF7 for a bad cluster. There may be 1 or more copies of the FAT stored on the disk (for backup purposes), as defined in the Reserved Area.

Following the copies of the FAT is a fixed size area for the Root directory entries, followed by the User files, and sub-directories (Figure 3).

Many of the destroy type Trojans attack the FAT. This is usually done by writing FFF's (end of chain), or 000's (cluster available) to all entries of the FAT. As you can see, this no longer allows DOS to access all of your data. Another popular tactic is to write over the reserved area of the disk, which renders the entire disk unusable. Still another tactic is to write Bad Clusters (FF7) to the disk.

One of the hardest destroy type programs to detect is referred to as either a Worm, or a Mole. These programs do not attack whole files or disks, but instead are content to destroy small parts of your file. The terms Mole and Worm refer to the fact that these programs dig their way through the data on the disk a little at a time. They are not usually discovered until they have destroyed major portions of many files, and often not until after several backups have been made of the affected data.

Another relatively new (and thankfully rare) problem is referred to as a computer Virus. The virus program, much as its name implies, attaches itself to a host computer, and proceeds to replicate itself wherever possible (often within other legitimate software). It can easily be spread to other disks and computers in program or data files. Usually a virus "attack" is triggered by some external factor such as system date.

BBS User File Copiers

This type of Trojan should be of concern to any BBS SYSOP. This program generally copies the contents of the BBS user file, which contains all of the user names and passwords, into a simple ASCII text file that the person who planted the Trojan can then download. This allows the perpetrator to log on to the BBS using any user name, particularly the SYSOP's id. Once logged on with SYSOP privilege, the user can manipulate many aspects of the BBS, crash it, delete files, or give everyone SYSOP priority.

What Can I Do?

There are several programs out in the public domain designed to detect Trojans, and to prevent them from doing any damage. Programs such as BOMBSQAD run in the background and intercept all calls to the system for disk access (read, write, format or verify), tell the user the nature of the call, and ask the user if it should process the call. Other programs such as CHK4BOMB call. Other programs such as CHK4BOMB scan a suspect program's code for all ASCII strings (strings such as "Got You", are quite popular), and format or disk write commands without running the suspect program.

Your best defense against Trojans is probably your local BBS SYSOP. Most SYSOP's test any newly uploaded programs before they are made available to download. If you suspect that a program that you are running is a Trojan, please notify the SYSOP of the BBS that you got it from.

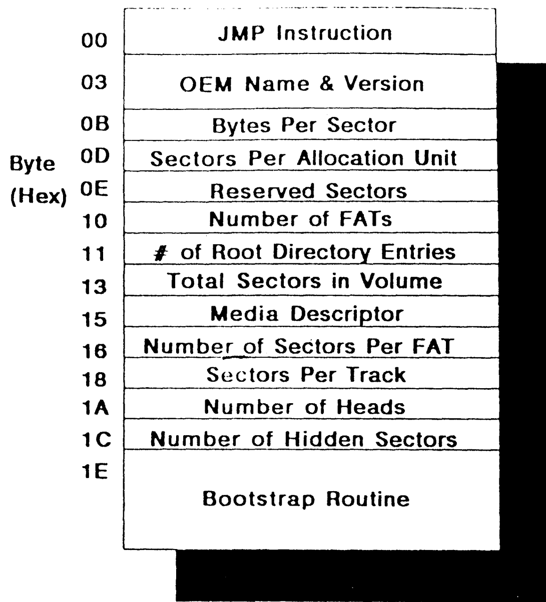



Figure 1 – MS-DOS Disk Reserved Area

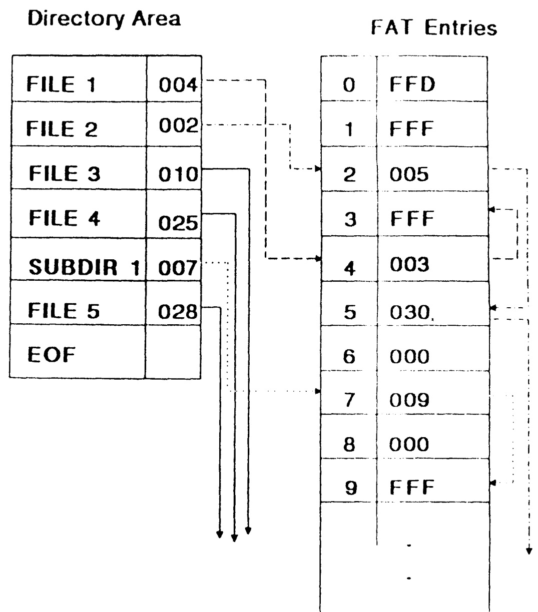



Figure 2 – FAT Utilization

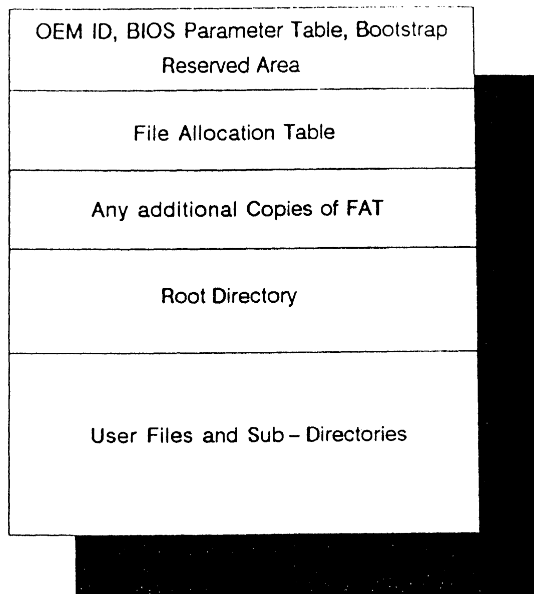



Figure 3 – Typical MS-DOS Disk Configuration

# PC Workstation to VAX Connections for Maximizing Resource Flexibility

Robert Gary Mauler
and
Valerie Cabral Mauler

Westinghouse Electric Corporation
Baltimore, Maryland

**ABSTRACT**

**Enhancement of Productivity is a major goal in connecting Personal Computer Workstations to VAXs. This paper defines a solution for maximizing resource availability and flexibility in a large computer environment. Connecting Personal Computer Workstations to VAXs makes available a wide variety of network resources from specialized software and output devices to alternative computers such as UNIVAC and CRAY. Issues addressed will include high and low end connectivity needs and software selection.**

An observation was made in 1983 that having personal computers can make individuals in our organizations more productive. Productive individuals, however, don't solve most problems by working alone. They consult with colleagues; they exchange information; they build on one another's ideas. Networking personal computers enables "PC" users to share information, ideas, and resources thereby working efficiently as a team to solve problems. The goal of this paper is to share lessons learned over the last five years in connecting personal computers and their peripherals over Ethernet to a large network of DEC VAX mini-computers.

A book or even a volume of books could be written about the process of connecting PC workstations to a VAX network. The number of possible and workable solutions is almost endless. The trick is that some combinations just happen to work better than others. The two areas that this paper addresses are connectivity and resource sharing.

Connectivity is the glue that is used to link personal computers together to form a network. We refer to networked personal computers as workstations. The goal is for the user to be able to access any and all of his computing resources from one multi-function workstation linked by one cable to the rest of the world.

At the Westinghouse Defense Center in Baltimore we chose a "DEC" based solution mainly because our engineering departments are largely VAX users. As one would expect, other Westinghouse sites that are IBM based naturally migrate to IBM type solutions. But connectivity is not lost since we do provide for gateways out of our local area network (LAN). Industry standards and multi-vendor support were also an important consideration since our site tends to have "one of everything" and they all need to talk together. Even though our MAC users used to swear their Apples only needed to talk to LaserWriters, it was not long before they wanted to talk to a VAX, an IBM, or whatever!

To meet our connectivity needs, Ethernet was selected as the primary networking scheme for our LAN. Digital's Network Architecture (DNA) or DECnet was selected as the network protocol to use over Ethernet.

Even though DECnet is not an ISO standard, the fact is that there are a whole lot of machines out there today that know how to speak DECnet, and more are coming as DECnet is ported to various machines. Technology Concepts Inc. in Massachusetts is building a business around porting DECnet to various vendors' machines that have "C" compilers available. Westinghouse's R & D Center in Pittsburgh has successfully implemented its own version of DECnet on a UNIVAC system.

In 1983 when we first started thinking about workstations, DECnet was not available for the IBM PC. So our first venture into networking was with VIM from SYNTAX in Seattle, Washington. The VIM product was based on 3COM's EtherShare networking software. This software worked very well and still does today, but it really is its own little network. The problem is that even though the PC files were stored on a VAX, VMS did not know what to do with them since they were not RMS type files. DECnet routers had similar types of problems since EtherShare talked XEROX's XNS and not the familiar DECnet protocol. To solve these limitations we chose to standardize on VMS Services for MS-DOS from Digital. Even though this product will not win a speed race with 3COM, NOVELL, or VIM, the functionality "makes the wait worthwhile". VMS Services was also attractive to us because it embraces industry standards such as MS-WINDOWS and MS-NET from MICROSOFT. The icing on the cake is the fact that VMS Services also includes the DEC Local Area Transport (LAT) protocol thereby giving workstations the ability to connect to VAX's directly as VT type terminals and even to non-DEC hosts by using DEC Terminal Servers (DECserver200 for example). This feature eliminated the need for us to expand our data switch system.

Once you have connected the standalone PC into a network environment, it is possible to share the many resources available on any of the servers and hosts connected. The possibilities are almost endless but typically at Westinghouse high performance and high quality (and naturally high cost) peripherals are connected to servers and distributed conveniently around our offices. We feel that this approach makes more sense than purchasing noisy, low performance, low quality peripherals and connecting them to each PC. Another benefit is that maintenance, service manpower, and cost for the shared peripherals can be absorbed into the operation budgets of the VAX and other Main-Frame servers. Another benefit which is sometimes overlooked on PC based networked systems such as 3COM and NOVELL is the ability to almost transparently use any of the software tools available on the network. For example, a programmer could edit his source code on a PC and then submit it for compilations on a large VAX or even a CRAY. This approach takes advantage of the fact that the PC does very well with interactive jobs such as "mouse" driven free hand graphics programs and other highly interactive programs, while you get the most "bang for the buck" when you let the VAX CPU run compute intensive applications. The key point here is to distribute your computing load across many processors to maximize your utilization of available resources.

For most sites a networking project starts out very small, as maybe only a few Workstations and one server as pictured in Figure 1. But in most cases it grows very quickly and usually larger and sooner than the management ever dreamed it would. Our choice for a network server was a DEC MicroVAX II. The MicroVAX II was chosen for several reasons. First, we expected a large number of Workstations (40-60) on the network and therefore needed a CPU that had enough horsepower to handle not only the file and print server functions but also to support network functions such as multiple physical network connections (Ethernet, X.25, Async, and Sync), DECnet Routing, and Gateways to other systems such as IBM/SNA, PROFS, and EMAIL systems. Secondly, with 40 to 60 Workstations at about 3 to 4 users per Workstation we wanted to have a large amount of disk storage available. The microVAX II can easily handle two RA81 disk drives which give us about 900 megabytes of disk space. Finally, faced with backing up 900 megabytes of data, we definitely wanted to have a high density and high speed tape drive. Our choice for a tape drive was DEC's TU81 6250 drive.

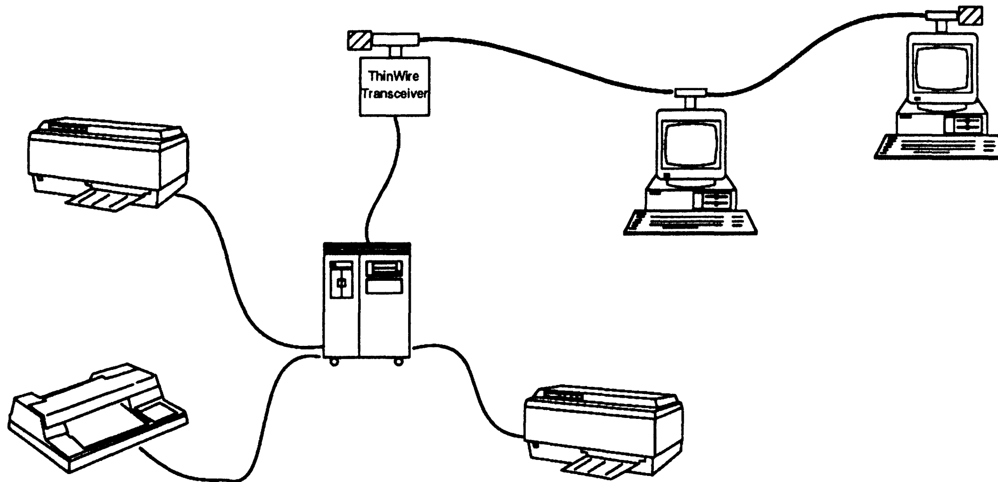



Figure 1

Even though it is possible to use a personal computer as a server, and in some cases it is a good way to start, for us it just would not "cut the cake," so to speak. There are several drawbacks to a PC server which must be considered. First, the standard out-of-the-box PC's tend to be shipped with slower access time hard disks in order to keep the unit cost down. That is something you definitely don't want when you are trying to share a disk with many workstations. Secondly, with the current architecture of the AT class of PC's, you quickly run out of physical ports as well as software interrupts to which you can connect your peripheral devices. If you need to connect less than 4 peripherals, this won't be a problem for you, but at our site we needed to connect about 8 serial laser printers and 3 plotters to one server. A third serious limitation with the PC class of server is the PC DOS operating system itself (OS/2 may change this). PC DOS works well as a single user system, but falls short when compared to a multi-user, multi-process operating system like VAX/VMS. As an example, our Talaris T800 laser printers are controlled by a process called a Font Manager. The Font Manager manages the down-loading of character fonts as well as forms overlays to the printer. The whole process is totally transparent to the workstation user. A similar type of software also exists, courtesy of the DECUS SIG Tapes, for PostScript printers. In this case the PostScript symbiont translates ASCII text files into PostScript programs and queues them for printing on our PostScript printers. In both of these examples the control software process must run concurrently with users as well as the server process. This works fine under a multi-tasking operating system like VAX/VMS but will not work in a single tasking environment like MS-DOS.

In Figure 2 the basic LAN configuration that was presented in Figure 1 has been expanded. The configuration shown is used in a situation where the network server is either too far for RS232 cables to reach your peripherals or where the VAX server does not have the physical room for an 8 to 16 line asynchronous interface card and connectors in its cabinet. Such could be the case if a MicroVAX 2000 was used. Digital's DECserver 200 allows us to use the already installed Ethernet backbone to connect 8 asynchronous devices per DECserver 200 to a VAX. Another benefit that the DECserver provides is that modems or other non-DEC hosts can be

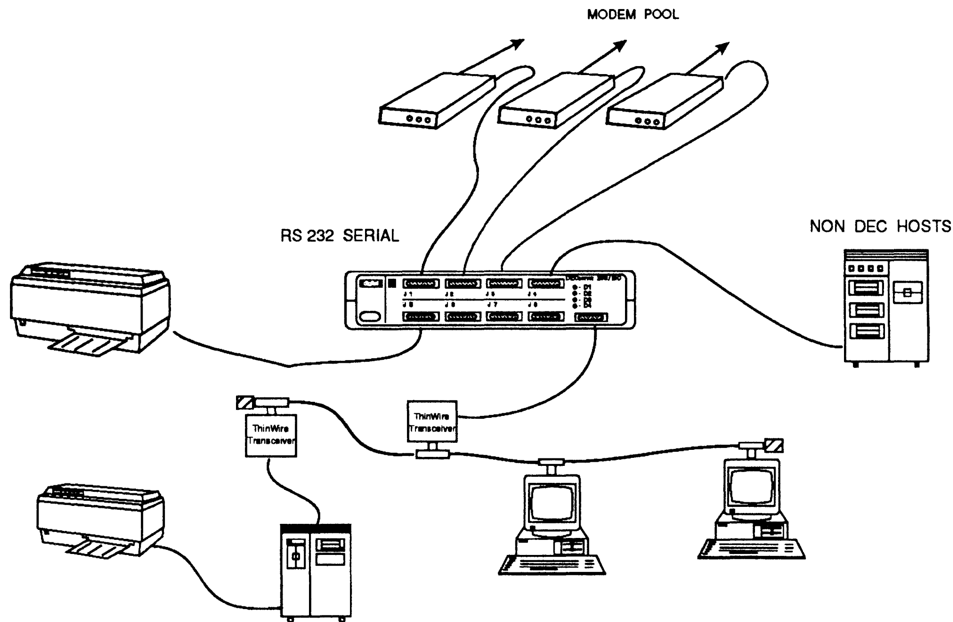



Figure 2

connected to it. As a result, the workstations use a VT220 terminal emulation and Digital's LAT protocol to connect to "services" on the DECserver such as a modem bank. This eliminates the need for modems and data lines for each PC since now they can be shared over the network.

For small companies with only one office the basic LAN we described earlier is all that is needed. But if your organization is of the national or international class, Wide Area Networking plans (see Figure 3) should be incorporated into the Local Area Networks throughout the organization. A detailed discussion is well beyond the scope of this paper so we will only point out the solutions we are using now or planning to incorporate into our network design. For the DECnet and Ethernet/802.3 worlds, devices such as DECnet routers and LAN bridges are certainly the first choices. The DECnet routing function can either be handled by a standalone box called a Router Server or it can be performed on one of the VAX/VMS servers equipped with full function DECnet routing software. The trade-off here is in cost vs performance and in network topology. For example, if there is a lot of DECnet traffic to route, it probably makes more sense to purchase a DECnet Router Server box than to load down an already heavily loaded VAX CPU when it could be doing useful things like compiling programs. The drawback with routers is throughput. There are basically two areas where problems can arise. First, the CPU performing the routing function just might run out of horsepower. Secondly, the physical links tend to be 56 Kbaud or less. If throughput might be a problem in your system, you can replace the router with a device called a bridge. The bridge works at only level 2 of the network model. Therefore it only needs to know about Ethernet/802.3 packet source and destination addresses. Since it looks only at packet addresses and not at higher level protocols it will not only forward DECnet packets but also XNS, TCP/IP, and other protocols. This is a big advantage in a mixed vendor environment such as ours. Also, because of the reduced processing load on the CPU due to only processing level 2 protocol, bridges such as DEC's LAN Bridge 100 can forward packets at the full Ethernet bandwidth. The physical links available for bridges range from RS-232 data speeds up to 10 MBits over fiber optic cable or microwave.

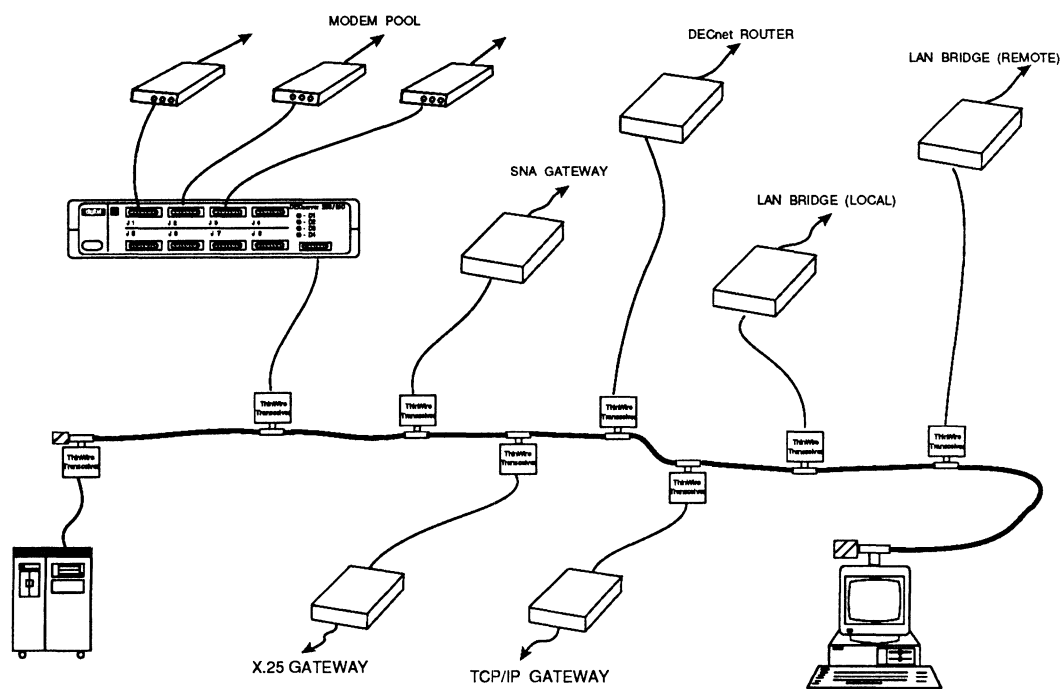



Figure 3

A gateway for your WAN that is worth looking at, especially for large organizations, is the X.25 packet switched network. More and more large companies such as Westinghouse are putting in their own private X.25 networks with gateways into the public packet networks. The X.25 network provides a cost effective way to link local LAN's together to form a company wide WAN. Although they are a far cry from Ethernet in terms of speed, they are cost effective if your data traffic is medium and not of a highly interactive nature. For high speed WAN's the choice should probably be bridges linked with T1 (1.544 MBit line).

It almost goes without saying for large corporations that there should be some type of link to the IBM SNA world. Here again the SNA gateway function can either be performed in a dedicated SNA Gateway box or hosted on a VAX/VMS system along with everything else. Issues that need to be considered when choosing an SNA gateway are: throughput (packets per second), number of user sessions supported concurrently, compatibility with the IBM application software of interest, and naturally, cost.

Networking your workstations together is really only half the problem. The real challenge lies in choosing application software that works together with the network system to give users a consistent view of computing tools in a distributed processing environment. Selecting a standard set of "core" software applications that works for an organization is essential to the support group's sanity as well as the profitability and competitiveness of the organization. For example, allowing every employee to choose his own word processing package might seem like a friendly thing to do (We know of one engineer who even wrote his own in BASIC so he did not have to buy one). But think of the problems involved when a team of people are given the task of writing a document. Each one of them could be using a different word processor or editor. When it comes time to integrate individual sections to form the final document it could turn into a nightmare. Simply put, there must be a grand plan for the way data and information flows through an organization. Either you pay up front for good consulting and design of a system or you pay and keep paying all

of the companies out there selling those translation programs and conversion services. Our core software list includes only about 10 packages out of the tens of thousands available for the PC Workstation. Those users that have an application that is not already covered by the "core list" are responsible for their own integration with time available consulting from our support group.

At Westinghouse we started the process of integrating our computing environment before too many bad habits were developed. Anyone in the defense business is well aware of the large amount of documentation required on government contracts. Westinghouse's Defense Center standardized on the WANG OIS word processor for secretarial personnel, MASS-11 on VAX's and PC's for engineering professionals, and Multi-Mate on PC's for management and business employees. A newcomer to the scene is the Interleaf TPS technical publishing system which is used for high quality reports and proposal generation. The Interleaf TPS package is hosted at Westinghouse on both the VAX and SUN platforms.

Our mixture of word processors provided a real challenge for software application integration. A diagram of our data flow is show in Figure 4. Starting on the left, text enters the system in one of three methods: (1) a secretary types in text from hand written copy; (2) an engineer composes text using MASS-11 on either a Workstation or terminal; or (3) other professionals enter text using Multi-Mate on a Workstation. In some cases text is also entered directly into the Interleaf system. Memos and business letters are created for the most part on WANG and Multi-Mate. Similarly, internal engineering reports, presentations, and some documentation are done totally in MASS-11. In the case where the text needs to be combined with scanned photographs and/or scanned artwork, an Interleaf Technical Publishing System (TPS) is used to do the page make-up. This process starts with text keyed into either WANG, Multi-Mate, or MASS-11, transferred through the appropriate communication link, and translated into Interleaf's document format.

The big problem we ran into when we wanted to integrate the WANG OIS system was that it does not know what the words

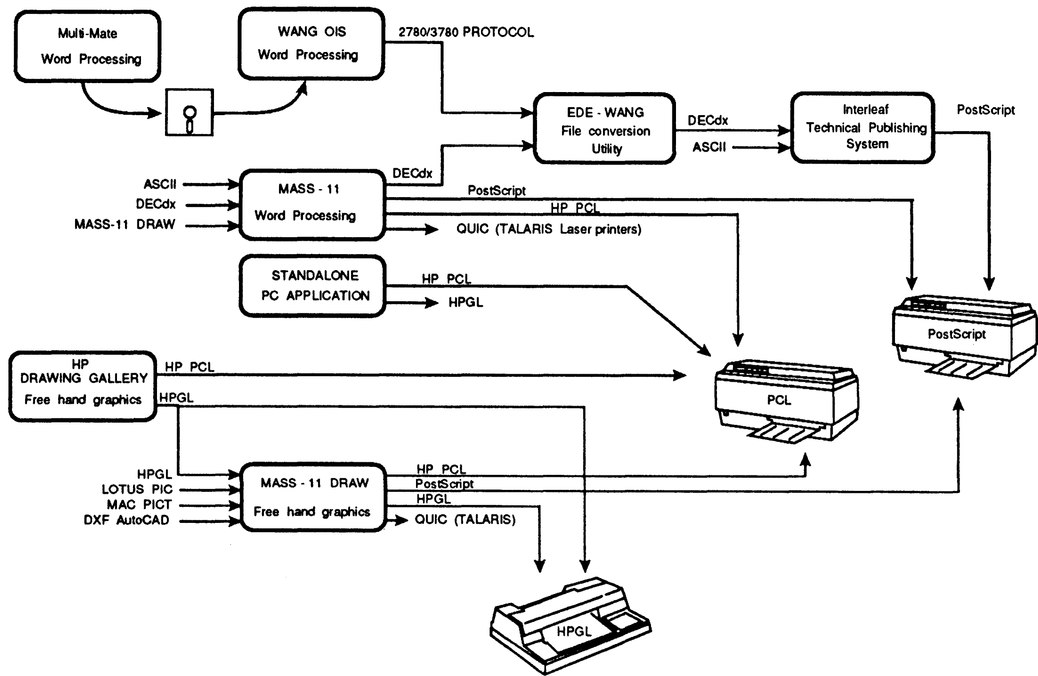



Figure 4

"integration" and "networking" mean. Thanks to a layered product from Digital called EDE-W (Electronic Document Exchange - WANG), documents created on our WANG OIS system can be easily transferred to and from a VAX while retaining document text format code information intact. The whole process uses the standard WANG telecommunication software. This means that except for adding two lines to the WANG document everything is "as usual" for the WANG user. The advantage here is that we did not have to add a new piece of equipment such as a "disk/file conversion box" which would have required training and required the user to leave his workstation to manually load and unload the floppy disks to be converted. EDE-W works by receiving WANG EBCDIC code over a 2780/3780 protocol communication link and then storing the document in an intermediate form called DECdx. Once the document is in DECdx form, MASS-11 and Interleaf can import the document into their respective systems, retaining the original text format. Although a document can be transferred back to WANG from MASS-11, this is not usually desirable since MASS-11 can provide enhancements to the document such as integration of graphics, multiple fonts, and varying point sizes which have no equivalent on the WANG system. The same is true for Interleaf, only more so, since Interleaf can include scanned photographs in a document. In the future we see the possibility of using the Interleaf Editing Workstation package and their ViewStation software which runs on a PC based Workstation. These two tools from Interleaf will enable us to edit and review text in Interleaf's format on the PC Workstation.

Since good old fashioned terminals are more plentiful than PC Workstations, MASS-11 and MASS-11 DRAW are used to integrate text and graphics from both the PC world and the VAX terminal world. MASS-11 word processing will run and, more importantly, looks the same to the user on both a PC and a VAX. With the addition of MASS-11 DRAW executing on the PC Workstation and MASS-11 Graphics Processor executing on a VAX, it is possible to integrate graphics produced by almost any application. The key to this is Hewlett Packard's Graphics Language (HPGL). There is hardly a package out there that does not support HPGL (HP Plotters) as standard output. The MASS-11 folks have a filter that can translate HPGL into their DRAW system where the drawing can be enhanced and then integrated with text from MASS-11 word processing resulting in a composite document output. MASS-11 DRAW also accepts inputs directly form LOTUS PIC, MAC PICT, and AutoCAD DXF, all of which are our standard PC Workstation applications.

In summary I would like to emphasize the importance of selecting a connectivity solution based on industry standards, such as Ethernet. Where there are standards there are multiple vendors competing to lower the cost to you. Easy expansion when new workstations need to be added is an absolute must and the network should not have to be taken down to do it. You should be interested in a network technology that will be around for more than a few years, and network protocols that work today and provide a committed migration/growth path to future enhancements and industry standards compliance. The software vendors that are selected to provide the core software applications should be people you get to know and can work with on a regular basis. Stay away from the traveling salesman type. They won't be there when you need a bug fixed or an improvement made to their software. Software applications should support your "standard" peripherals. You definitely need your organization's support to enforce the standards and policies on use and procedures that you will be setting up. Having networked our personal computers and standardized on a software environment, we have eliminated time wasted in moving and adapting information to different systems around our organization, giving individuals more time to use that information productively.

# RSX-11 SIG

# MODIFYING FMS-11 TO PROVIDE READ-WITH-TIMEOUT AND VIDEO ATTRIBUTE CONTROL

**Joseph E. Kulaga**
Argonne National Laboratory
Chemical Technology Division
Argonne, Illinois

### Abstract

FMS-11/RSX V2.0 is a forms-oriented video I/O management system that is well suited for use in traditional data entry applications. A recent attempt to use FMS-11 in a real-time process control application exposed several deficiencies. This paper describes modifications and additions to FMS-11 required by the application, such as video attribute control, "read-with-timeout," and programmable default menu responses. Some discussion is presented on the techniques used to approach this type of problem.

## Background

The Fossil Energy Users Laboratory (FEUL) at Argonne National Laboratory is a Department of Energy facility constructed to accommodate a wide range of experiments related to coal and oil combustion. The facility has been used to acquire data for the design and operation of a magneto-hydrodynamic power station, to study the control of $NO_x$, $SO_x$, and particulate emissions, and to verify instrumentation and heat transfer computer codes. The oil combustor portion has been in operation since 1980, and the coal combustor portion since 1985.

A decision was made recently to upgrade both the hardware and software used in the facility's data acquisition system, replacing an LSI 11/23 running an RT-11 based software system by a MicroPDP-11/73 running RSX-11M+. The software system requirements included the capability of periodic accumulation of data of from 100 to 300 sensors, periodic display of the data in real time, logging of all data to disk, and printing subsets of the data periodically. Most of the off-line data analysis was to be accomplished on a VAX.

In designing the method by which a user interacts with an on-line data acquisition system such as we are discussing, one generally must decide between a command language based system and a menu driven system. The command language based system has the potential of supplying a wider range of interaction capabilities, but generally requires a more sophisticated user, i.e., one more aware of the computer environment and the details of the software system and its components. The menu driven approach, however, provides a much more easily mastered interaction mechanism. With this in mind, FMS-11 was chosen as the input/output mechanism for all normal user interaction.

[1] The submitted manuscript has been authored by a contractor of the U.S. Government under contract No. W-31-109-ENG-38. Accordingly, the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes

## Hardware

Both coal and oil combustor systems accumulate data from a range of sensor types: thermocouples, pressure transducers, air flow transducers, weight scales, valve positioners, and gas analytical instruments. These sensors are multiplexed into an analog to digital converter and read by the data acquisition task. The multiplexers and ADC are part of a CAMAC-based[1] data acquisition system interfaced to the host MicroPDP-11/73 Q-bus by means of a CAMAC crate controller. An RSX-11M/M+ driver is used to access the CAMAC equipment via the standard QIO mechanism.

## Software

### Overview

Much of the user interaction is in the form of a typical menu based system. The user selects options from a main menu that in turn brings up various forms. One form is used to supply sensor addressing information to the data aquisition program's database. A second form is used to supply data conversion / calibration coefficients for the fifth order polynomial expression that is used to convert

[1] CAMAC is an internationally accepted interface standard widely used in data acquisition and process control environments. See the IEEE-583 document.

the raw data to engineering units. (The system supports up to 64 different coefficient sets which may be assigned to the various sensors.) The option to start data acquisition brings up another form which defines run-time variables for the system such as the length of the experiment, the data logging rate, the data file name, the screen refresh rate, and several others.

When data acquisition begins, a number of independent tasks begin to execute in traditional RSX fashion. An acquisition task is begun which accumulates data, converts the raw information into engineering units, checks for out-of-range conditions, and stores the results in a dynamic region. A disk update task is initiated whose function is to retrieve data from the dynamic region upon request of the data acquisition task and write the data to a file. A printing task is initiated whose function is to periodically print results for selected sensors at a user defined interval. A data display task is also invoked which periodically displays selected groups of data on a terminal using the FMS routines. Since the coal combustor data acquisition system uses slightly more than 100 sensors and the oil combustor data acquisition system uses slightly more than 300 sensors, it is obvious that only a small fraction of these may be displayed at any given time.

It was decided that for each system it was necessary to display a small subset of all data at all times in a primary display window, with a secondary window reserved for selected subsets of data. The users thus needed the capability of selecting which subset should be displayed in the secondary window. A problem arises, however, in allowing a periodic screen refresh (on the order of once every ten seconds) and simultaneously allowing user input to select the secondary window contents.

### Required features

A user option field is normally trivial to implement with the FMS-11 FGET routine. The problem, however, is that the read I/O request must be satisfied with user input. In this particular case, however, the data display must be periodically refreshed at a predetermined user rate without user intervention, i.e., the user should not have to request a screen update. The FGET routine (as well as other FMS routines) does not feature a timeout option on the read request. In order to provide the necessary functionality it was decided to add a read-with-timeout option to the DEC FMS-11 code.

A second feature not supplied by DEC was also deemed necessary for this application, namely run-time control of the video attributes of a particular field. The user has the option of setting high and low alarm points for all sensors, and the alarm flag state for each sensor is logged with the data when the data file is periodically updated. It was decided that the user should be given some visual indication on the CRT if one or more sensors on the current display exceeded either the high or low alarm points. The obvious choice would be to have the video attribute of the sensor data display set to "blink" for any out-of-range sensor, a capability not supplied by FMS-11.

### Implementation goals

The primary goal of this effort was to supply the needed functionality but with the absolute minimum modification of DEC code in the form driver package. In addition, any modification was to be restricted to a single routine if possible, and as much support code as possible was to be contained in subroutines independent of the form driver package.

## Investigating FMS-11

The FMS-11 manual supplied by DEC does not reveal the data structures used by the form driver. The source code distributed by DEC with the FMS-11 package is, for the most part, undocumented. Without documentation, one must resort to a little detective work.

### Researching terminal input

An initial attempt was made to trace the program flow from a high level entry point, such as the FGET call, down to the lowest level code which actually performed the QIO. This proved difficult since the code consists of subroutines which call subroutines which invoke macros which call subroutines which invoke macros, ..., and so on! The source code was searched with the EDT editor for any QIO's. A single character QIOW call was found in the FDVTIO subroutine. This QIOW performs a read with terminator table and no echo, using event flag T$IEFN. The code appears to read a single character into a buffer and pass it along to the routine which verifies that the input character is appropriate for the field that is being read. A legitimate character is then echoed to the terminal screen and the input process continues for another character. This seemed to be the appropriate point to add the timeout feature.

### Researching video attribute control

Researching the method of video attribute control required a bit more work. One can set any combination of four basic video attributes for any field on a form, namely: underline, bold, reverse video, and blinking. It was logical to assume that these attributes must be defined in some fashion in the impure area that must be reserved for each form that is to be displayed on the screen. A special program was written that displayed a trivial form on the screen, and then wrote a file containing the contents of the impure area after the form was displayed. Using the file dump utility DMP to look for ASCII fields in the impure area revealed that the field names are stored as six-byte ASCII fields. A minor change was made to the video attributes of a field on this test form and the contents of the impure area was again dumped to a file. The two files were then compared using the DMP and CMP utilities. After several simple changes were made to the test form and the impure areas compared as described above, a good deal of the data

structure maintained in the impure area was revealed. The location of the video attribute byte was readily apparent, as were several other control variables. The key item was that the location of the video attribute byte was at a fixed offset from the field name.

A segment of the basic data structure of a field entry in the impure data region may be represented as follows:

| | |
|---|---|
| Flag word | |
| Field Name | |
| ( six characters ) | |
| ( ASCII ) | |
| | |
| Clear Char | Video Attr |
| | |
| | # Field Char |
| # Def Char | |
| Field type | # Help Char |
| etc. | |

Though most of the entries of this structure have fixed offsets (from the assumed beginning of the structure), several form options may produce a data structure that is quite a bit longer than the segment pictured above. For example, if one has defined a default field content, the actual contents in ASCII follow the default field length byte immediately, displacing all that follows. Likewise, if help text is defined for a given field, it immediately follows the help text length byte, displacing anything that follows. The video attribute byte, however, is found at a fixed offset from the field name.

## Implementing the features

### Read-with-Timeout

In keeping with the twin goals of modifying the DEC code as little as possible and doing only what was necessary to accomplish the project requirements, it was decided to use the timeout feature for a single character option field rather than a field of arbitrary length. A single character field using decimal input allows up to ten user options, numbered 0 - 9, more than sufficient for this application. One could likewise support single character options containing alphanumerics and support many more options if necessary, though if the application became that complex, one should probably find a better way to implement it.

Adding the read-with-timeout for a single input character feature was simply a matter of changing the QIOW in subroutine FDVTIO to a QIO, adding a MRKT$ mark time directive for the desired timeout period, and performing a WTLO$ wait for logical OR of the event flags for the QIO and the mark time. The handling of user input proceeds as normal. In the case of a timeout, however, the code must supply data as if the user had entered it, echo it to the screen, and return it to the calling program.

Two global variables were introduced into the FDVTIO source, called $TMO$ and $DCHR$, the timeout period in seconds, and the default input character, respectively. Two simple subroutines were written to set the timeout period (subroutine FSTMO) and set the default character (subroutine FSDEFC). At execution time, a test is made on $TMO$ and if no timeout period is specified, the standard FDVTIO routine QIOW is processed normally. If $TMO$ is non-zero, the alternate QIO is executed and the mark time is issued. If the mark time event flag is detected prior to I/O completion, an IO.KIL is issued for the terminal QIO and the default character supplied by the calling program is inserted into the terminal buffer, simulating user input.

### Video Attribute Control

Once the data structure maintained in the impure area for a given field was understood, it was a simple matter to write a routine to permit run-time manipulation of a field's video attribute. Subroutine FMVID was written to scan the impure area for the field name, which is stored as a blank padded ASCII string. The video attribute byte for the field in question can then easily be changed to any legitimate value reflecting the desired combination of the following four possible attributes:

```
bit 0 - underline
bit 1 - reverse video
bit 2 - bold
bit 3 - blink
```

For example, to set a field to blinking bold, call FMVID with the video attribute byte argument set to $12_{10}$ or $14_8$. One assumption made in implementing the field name scan routine was that the name starts on a word boundary. It was also observed that the first field name never began less than $400_8$ bytes from the beginning of the impure area.

## Software availability

All of the software discussed above has been submitted to the Fall 1987 RSX SIG tape. The distribution consists of four Macro-11 assembly language subroutines (FMVID, FSTMO, FSDEFC, FRVID) and a SLP file to modify the FDVTIO subroutine of the form driver library. The SLP file may be applied to the FDVTIO.MAC subroutine source of FMS-11 V2.0 or V2.1.

# RT-11 SIG

# XL/XC/CL Programming for RT-11 / TSX-Plus

**Ned W.Rhodes**
Software Systems Group
2001 North Kenilworth Street
Arlington, VA 22205-3130

**Abstract**

Digital created a standardized way to interface with serial devices under RT-11 when they created the XL and XC handers. TSX-Plus adapted that standard and implemented what they called the CL handler. Through the use of these handlers and standard RT-11 programmed requests, user programs are able to acquire data and transmit and receive information from almost any serial device attached to an RT-11 or TSX-Plus system.

## Introduction

Users of RT-11 and TSX-Plus systems use RS-232 serial lines to transmit and receive information from other computer systems. With the advent of the cheaper personal computers, instrumentation manufacturers have started to build instruments that communicate to host computers over RS-232 serial lines. As a result, many RT-11 and TSX-Plus systems have the need to be able to communicate with a variety of RS-232 devices for communications and data acquisition.

There has been an evolution in the way that RT-11 and TSX-Plus users have approached serial communications. Initially, users wrote special purpose communications programs that implemented only ASCII character send and receive modes. The problem with this type of a solution was that each user wrote a different type of program that had a different user interface and "talked" to the serial ports on the system in a different way. Some of the later programs attempted to standardize the way that the programs "talked" to the serial ports by using a handler rather than an inline interrupt service routine, but each user-written handler used a different set of standards and protocols. Two examples of these early serial handlers are KB and MO, which can be found on early RT-11 DECUS swap tapes.

Digital utimately solved the problem of serial communications by writing a handler that could control serial devices and as a result, a standard was set. These Digital handlers were called XL on QBUS and UNIBUS machines and XC on the PRO series of computers. Digital started shipping these handlers with RT-11 Version 5. In order to maintain compatibility with RT-11, version 5.1 of TSX-Plus also supplied handlers that could control serial devices using the standards set by Digital. The TSX-Plus handler was called a CL handler or Communications Line handler.

This paper will provide an overview of the XL, XC and CL handlers and then go on to discuss the ways to interact with the handlers in order to transmit and receive serial data. An example program will be discussed that will demonstrate most of the communications features.

## Handler Overview

The RT-11 handlers XL and XC are similar in concept to the TSX-Plus CL handler, but all are implemented in different ways. The three handlers have the following characteristics :

- All are communication port controllers, which means that the handler is responsible for controlling a physical port on the computer.
- All are internally queued, which means that the handler will support full duplex communications (i.e. simultaneous transmit and receive).
- All use XON/XOFF flow control.
- All have internal buffers for the storage of characters which means that the applications program can read a few characters from the serial device, process those characters and then go back to read more characters without the fear of losing characters.
- All allow the application to control the DTR line.

Through the use of an applications program and the XL, XC, or CL handler, all types of serial devices can be controlled. In addition, programs can assume one standard interface to these handlers and the devices that are connected to an RT-11 or TSX-Plus system.

### RT-11 XL and XC handlers

The XL handler was originally included on the RT-11 distribution kit to work with VTCOM and TRANSF virtual terminal and file transfer utilities. Because of this, the XL handler is designed to support only one port of a DL-type serial interface. This means that a system will require four XL handlers to support all four serial ports on a DLV-11J interface board.

The XC handler is conditionally assembled from the XL handler sources for use on the PRO 350 or 380. In all other respects, the XL and the XC handler can be considered identical due to the fact that their both share a lot of common code.

### TSX-Plus CL Handler

The CL handler under TSX-Plus is compatible with programs that use the XL handler, but the implementation of the CL handler is very different. The CL handler is implemented as a system overlay instead of a handler, so that it consumes very little out of low memory. Also, the CL handler is a more general solution to the control of serial lines because it supports serial lines on DL, DZ or DH interface boards. And finally, the CL handler will support up to sixteen devices ( CL0-CL7 and C10-C17 ) for additional flexibility.

## XL, XC and CL Programming

Like all standard RT-11 and TSX-Plus handlers, the XL, XC and CL handlers support .READx and .WRITx requests and their equivalent formats contained in SYSLIB. The problem with the normal .READx and .WRITx requests is that only word oriented transfers are supported. Normally, byte oriented transfers are what is required when working with serial devices and the XL, XC and CL handlers support byte oriented transfer through the use of special function requests ( .SPFUN ).

The general use of the RT-11 programmed requests will not be discussed in this paper as they are fully explained in the RT-11 Programmer's Reference Manual. Instead, this paper will show how to use these programmed requests to communicate with the XL, XC and CL handlers.

### Special Function Requests

Special function requests are used to interact with a device in a device dependent manner. For example, RT-11 uses the standard calls .READ and .WRITE, to read and write data to a block oriented device in a device independent manner. Any program using these calls can be assured that data will be read or written to a disk without the programmer having to know the low level interface to the disk device. In other situations, the programmer needs to interact with a device at a low level. To do this, most RT-11 handlers have special function request codes so that programmer and device can interact at this very low level.

Special function request codes and the format of the special function request itself are documented in the RT-11 Programmer's Reference manual and will not be repeated here. The use of the special function request and the various codes will be shown in the example that will follow.

The following special function codes are useful in programming the XL, XC and CL handlers :

- #201 - This code will reset an XOFF condition. This may be useful in situations where the attached device sends an XOFF to the computer to signal the fact that data should not be sent until an XON is sent. If the device never sends the XON character, then the handler will not transmit any data until either an XON is received or the handler internal XOFF flag is reset with this special function code.
- #202 - This code allows a program to send a BREAK.
- #203 - This is a special read request that will be explained in the next section.
- #204 - This code will return the driver status to the program.
- #205 - This code will disable the handler interrupts with the net affect being that the handler will ignore any incoming characters.
- #206 - This codes allows the program to set and clear the DTR line.

### Special Read Special Function Request

Special function code #203 is used to read in character data from a serial device. This special function code is byte oriented and will return control to the user after :

- The number of characters requested in the byte count are available, or
- All available characters have been transferred, which is less than the byte count, or
- When one character is available when none were available when the special function read requested was posted.

All characters are returned in the user supplied buffer and the buffer is null-byte terminated due to the fact that the program never knows how many characters might be available when the read request is posted.

### TSX-Plus Only Special Functions Codes

The TSX-Plus CL handler has an additional 13 special function codes that are defined and documented in their documentation set. Programs should use these TSX-Plus specific special function codes only in cases where the program will only be used in the TSX-Plus environment. These special functions codes are NOT compatible with XL and XC handlers.

### XL, XC and CL Limitations

The following limitations apply when the XL, XC or CL handlers are used :

- The first call to the handler must use a block number of zero in the .READ, .WRITE or .SPFUN request. The zero in the block number signals to the handler that this is the first request so that the handler will initialize itself and enable the interrupts. All other requests after the first should use a block number of greater than zero.
- Null characters are ignored on both the transmit and receive sides of the handlers.
- The 8th bit is not transmitted or received normally. Under TSX-Plus, the eighth bit can be enabled with a set option. To accomplish the same under RT-11, the handler source code must be patched.

## Example Use of XL, XC and CL

The following example program demonstrates the use of RT-11 programmed requests to talk to a serial device using XL, XC or CL. The program was developed under TSX-Plus, but is compatible with RT-11 due to the fact that all the special function request codes that were used are supported under both TSX-Plus and RT-11.

The example routine is a simple automated dialing program that has four separate files associated with it. The first is a command file that is used under TSX-Plus to configure a particular CL port for dial out. The second file is a script file that tells the program what character sequences to send out and what character string to look for coming back. The third program is a Pascal routine that uses the functions and subroutines contained in the fourth file to communicate with the handlers. This fourth file contains code in the form or subroutines and functions that allow programs to read and write character strings to the handlers.

### TSX-Plus Command File

The following command file is used to setup a TSX-Plus line for use by the CL handler and the dialing program.

### Script Language

The script language is read by the dialing program. The script language contains a string to be sent out the CL device and the string that is expected back from the attached device. If the expected string is not read, then the dialing program will try again until the retry count has gone to zero. In that case, the program will just abort.

The script file uses the "~" character to indicate that a carriage return should sent. This special character is used so that it is obvious in the script file where carriage returns should be inserted.

### Pascal Controlling Routine

The Pascal controlling routine appended to this paper reads in the script file and attempts to dial and log into a remote system. This controlling routine calls subroutines that are contained in the XLSUBS file.

### XL Subroutines

Subroutines and functions needed to control the XL, XC or CL device are also appended to this paper. These routines initialize the CL device and then hang a read of 10 characters to the handler. When the handler completes the read, a completion routine is entered. In the completion routine, the characters are stored into a large ring buffer and then another read is hung to the CL device.

Using this type of design, the program will always be collecting data from the CL device and storing it in a local ring buffer. This also means that characters are immediately removed from the ring buffer in the CL device so that there is little chance that the CL handler will send an XOFF character because there is little chance that the handler ring buffer will ever become full.

Two pointers are used to access data in the ring buffer. One pointer is used to point to the next available location in the ring buffer, while the second pointer is used to point to the next character to be taken out of the ring buffer for the user. If both pointers are the same, then all the data has been taken out of the ring buffer. If both pointers are different, then there is information in the buffer that the user's program has not read yet.

## Conclusions

The XL, XC and CL handlers are good general purpose software tools that can be used to communicate with all types of serial devices. The use of any of these handlers allows an application to be easily moved from one machine to another. Finally the handlers provide a program with a flexible and standard environment for operation.

```
set cl3 line=3 ! Use cl3 and attach it to TSX line 3
set cl3 nolfout ! Don't send line feeds out
set cl3 tab ! Pass tabs through
set cl3 form ! Pass forms through
ass cl3 xl ! VTCOM likes to see either XL
ass cl3 xc ! or XC
set cl3 speed=2400 ! Speed of dial out line
allocate cl3: ! Allocate it to me
R DIAL ! Run the dial program
deas xl ! Deassign the logical names
deas xc ! Deassign the logical names
set cl3 line = 0 ! And return it to TSX-Plus
```

```
~       [Initially send a  C/R]
1 2 Loop      [Look one time, with a 2 second timeout]
//  R~    [Send this]
A       [Look for this]
3 2 LOOP [Loop 3 times, with a 2 second timeout]
~ [Send carriage return]
CDS >     [Look for this modem prompt]
1 15 loop  [Loop 1 time with a 15 second timeout]
D 340 0197~    [Dial the number]
ARQ ACTIVE   [Modem connected]
3 10 loop [Loop 3 times with a 10 second timeout]
<System Password>~ [Send this system password]
Username: [Wait for this prompt]
4 10 LOOP [Loop 4 times with a 10 second timeout]
<Username>~ [Send this username]
Password:  [Wait for this prompt]
2 30 Loop [Look 2 times with a 30 second timeout]
<Password>~ [Send this password]
$       [Wait for this prompt]
0 0 LOOP [Do nothing loop]
EXIT     [And exit dial]
```

```
PROGRAM dial;
{       Program to automatically dial and log into
        another computer system.  Uses device CL:
        and script file DIAL.DAT.
}
        CONST
        line_limit = 80;
        carriage_return = 13;
        line_feed = 10;
TYPE
        string = PACKED ARRAY [0 .. line_limit] OF
                 CHAR;
VAR
        inbuf,
        outbuf         : string;
        compare_string : string;
        crlf           : string;
        seconds        : INTEGER;
        times          : INTEGER;
        iloop          : INTEGER;
        time_out       : BOOLEAN;
        Got_string     : BOOLEAN;
        numb           : INTEGER;
        ch             : char;
        cmd_file,
        xlin,
        xlout,
        fout           : text;
        inchan,
        outchan        : INTEGER;

%include 'sy:stringas';        { Pascal string package }
```

```
function GetChannel ( var n : text): integer;
{
        This function gets the RT-11 channel number for a
        file opened by the Pascal run-time system.
}
%include 'SY:libdefas';

var
        f : user_file_variable;

begin
  f := loophole(user_file_variable,n);
  GetChannel := f^.channel;
end;

PROCEDURE Setup ( VAR inchn, outchn : INTEGER );
nonpascal;

PROCEDURE hang; nonpascal;     { Hang a read to XL }

FUNCTION Getxl : char; nonpascal;    { Get a character from XL }

PROCEDURE xlwrit ( VAR number : INTEGER; VAR
buf : string ); nonpascal;      { Write a string out XL }

PROCEDURE rtexit; nonpascal;    { RT-11 .exit }

PROCEDURE togdtr; nonpascal;    { Toggle DTR }

PROCEDURE Substitute ( VAR buffer : string );
{
        This procedure substitutes a real carriage return for
        a "~"
}
VAR
        iloop : INTEGER;
BEGIN
  FOR iloop := 1 to len(buffer) DO
  BEGIN
    IF buffer[iloop] = '~' THEN
       buffer[iloop]:= chr(carriage_return);
    IF buffer[iloop] = ''' THEN
       buffer[iloop]:= chr(line_feed);
  END;
END;
```

```
PROCEDURE init;
{
        This is the initialization routine.  It first opens two
        channels to the XL device.  One will be used for reading
        the other will be used for writing.  Next the routine
        converts the Pascal run-time channel numbers into RT-11
        channel numbers, calls the setup routine, toggles DTR
        and then writes the initial string out the XL device.
}
BEGIN
  Rewrite ( xlout, 'XL:' );
  Reset ( xlin, 'XL:' );
  Reset ( cmd_file, 'Dial.dat' );
  inchan := GetChannel ( xlin );
  outchan := GetChannel ( xlout );
  Setup ( inchan, outchan );
  ReadString ( cmd_file, outbuf );{Get init string }
  Trim(outbuf);                  { Trim trailing blanks }
  Substitute ( outbuf );
  togdtr;
  numb := len ( outbuf );
  xlwrit ( numb, outbuf );
  hang;
END;
```

```
FUNCTION readln_xl ( VAR buf : string;
                     VAR equal_string : string;
                         wait_time : INTEGER) : BOOLEAN;
{
        This function reads a string from the XL device and compares
        it to another string.  If the string returned from the XL device
        is not equal to the expected string, then the process is
        repeated.  In addition, if the string is not received within
        the timeout period, then function returns with a timeout error
}
VAR
        end_of_line : BOOLEAN;
        time_out    : BOOLEAN;
        ch_string   : string;
        start_time  : REAL;
        delta_time  : REAL;
BEGIN
  time_out := FALSE;
  end_of_line := FALSE;
  Clear ( buf );
  start_time := time;
  Repeat
    ch := Getxl;                             { Get a character from XL }
    IF ( ord(ch) <> 0 ) AND
       ( ord(ch) <> line_feed ) THEN
      IF ord(ch) = carriage_return THEN
      BEGIN
        end_of_line := TRUE;
        writeln;
        Clear ( buf );
      END
      ELSE BEGIN
             Write(ch);
             AssChar ( ch_string, ch );
             Concatenate ( buf, ch_string );
           END;
  delta_time := time;
  time_out := ( ( (delta_time-start_time) * 3600.0)
                  > wait_time );
  UNTIL (time_out) OR (equal (equal_string, buf) );
  IF time_out THEN BEGIN
                     readln_xl := FALSE;
                   END
              ELSE readln_xl := TRUE;
END;
```

```
BEGIN   { main routine }
  init;
  readln (cmd_file, times, seconds );   { repeat count and timeout }
  ReadString ( cmd_file, outbuf );
  Trim ( outbuf );
  Substitute ( outbuf );
  time_out := FALSE;
  WHILE (NOT equal ( outbuf, 'EXIT' ) ) DO
  BEGIN
    ReadString ( cmd_file, compare_string );
    Trim ( compare_string );
    iloop := 0;
    time_out := FALSE;
    numb := len ( outbuf );
    REPEAT
      xlwrit ( numb, outbuf );          { write this }
      got_string := readln_xl (inbuf,compare_string,
                                     seconds );
      iloop := succ(iloop);
      IF NOT got_string THEN writeln ( 'Timeout' );
    UNTIL ( iloop > times ) OR (got_string);
    IF got_string THEN
    BEGIN
      readln (cmd_file, times, seconds ); { repeat loop and timeout }
      ReadString ( cmd_file, outbuf );
      Trim ( outbuf );
      Substitute ( outbuf );
    END ELSE BEGIN
               WRITELN ('Too many retries.',
                        '  Program terminated');
               rtexit;
             END;
  END;
  rtexit;
END.
```

```
        .title  XL/XC/CL communications subroutines
;
;
;       Subroutines to control the XL/XC/CL ports
;       under RT-11 or TSX-Plus.
;
;       Ned W. Rhodes
;       Software Systems Group
;       2001 North Kenilworth Street
;       Arlington, VA  22205-3130
;
;
        .mcall  .ttyout
        .mcall  .readw,.writw
        .mcall  .exit,rint,.twait
        .mcall  .readc,.ttyout
        .mcall  .read,.wait,.qset
        .mcall  .spfun
        .ident  /V1.0/
        .enabl  lc
        .nlist  bex
        bufsiz = 1000.          ; Large ring buffer

        .sbttl  SETUP
        .psect  code
;
;
;       Turn off the interrupt handler
;
;
xloff::
        .spfun  #area,xlwrt,#205,#rbuf,#5.,blk
        return
```

```
;
;
;       RT-11 hard exit.  Will exit even with files open
;
;
rtexit::.exit
;
;
;       Toggle the DTR line
;
;
togdtr::
        .spfun  #area,xlwrt,#206,#rbuf,#0,blk   ; lower it
        .twait  #area,#twosec                   ; wait 2 seconds
        .spfun  #area,xlwrt,#206,#rbuf,#1,blk   ; raise it
        return
setup::
        .qset   #queue,#10.             ;Add more queue elements
        mov     @2(r5),xlread           ; Get the read channel
        mov     @4(r5),xlwrt            ; Get the write channel
;
;
;       zero out the ring buffer
;
;
2$:     mov     #bufsiz,r0      ; n words
        mov     #ring,r1        ; the address of the buffer
3$:     clrb    (r1)+           ; clear a word
        sob     r0,3$           ; and loop
        mov     #ring,cur       ; initialize pointer
        mov     #ring,next      ; initialize pointer
        return                  ; and return
```

```
hang::
;
;
;       Hang a read to xl
;
;
        .spfun  #area,xlread,#203,#rbuf,#10.,blk,#xlcomp
                                ; hang a read
        bcc     4$              ; no error
        mov     #reader,r0      ; get the message
        jmp     error           ; and say the error
4$:     mov     #1,blk          ; Set block number to 1
        return                  ; and return

        .sbttl  XL read completion routine
;
;
;       Read characters and start a read for more
;
;
xlcomp: mov     xloop:  movb    (r1)+,@next    ; save in ring buffer
;       .ttyout @next           ; Debug print
        inc     next            ; bump pointer
        cmp     #rend,next      ; at the end?
        bne     5$              ; nope
        mov     #ring,next      ; reset address
5$:     tstb    (r1)            ; A null?
        bne     xloop           ; Nope, transfer more
        .spfun  #area,xlread,#203,#rbuf,#10.,blk,#xlcomp
                                ; hang another read
        return                  ; and return
```

```
        .sbttl  GETXL
        .page
;
;
;       Take a character out of the ring buffer
;
;
getxl:: clr     r0              ; start with a zero
        cmp     cur,next        ; are pointers the same?
        beq     6$              ; yes, exit
        movb    @cur,r0         ; get the character
        inc     cur             ; bump pointer
        cmp     #rend,cur       ; at the end
        bne     6$              ; nope
        mov     #ring,cur       ; reset address
6$:     return                  ; and return

        .sbttl  XLWRIT
        .page
;
;
;       Write characters to XL
;
;       Pascal strings start in the second byte of the
;       string.  The length of the string is contained
;       in the first byte.
;
;       This routine uses the .writw programmed request to
;       write one word at a time to the CL device.  In
;       reality only one byte at a time is sent as we are
;       forming a word that consist of the character to be
;       sent and a null.  As we know, the handler will ignore
;       the null and so only one byte will be transmitted
;       at one time.
;
xlwrit::mov     @2(r5),r2       ; Get the byte count
        mov     4(r5), r1       ; Get buffer address
        inc     r1              ; For strings
8$:     movb    (r1)+,baddr     ; Save in a word
        .writw  #area,xlwrt,#baddr,#1,blk  ; write it up
        sob     r2,8$
        return                  ; and return
```

```
        .sbttl  Error exit
error:  .print                  ; say the error
        .exit

        .sbttl  Data and Storage
        .psect  data
queue:  .blkw   140.            ; additional queue elements
area:   .blkw   10.             ; emt area
xlread: .word   0               ; XL read channel
xlwrt:  .word   0               ; XL write channel
ring:   .blkb   bufsiz          ; ring buffer
rend    =       .               ; end of buffer address
        .even
cur:    .word   ring            ; current pointer
next:   .word   ring            ; next empty position
rbuf:   .blkw   100.            ; Receive buffer
twosec: .word   0,60.*2         ; 2 seconds
blk:    .word   0               ; Block counter
wcnt:   .word   0               ; Word count
baddr:  .word   0               ; Output Character


        .psect  msgs
reader: .asciz  /?SETUP-F-Error posting a read to XL/
hit:    .asciz  /?XLCOMP-I-Got a read from XL/
        .end
```

# REAL WORLD DISK COMPARISONS

Robert C. Peckham
Computer Programming Services, Glendale, CA
&
Milton D. Campbell
Talisman Systems, Manhattan Beach, CA

## ABSTRACT

Many computer users are interested in the actual data transfer rates achieved when real controllers and disks operate with a real operating system, doing real data transfers, as compared with the data transfer rates claimed in manufacturers' literature.

Test programs were written to exercise the various operational parameters of a disk, while doing the type of transfers that might be observed in real-world applications. These test programs were run on a wide variety of disks by approximately twenty DEC end-user sites.

The test programs were run on disks ranging from RX01 through the more common cartridge disks, on to some relatively large and exotic Winchester and memory disks, and even on an Ethernet virtual disk.

The results are presented in tabular form so that direct comparison is possible. The results of this project are very interesting to those interested in real-world disk performance.

## INTRODUCTION

The authors and, we discovered, a significant number of other people, were interested in the actual data transfer rates achieved when real controllers and disks operate through a real operating system, doing real data transfers.

This paper presents the results of a group of test programs which were run on numerous disk and controller combinations.

In disk-based operating systems, which all PDP-11 and VAX systems are, the performance of the system disk, and any auxiliary disks, has a major impact on system performance. Most sites have no realistic way to compare the price/ performance characteristics of one disk against another, or the absolute performance of any disk in their system, before purchase.

Many sites have had the disappointing experience of purchasing a disk based on salesmens' claims, or based on printed "performance specifications", and subsequently discovering, to their chagrin, that those specifications meant relatively little in a "real world" environment.

The test sequence which produced the results reported in this paper consisted of a group of programs which created and manipulated files with a variety of file layouts, with the primary measurement being the elapsed time for the test.

The process of getting data to and from a disk involves both hardware and software. When a user program requests a disk operation, the operating system fields the request and eventually issues the necessary commands to the disk controller. The controller translates these commands into hardware instructions to the disk drive, which causes data to be read from the disk, transferred back through the controller, and eventually into memory, where it is available to the user program.

The first step is for the operating system to handle the request. The system may have to load the device handler, do a context switch, swap jobs, or do other housekeeping before it can actually start issuing commands to the disk controller. The time consumed in this process is called "system latency".

Once commands are issued to the controller, the controller may have processing to do before it starts sending instructions to the disk. This delay is "controller latency".

Much of the time, a disk transfer will require that the disk head be moved to a track other than the current one. There is a significant delay involved in this process, since the head positioning system is an electro-mechanical device and responds relatively slowly. This "seek delay" is usually quite large and is often the overriding factor in disk access time.

Once the head has been moved to the proper track, the system must wait until the desired sector moves under the head. This "rotational latency" is basically a function of the rotational speed of the disk.

Finally, the disk subsystem is ready to transfer data into memory. The upper limit on this process, or the "peak transfer rate", is the speed at which bit cells pass under the disk head. This maximum rate may be degraded by delays caused by the buss DMA transfer system (i.e., if the buss cannot keep up with data coming off the disk). It is common to use techniques such as interleaving to ensure that as few disk rotations as possible are required to read the data once the beginning of the desired data has been found. In most cases, the data transfer rate is not a major contributor to the time it takes to access the desired data.

There are other places where time can be lost in this process, which might be categorized as "overhead". Interrupts that interfere with the system's disk-handling software may cause delays that result in extra rotational latencies. Interrupts and higher priority DMA devices may cause the controller-to-memory data transfer to fall behind the disk-to-controller transfer. If this delay is large enough, further rotational latencies may occur. Depending on the operating system file structure, more than one seek and read may be required to service a single user program data request. Those systems that scatter portions of files around the disk may need to "collect" the scattered data needed by the user program, and they will probably need to read and update the various bookkeeping data areas to keep track of which disk areas are in use and which are available.

Because we wanted "real world" results, all disk activity in the test programs run by the various sites occurred through the normal operating system I/O system. Therefore, the various latencies discussed above are included in the test program run times.

The test programs were written in FORTRAN. This means that the FORTRAN run-time library overhead is included, which would be typical of any system using higher order languages.

## RT-11 OPERATING SYSTEM

The host operating system for the study was RT-11, single-job monitor, Version 5.0. RT-11 is characterized by fairly low I/O system overhead; however, most of the tests used named files on the test device. This means that an RT-11 file system was created on the disk being tested. The overhead involved in opening and closing files is included in the test results. Since directory processing is a significant part of most disk activity, a brief description of the RT-11 file system is in order.

An RT-11 directory is kept in the low numbered blocks of a disk, beginning in block 6. (Blocks 0 through 5 contain boot and identification information.) The directory is allocated (at disk initialization time) in two-block chunks, called segments, up to a maximum of 31 (decimal) segments. The number of directory segments is either user-specified or is determined by the number of total blocks available on the device. Each directory segment may contain up to seventy-two file entries.

RT-11 files are allocated as contiguous blocks from the available "empty" space on a disk. Each file requires one directory entry. In addition, each "empty" area requires a directory entry so that at any time the full disk space is described in the directory, either as allocated to files or as "empty".

The segments in the directory are connected together in a forwardly linked list, with files that have lower starting block numbers appearing in the directory before files with higher block numbers. Since the directory is organized by block number, a directory search by file name (the usual operation) is performed sequentially from the front of the directory until either the file or the end of the directory is found.

The RT-11 directory processing time costs should be affected primarily by two different hardware factors. The location of the directory in the low numbered blocks of the disk means that when a directory operation is needed, a seek will usually be required. In addition, since "new" files tend to be at the opposite end (from the directory) of the "used" portion of the disk, this seek will be longer than the "average" based simply on the disk usage pattern. The second component, once the directory has been read, will be the CPU time involved in searching through the directory.

In an attempt to make the effect of the directory organization on the study as uniform as possible, the tests were always run on an "empty" (i.e., freshly initialized) disk, with the file creation and manipulation performed by the test programs and distributed command files.

## LIMITS ON THE APPLICABILITY

This study was aimed at attempting to measure disk sub-system performance in the "real world". Since the authors use RT-11, the "real world" in this case was defined to be RT-11. In general, we were not trying to measure the maximum speed a disk could provide, nor were we trying to find optimum disk access techniques. The objective in the design of the test programs used in this study was to reproduce many of the circumstances encountered in the use of a disk in a normal interactive RT-11 environment. "Real Time" RT-11 is a good operating system environment for doing disk benchmarking.

The way a system is used at a site will affect hardware selection decisions. In particular, the results of this study do not necessarily apply to all RT-11 environments. The authors are primarily interested in general purpose systems used for program development, word processing, and business support, with some multi-user activities. This type of environment tends to have many fairly small files, accessed in a more-or-less random fashion. The test programs used were somewhat skewed to this type of disk use and are not necessarily a good guide to other environments.

## FACTORS NOT CONSIDERED

Disk speed is, obviously, not the only consideration when purchasing mass storage. Besides the major factors of capacity needs and the pocketbook, other items to consider are: maintenance (who, how, and at what cost?); DEC compatibility (emulation?, special drivers?); file backup (on what, how long it takes?); reliability, and vendor "track record".

## TEST PROGRAMS

This study used eleven programs that, combined with several different arrangements of data files, were designed to measure various aspects of disk performance with the primary "result" measure being the wall clock time required to complete the test(s).

TEST1: Created, wrote, and closed 150 one-block files.

Because the actual data transfered to the files was short, this test was primarily a seek test, particularly between the disk directory area and the data area. Actual data written was 76,800 bytes.

TEST2: Created, wrote, and closed one 150-block file.

This test was a measure of data transfer rate on a medium size file. Actual data written was 76,800 bytes.

TEST3: Created, wrote, and closed 300 one-block files.

This test was similar to TEST1. TEST3 created data files for use by later test programs. Actual data written was 153,600 bytes.

TEST4: Using the 300 one-block files created by TEST3, this program pseudo-randomly selected a file, opened, read, and closed the file, modified one data element of the file, then opened, wrote, and closed the same file, until all 300 files had been processed. Actual data read and written was 307,200 bytes.

This test primarily measured random block latency. The large number of directory operations means that the directory processing portion of the operating system was exercised. An effective data and directory caching scheme would speed up this process considerably by reducing the many seeks involved.

(Tests 4, 4A, 4B, 5 and 5A were written anticipating that any effective directory and/or data caching scheme would significantly reduce run time.)

TEST4A: Performed the same operations as TEST4, but the "write" sequence was performed on the null device, NL:. Asymmetry in "read" and "write" operations may be apparent when compared to TEST4. Actual data read and written was 307,200 bytes.

TEST4B: This test was similar to TEST4, except that the files were opened, read and closed only. Actual data read was 153,600 bytes.

This test looked for symmetry in read and write operations and was very seek-intensive. A large directory caching system would help significantly for this test.

TEST5: Using the 300 one-block files originally created by TEST3, this test sequentially opened, read and closed a file, then opened, wrote, and closed the file on NL:. Actual data read and "written" was 307,200 bytes.

This test was intended to look at the sequential performance of the disk. The large number of directory operations caused a large number of seeks and exercised the directory processing software. It was anticipated that a disk caching system would excel on this test.

TEST5A: Similar to TEST5, this program sequentially opened, read, and closed each of the 300 one-block files. Actual data read was 153,600 bytes.

ALTERNATE TESTS 4, 4A, 4B, 5, 5A: Two command files, SPACE2 and SPACE8, were used to insert two dummy files (2000-block and 8000-block, respectively) into the midst of the 300 data files in an attempt to increase required seek distances.

Tests 4, 4A, 4B, 5 and 5A were run both before and after this spacing-out of the data files.

TEST6: Created, wrote (sequentially), and closed one 300-block file. Actual data written was 153,600 bytes.

The run time result is useful for comparison purposes, and the 300 block data file was also needed for TEST7.

TEST7: Read the 300-block file left by TEST6, then wrote the file on NL:. Actual data read was "written" was 307,200 bytes.

TEST8: Created, wrote, and closed one 800-block file. Actual data written was 409,600 bytes.

This test was to compare data rates, writing a large formatted data file.

TEST9: Opened, read, and closed the 800-block file from TEST8. Acutal data read was 409,600 bytes.

Compare TEST8 and TEST9 results to look at read/write symmetry.

TEST10: Created, wrote, and closed 1000 five-block files. Actual data written was 2,560,000 bytes.

This test stressed the disk subsystem's sequential and random access capability. The transfer speed of the device should have been a relatively minor component.

TEST11: Read either the whole disk, or 32000 blocks (whichever was smaller), sequentially in 16384 byte "chunks". Compute byte transfer rate and report as "Figure of Merit" (FOM).

This test was designed to get a good idea of the disk's maximum effective read data transfer rate. The large size of the data buffer minimized the operating system/device handler overhead and allowed the device to operate at high transfer rates efficiently. Because the reads were performed at the device level (i.e., little system interference), there is no directory processing overhead. The sequential nature of the reads reduced the latency to single track seeks and the irreducible rotational latency of the device. It turns out that the buffer size used in this test has a measurable effect on the result found. For most disks, the optimal buffer size would be as large as possible, but one that is an even multiple of the number of disk blocks per disk track. The number used (32 512-byte blocks) is non-optimal for many devices, but its large size tends to reduce this effect.

The results gathered in this study are summarized in Appendices A through I.

Appendix A is the manufacturers' specification and rating for each of the various systems and disks tested, if available. The study participants supplied this data.

Appendix B is a summary of the various systems tested. Systems without a disk indicated used RT-11's VM: memory disk.

Appendix C is the VM: results which is useful for normalizing memory, CPU, and buss speed.

Appendix D is the results of the programs run on floppy disks. Note the almost total CPU independence.

Appendix E is the results for the smaller cartridge and Winchester disks.

Appendix F is the results for the larger cartridge and Winchester disks.

Appendix G is the results for disks not directly comparable due to CPU or buss type.

Appendix H is an operating system comparison. It was put in because the authors found it very interesting.

Appendix I is the results for "strange" pseudo disks. Note the outstanding performance of Ethernet and bulk semi "disks". Note that bulk semi performance is very dependent on the type of buss.

DISCUSSION

Not surprisingly, the disks with the fastest seek speed generally did the best in the tests. This confirms "theoretical" expectations, and corresponds to the authors' subjective experience.

An aspect of seek performance that can be important is the way the blocks of a disk are organized. In systems that emulate DEC controllers by partitioning a large disk into several smaller drives, the partitioning map can be important. For example, the three system combinations labled I1, I2, and I3 (in Appendix B) have essentially the same hardware performance specifications, but the controller used in I1 partitions its disk's logical devices so that each platter in the drive is a logical device, while the controllers in I2 and I3 use groups of contiguous cylinders for each logical device. The effect of the second method is that the blocks of a logical device are closer to each other than with the first method, resulting in better performance.

For the larger disks, several of the tests (4 thru 5a) were run several times with a reorganization of the disk between test series. The reorganization placed two large files in the midst of the test data files, which was intended to extend the seek travel distance for many of the seeks. In terms of measuring the difference between disk subsystems, this did not produce very interesting results! The relative performance of the various disks remained essentially unchanged, which indicates that none of the disks had a hardware caching system. These 'spaced-out' results were omitted except in the operating system appendix (H) where the effect of software caching was obvious.

While this "spacing-out" procedure produced no meaningful comparison data, it did turn up a useful anomaly that can be seen in Appendix H. In almost all of the sub-systems the performance results after the SPACE2 command file were better than the results before SPACE2. The results after the SPACE8 command file were slower than after the SPACE2 results, but faster than initial ones. This is because the disk directory was squeezed after the two large "spacer" files were added. The reduction in directory processing after adding the two 2000 block files more than made up for the increased seek travel. The increase in run time with two 8000 block space files replacing the two 2000 block files is consistent with increased seek travel and vividly demonstrates the importance of apple to apple disk comparisons.

Another interesting observation is the performance of the DEC RA-80 system on an 11/24 (system N1). For TEST11, which is basically a maximum speed sequential read, the RA-80 showed a remarkable transfer speed, but on the other tests, involving more random access, the results for the RA-80 were more mundane and placed the system with the rest of the "high end" pack.

The CPU's used for the tests were 11/2, 11/03, 11/23, 11/23+, 11/24, 11/34, and 11/73. All but the 11/24 and the 11/34 were Q-bus systems.

CPU speed did make a difference, although there were relatively few examples of the same disk sub-system with different CPU and buss types. This does not provide much opportunity for direct comparisons. The two Unibus systems performed well, but since there was no Q-bus system using the same disk it was not possible to make any direct comparisons. The exception to this was the Dataram bulk-semi which gave outstanding performance on both the Q-bus and U-bus. We note that it was three times as outstanding on the U-bus which does say something about bus speed!

The TEST11 Figure of Merit (FOM) was actually a somewhat idealized maximum data rate (bytes/sec) obtained by bypassing the RT "High level I/O facilities" and using very large buffers. In general, however, the FOM was MUCH lower than published disk performance specifications. Some of the tests that manipulated numerous small formatted files produced average data rates of less than 5000 bytes/sec, even with the fastest disks.

Appendix H shows the results of running the test programs on various operating systems and CPU's while keeping the disk system the same. As one might expect, the more sophisticated systems tend to have lower transfer rates. There was not much difference between the RT-11 Single-Job Monitor and the Foreground/Background Monitor. The XM system and TSX-Plus had significant overhead penalty. A very interesting result is shown by the TSX-PLUS with disk caching column. For most of the tests, caching produces a significant improvement over TSX-PLUS without caching. In most cases the caching system allows TSX-PLUS to beat the Single Job system, but in the raw throughput case of TEST11, the caching system overhead dramatically impedes performance.

The results of these disk comparisons demonstrate that the performance figures quoted by manufacturers are not useable for calculating the actual performance of a disk-based system which does disk I/O. On the other hand, the test results do correlate relatively well from a disk-to-disk comparison standpoint. Although cost was not discussed in this paper, the authors think it is worth noting that, to a large extent, the performance observed in these disk comparisons correlates relatively well with the cost of the disks and disk controllers exercised during this test program.

## 1984 CONCLUSIONS

1. We did not test a disk with a hardware-caching system. The TSX+ software caching performance indicates that an effective hardware-caching system would be of significant value for many I/O loads.

2. The CPU and operating system can be more important than the disk sub-system. Depending on the operating system, a faster disk system may provide very little improvement.

3. The type of operating system affects the apparent disk performance. The more complex and capable operating systems provided lower data transfer rates than the RT-SJ monitor in our tests. DEC sells RT as a "fast" operating system with low overhead compared to RSX and RSTS.

4. The variations we found suggest that disk system selection on any basis but a "test drive" is fairly risky. Vendor-published performance data is not a good indicator of actual disk performance in an operating computer system.

## 1987 CONCLUSIONS

1. Disk controllers with cache did produce very significant improvements in system performance where the disk subsystem was seek and/or rotational latency bound.

2. A site's unique hardware can have a significant effect on performance.

3. A "test drive" is still recommended.

4. In many "real world" situations, a "memory" disk is of little or no benefit over a disk with a "caching" controller.

5. Controller cache is better than system data cache.

The authors wish to thank all of the DEC sites and personnel who volunteered to run these test programs. Running the programs required the exclusive use of the system, one or more initialized volumes, and several hours of work on the part of the participants. Thus, it represented a considerable amount of time and trouble, and the authors could not have done this work without the active and enthusiastic participation of the people involved.

APPENDIX A

Description of computer systems.

A CPU type: 11/23 Memory: DEC MSV11-LK

A1 Controller and Disk: DEC RXV21/RX02
Rated "average" data transfer rate: 62K bytes/sec
Rated average seek time: 180 msec

--------------------------------------------------------------------------------

A2 Controller: EMULEX SC02/C, emulates RK06
Disk Brand: Fujitsu 2312
Rated "peak" data transfer rate: 1200K bytes/sec
Rated track-to-track seek time: 5 msec
Rated average seek time: 20 msec

********************************************************************************

B CPU type: 11/23
Memory: Motorola MMS1132 (1/4 MB); TI TMM10010 (1/4 MB)

********************************************************************************

C CPU type: 11/23 Memory: Chrislin CI-1123

Controller: Interlan NI2010 Ethernet interface 'Disk' is Ethernet going to a VAX 11/750 and SI Eagle disk connected through a Quniverter to a Q-bus Ethernet interface. Special software. Caching provided by VAX system.

Rated "peak" data transfer rate: 290K bytes/sec
Rated average seek time: `18msec

********************************************************************************

D1 CPU type: 11/23 Memory: DEC

Controller: Andromeda WDC11-A/B
Disk Brand: Andromeda CM 5616
(Computer Memories 12.6 MB formatted)

--------------------------------------------------------------------------------

D2 Controller and Disk: DSD 440; emulates RX02
Rated "peak" data transfer rate: 20K bytes/sec
Rated track-to-track seek time: 8 msec
Rated average seek time: 296 msec

********************************************************************************

E CPU type: 11/23 Memory: National Semi NS23M

--------------------------------------------------------------------------------

E2 Controller: Sigma SDC-RXV31; emulates RX02
Disk Brand: Mitsubishi M2696-63

--------------------------------------------------------------------------------

E3 Controller: Sigma SDC-RLV12; emulates RL02
Disk Brand: Rodime R204 with Xebec S1410 formatter

********************************************************************************

F1 CPU type: 11/23 Memory: DEC MSV11-LK

Controller: DSD 880; emulates RX02
Disk Brand: DSD 880/20
Rated "peak" data transfer rate: 20K bytes/sec
Rated "average" data transfer rate: 18K bytes/sec
Rated track-to-track seek time: 18 msec
Rated "average" seek time: 174 msec

--------------------------------------------------------------------------------

F2 Controller: DSD 880; emulates RL02
Disk Brand: DSD 880/20 (special handler)
Rated "peak" data transfer rate: 204K bytes/sec
Rated "average" data transfer rate: 143.8K bytes/sec
Rated track-to-track seek time: 15 msec
Rated "average" seek time: 60 msec

--------------------------------------------------------------------------------

F3 Controller: Emulex SC03; emulates RM03
Disk Brand: Alpha Data - Atlas
Rated "peak" data transfer rate: 1200M bytes/sec
Rated "average" data transfer rate: 942K bytes/sec
Rated track-to-track seek time: 8 msec
Rated "average" seek time: 18 msec

```
*****************************************************************************
G       CPU type:  11/23   Memory:  DEC MSV11-LK
-----------------------------------------------------------------------------
G1              Controller and Disk:  DEC RLV11/RL01
                Rated "average" data transfer rate:  512K bytes/sec
                Rated track-to-track seek time:  17 msec
                Rated average seek time:  55 msec
*****************************************************************************
H1      CPU type:  11/23   Memory:  Camminton

                Controller:  Emulex Sabre System; emulates RL02
                Disk Brand:  Atasi drive
-----------------------------------------------------------------------------
H2              Controller:  Plessey PM DCV 11A ; emulates RK05
                Disk Brand:  Diablo Model 30 disk
*****************************************************************************
I       CPU TYPE:  11/23   Memory:  Black Bear BBE-256
-----------------------------------------------------------------------------
I1              Controller:  Dilog  DQ200; emulates extended RK05
                Disk Brand:  Control Data  9448-96
                             (CMD, 14 inch cartridge, 16 removable, 80 fixed)
                Rated "peak" data transfer rate:  1200K bytes/sec
                Rated track-to-track seek time:  7 msec
                Rated average seek time:  30 msec
-----------------------------------------------------------------------------
I2              Controller:  Dilog 202A; emulates RP02
                Disk Brand:  Control Data 9730-160
                             MMD - 160 MB winchester, 14 inch)
                Rated "peak" data transfer rate:  1200K bytes/sec
                Rated track-to-track seek time:  7 msec
                Rated average seek time:  30 msec
-----------------------------------------------------------------------------
I3              Controller:  Dilog DQ-215; emulates RK07
                Disk Brand:  Control Data 9715-160
                             (FSD  8 inch winchester)
                Rated "peak" data transfer rate:  1200K bytes/sec
                Rated track-to-track seek time:  7 msec
                Rated average seek time:  30 msec
-----------------------------------------------------------------------------
I4              Controller and Disk:  DEC RXV21/RX02
-----------------------------------------------------------------------------
I5              Controller and Disk:  DEC RLV12/RL02
                Rated "average" data transfer rate:  512K bytes/sec
                Rated average seek time:  55 msec
-----------------------------------------------------------------------------
I6              11/23 CPU with Peritek boards that allow DMA
                transfers from high memory to low.  The function
                is similar to VM:, but the actual transfer is
                performed by DMA hardware rather than by CPU
                instructions.
-----------------------------------------------------------------------------
I7              Controller and Disk:  DEC RXV11/RX01
                Rated "peak" data transfer rate:  50K bytes/sec
                Rated track-to-track seek time:  6 msec/track
*****************************************************************************
J1      CPU type:  11/23+   Memory:  Chrislin

                Controller:  Dilog DQ 100; emulates RK05
                Disk Brand:  Diablo 31; 2.5 MB
                Rated "average" data transfer rate:  180K bytes/sec
                Rated track-to-track seek time:  15 msec
                Rated average seek time:  70 msec
-----------------------------------------------------------------------------
J2              Controller and Disk:  DSD 440; emulates RX02
*****************************************************************************
K1      CPU type:  11/2   Memory:  Black Bear BBE 256

                Controller and Disk:  RXV21/RX02
-----------------------------------------------------------------------------
K2              Controller and Disk:  DEC RLV12/RL02
*****************************************************************************
```

```
L1      CPU type:  11/03   Memory:  DEC

               Controller:  DSD 880; emulates RL02
               Disk Brand:  DSD 880
*****************************************************************************
M       CPU type:  Prototype 11/73 with 10 MHZ clock
        Memory:  Black Bear BBE 256
-----------------------------------------------------------------------------
M1             Controller and Disk:  DEC RXV21/RX02
*****************************************************************************
N       CPU type:  11/24   Memory:  DEC M7891
-----------------------------------------------------------------------------
N1             Controller:  UDA 50
               Disk Brand:  DEC RA 80 winchester
               Rated "peak" data transfer rate:  2200K bytes/sec
               Rated track-to-track seek time:  6 msec
               Rated average seek time:  28 msec
*****************************************************************************
O       CPU type:  11/34   Memory:  DEC MS11-LD
-----------------------------------------------------------------------------
O1             Controller and Disk:  RL11/RL01
               Rated "average" data transfer rate:  512K bytes/sec
               Rated track-to-track seek time:  12.5 msec
               Rated average seek time:  55 msec
*****************************************************************************
P       CPU type:  11/23   Memory:  Clearpoint 22B (1 Meg)
*****************************************************************************
Q1      CPU type:  11/23   Memory:  Clearpoint 22B (1 Meg)

               Controller:  Dilog DQ-202A; emulates RP02/03
               Disk brand:  Fujitsu 2284, 160 Meg
               Rated "peak" data transfer rate:  1000K bytes/sec
               Rated track-to-track seek time:  6 msec
               Rated average seek time:  27 msec
*****************************************************************************
R1      CPU type:  11/73   Memory:  Clearpoint 22B (1 Meg)

               Controller:  Dilog DQ-202A; emulates RP02/03
               Disk brand:  Fujitsu 2284
*****************************************************************************
S1      CPU type:  11/73   Memory:  Clearpoint 22B (1 Meg)

               Controller:  Dilog DQ-202A; emulates RP02/03
               Disk brand:  Fujitsu 2284
               Disk caching:  software data and directory caching
*****************************************************************************
T1      CPU type:  11/23+  Memory:  Dataram DR-223    Block mode NOT used

               Disk brand:  Dataram BS-202    (bulk semi) RF emulation
               Rated "peak" data transfer rate:  500K bytes/sec
               Rated track-to-track seek time:  NA
*****************************************************************************
U1      CPU type:  11/34   Memory:  Dataram DR-244    Block mode NOT used

               Disk brand:  Dataram BS207/MC207  (bulk semi) RF emulation

               Rated "peak" data transfer rate:  1460K bytes/sec
               Rated track-to-track seek time:  NA
*****************************************************************************
V       CPU type:  11/23+  Memory:  MDB  MLSI-MSV11LK
-----------------------------------------------------------------------------
V1             Controller:  MDB MLSI-WFC11
               Disk brand:  TANDOM; emulates RX02
-----------------------------------------------------------------------------
V2             Controller:  MBD MLSI-WFC11
               Disk brand:  TANDOM; emulates RL02
```

APPENDIX B

THIS APPENDIX CONTAINS A BRIEF DESCRIPTION OF THE SYSTEMS

| SYS | CPU | MEMORY | CONTROLLER | DISK | EMULATES |
|---|---|---|---|---|---|
| A | 11/23 | DEC | | | |
| A1 | 11/23 | DEC | RXV21 | RX02 | |
| A2 | 11/23 | DEC | EMULEX SC02/C | FUJITSU 2312 | RK06 |
| B | 11/23 | MOTOROLA, TI | | | |
| C | 11/23 | CHRISLIN | INT NI2010 | ETHERNET | SPECIAL |
| D1 | 11/23 | DEC | AND WDC11-A/B | AND CM 5616 | RK05 |
| D2 | 11/23 | DEC | DSD440 | DSD440 | RX02 |
| E | 11/23 | NAT SEMI | | | |
| E2 | 11/23 | NAT SEMI | SIGMA SDC-RXV31 | MITSUBISHI M2696-63 | RX02 |
| E3 | 11/23 | NAT SEMI | SIGMA SDC-RLV12 | RODIME R204 | RL02 |
| F1 | 11/23 | DEC | DSD880 | DSD880/20 | RX02 |
| F2 | 11/23 | DEC | DSD880 | DSD880/20 | RL02 |
| F3 | 11/23 | DEC | EMULEX SC03 | ATLAS | RM03 |
| G | 11/23 | DEC | | | |
| G1 | 11/23 | DEC | RLV11 | RL01 | |
| H1 | 11/23 | CAMMINTON | EMULEX SABRE | ATASI | RL02 |
| H2 | 11/23 | CAMMINTON | PLESSEY DCV11A | DIABLO 30 | RK05 |
| I | 11/23 | BLACK BEAR | | | |
| I1 | 11/23 | BLACK BEAR | DILOG DQ200 | CDC 9448-96 | RK05 |
| I2 | 11/23 | BLACK BEAR | DILOG DQ202A | CDC 9730-160 | RP02 |
| I3 | 11/23 | BLACK BEAR | DILOG DQ215 | CDC 9715-160 | RK07 |
| I4 | 11/23 | BLACK BEAR | RXV21 | RX02 | |
| I5 | 11/23 | BLACK BEAR | RLV12 | RL02 | |
| I6 | 11/23 | BLACK BEAR | PERITEK | | SPECIAL |
| I7 | 11/23 | BLACK BEAR | RXV11 | RX01 | |
| J1 | 11/23+ | CHRISLIN | DILOG DQ100 | DIABLO 31 | RK05 |
| J2 | 11/23+ | CHRISLIN | DSD440 | DSD440 | RX01 |
| K1 | 11/2 | BLACK BEAR | RXV21 | RX02 | |
| K2 | 11/2 | BLACK BEAR | RLV12 | RL02 | |
| L1 | 11/03 | DEC | DSD880 | DSD880 | RL02 |
| M | 11/73 | BLACK BEAR | | | |
| M1 | 11/73 | BLACK BEAR | RXV21 | RX02 | |
| N | 11/24 | DEC | | | |
| N1 | 11/24 | DEC | UDA50 | RA80 | |
| O | 11/34 | DEC | | | |
| O1 | 11/34 | DEC | RL11 | RL01 | |
| P | 11/23 | CLEARPOINT | | | |
| Q1 T+ | 11/23 | CLEARPOINT | DILOG DQ202A | FUJITSU(14") | RP02/03 |
| R | 11/73 | CLEARPOINT | | | |
| R1 | 11/73 | CLEARPOINT | DILOG DQ202A | FUJITSU(14") | RP02/03 |
| S1 T+ | 11/73 | CLEARPOINT | DILOG DQ202A | FUJITSU(14") | RP02/03 |
| T1 | 11/23+ | DATARAM | DATARAM bulk memory | BS202 | RF11 |
| U1 | 11/34 | DATARAM | DATARAM bulk memory | BS207/MC207 | RF11 |
| V | 11/23+ | MDB | | | |
| V1 | 11/23+ | MDB | MDB | TANDOM | RX02 |
| V2 | 11/23+ | MDB | MDB | | RL02 |

APPENDIX C

| System: | A | B | E | G | I | P | M | R | N | O | V |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Test | | | | | | | | | | | |
| 1 | 0:42 | 0:35 | 0:35 | 0:35 | 0:35 | 0:36 | 0:25 | 0:23 | 0:36 | 0:30 | 0:48 |
| 2 | 0:25 | 0:25 | 0:25 | 0:24 | 0:25 | 0:25 | 0:17 | 0:13 | 0:25 | 0:21 | 0:28 |
| 3 | 1:22 | 1:20 | 1:20 | 1:21 | 1:22 | 1:22 | 0:59 | 0:43 | 1:24 | 1:11 | 1:33 |
| 4 | 2:04 | 2:03 | 2:03 | 2:04 | 2:05 | 2:05 | 1:25 | 1:02 | 2:08 | 1:49 | 2:23 |
| 4a | 1:56 | 1:55 | 1:55 | 1:54 | 1:57 | 1:56 | 1:18 | 0:56 | 1:59 | 1:42 | 2:14 |
| 4b | 1:04 | 1:03 | 1:03 | 1:04 | 1:04 | 1:04 | 0:43 | 0:30 | 1:06 | 0:55 | 1:14 |
| 5 | 1:56 | 1:54 | 1:55 | 1:55 | 1:56 | 1:56 | 1:16 | 0:56 | 2:00 | 1:42 | 2:14 |
| 5a | 1:04 | 1:04 | 1:03 | 1:03 | 1:04 | 1:04 | 0:43 | 0:30 | 1:06 | 0:56 | 1:14 |
| 11 | 281K | 281K | 281K | 281K | 274K | 281K | 314K | 399K | 310K | 407K | 262K |
| CPU | 23 | 23 | 23 | 23 | 23 | 23 | 73 | 73 | 24 | 34 | 23+ |
| Memory Brand | DEC | MOT TI | NAT SEMI | DEC | BLACK BEAR | CLEAR POINT | BLACK BEAR | CLEAR POINT | DEC | DEC | MDB |

APPENDIX D

| System: | A1 | I4 | E2 | D2 | F1 | M1 | K1 | J2 | I7 | V1 |
|---|---|---|---|---|---|---|---|---|---|---|
| Test | | | | | | | | | | |
| 1 | 3:26 | 3:26 | 3:00 | 2:57 | 2:58 | 2:57 | 3:59 | 4:03 | 4:32 | 3:17 |
| 2 | 0:37 | 0:38 | 0:37 | 0:53 | 0:36 | 0:30 | 1:18 | 0:58 | 1:01 | 0:53 |
| 3 | 11:45 | 11:44 | 9:20 | 9:13 | 9:15 | 9:11 | 12:59 | 13:01 | 19:34 | 11:05 |
| 4 | 13:43 | 13:46 | 8:49 | 8:18 | 8:23 | 7:40 | 15:34 | 13:20 | 17:08 | 13:01 |
| 4a | 7:54 | 7:56 | 5:23 | 5:07 | 5:16 | 4:32 | 9:44 | 7:42 | 9:32 | 7:40 |
| 4b | 6:53 | 6:53 | 4:32 | 4:13 | 4:15 | 3:53 | 7:51 | 6:45 | 8:36 | 6:32 |
| 5 | 7:54 | 7:56 | 5:25 | 5:08 | 5:16 | 4:31 | 9:43 | 7:42 | 9:32 | 7:41 |
| 5a | 6:55 | 6:55 | 4:30 | 4:10 | 4:12 | 3:47 | 7:51 | 6:42 | 8:36 | 6:35 |
| 6 | 1:14 | 1:15 | 1:14 | 1:47 | 1:13 | 0:59 | 2:37 | -- | -- | 1:44 |
| 7 | 2:01 | 2:03 | 2:02 | 2:34 | failed | 1:30 | 4:13 | -- | -- | 2:42 |
| 11 | 17.6K | 17.7K | 17.5K | 17.8K | 17.7K | 17.7K | 15.6K | 8.9K | 8.9K | 17.7K |
| CPU | 23 | 23 | 23 | 23 | 23 | 73 | 11/2 | 23+ | 23 | 23+ |
| CONT<br>DISK | DEC<br>RX02 | DEC<br>RX02 | SIGMA<br>MITSUB | DSD<br>440 | DSD<br>880 | DEC<br>RX02 | DEC<br>RX02 | DSD<br>440<br>RX01 | DEC<br>RX01 | MDB<br>TANDOM |

APPENDIX E

| System: | D1 | G1 | O1 | F2 | H2 | J1 | V2 |
|---|---|---|---|---|---|---|---|
| Test | | | | | | | |
| 1 | 1:07 | 0:53 | 0:50 | 0:53 | 1:13 | 0:56 | 0:55 |
| 2 | 0:26 | 0:26 | 0:24 | 0:27 | 0:33 | 0:29 | 0:30 |
| 3 | 2:41 | 2:21 | 2:12 | 2:18 | 3:27 | 2:21 | 2:23 |
| 4 | 4:00 | 3:22 | 3:13 | 3:19 | 4:44 | 3:30 | 3:31 |
| 4a | 2:55 | 2:35 | 2:25 | 2:35 | 3:25 | 2:47 | 2:49 |
| 4b | 2:04 | 1:43 | 1:38 | 1:43 | 2:24 | 1:46 | 1:48 |
| 5 | 2:56 | 2:35 | 2:25 | 2:35 | 3:24 | 2:47 | 2:49 |
| 5a | 2:00 | 1:43 | 1:37 | 1:42 | 2:24 | 1:47 | 1:48 |
| 6 | 0:52 | 0:52 | 0:48 | 0:52 | 1:05 | NR | 1:00 |
| 7 | 1:43 | 1:42 | 1:31 | 1:46 | 2:00 | NR | 2:00 |
| 8 | 2:19 | 2:20 | 2:08 | 2:17 | -- | -- | 2:38 |
| 9 | 2:25 | 2:26 | 2:07 | 2:33 | -- | -- | 2:50 |
| 10 | -- | 29:08 | 27:47 | 27:39 | -- | -- | 29:37 |
| 11 | 81K | 197K | 187K | * | 88K | 88K | 96.7K |
| CPU | 23 | 23 | 34 | 23 | 23 | 23+ | 23+ |
| CONTROLLER | ANDRO | DEC | DEC | DSD | PLES | DILOG | MDB |
| DISK | ANDRO RK05 | RL01 | RL01 | 880 RL02 | DIABLO RK05 | DIABLO RK05 | TANDOM RL02 |

APPENDIX F

| System: | A2 | E3 | F3 | H1 | I1 | I2 | I3 | I5 |
|---|---|---|---|---|---|---|---|---|
| Test | | | | | | | | |
| 1 | 0:44 | 0:56 | 0:41 | 0:52 | 1:09 | 0:38 | 0:44 | 0:54 |
| 2 | 0:27 | 0:27 | 0:27 | 0:30 | 0:21 | 0:21 | 0:21 | 0:28 |
| 3 | 1:52 | 2:30 | 1:46 | 2:08 | 2:42 | 1:39 | 1:39 | 2:20 |
| 4 | 2:44 | 3:29 | 2:35 | 3:08 | 3:43 | 2:24 | 2:25 | 3:20 |
| 4a | 2:17 | 2:38 | 2:12 | 2:35 | 3:08 | 1:58 | 1:58 | 2:36 |
| 4b | 1:25 | 1:46 | 1:20 | 1:36 | 1:53 | 1:13 | 1:13 | 1:42 |
| 5 | 2:16 | 2:38 | 2:11 | 2:36 | 3:08 | 1:58 | 1:58 | 2:36 |
| 5a | 1:25 | 1:47 | 1:19 | 1:36 | 1:53 | 1:14 | 1:14 | 1:43 |
| 6 | 0:52 | 0:53 | 0:53 | 0:59 | 0:41 | 0:42 | 0:42 | 0:56 |
| 7 | 1:41 | 1:42 | 1:42 | 1:58 | 1:19 | 1:27 | 1:27 | 1:44 |
| 8 | 2:19 | 2:22 | 2:21 | 2:36 | 1:48 | 1:51 | 1:52 | 2:27 |
| 9 | 2:23 | 2:25 | 2:24 | 2:48 | 1:51 | 2:03 | 2:04 | 2:27 |
| 10 | 23:56 | 29:25 | 23:21 | 26:30 | 24:44 | 21:20 | 21:11 | 29:20 |
| 11 | 254K | 73K | * | 211K | 220K | 254K | 252K | 197K |
| CPU | 23 | 23 | 23 | 23 | 23 | 23 | 23 | 23 |
| CONTROLLER | EMULEX | SIGMA | EMULEX | EMULEX | DILOG | DILOG | DILOG | DEC |
| DISK | FU2312 RK06 | RODIME RL02 | ATLAS RM03 | ATASI RL02 | CDC RK05 | CDC RP02 | CDC RK07 | RL02 |

APPENDIX G

| System: | K2 | L1 | N1 | R1 |
|---|---|---|---|---|
| Test | | | | |
| 1 | 1:24 | 1:23 | 0:50 | 0:30 |
| 2 | 0:52 | 0:51 | 0:33 | 0:13 |
| 3 | 3:23 | 3:20 | 1:56 | 1:29 |
| 4 | 5:12 | 5:02 | 2:48 | 1:55 |
| 4a | 4:25 | 4:20 | 2:24 | 1:24 |
| 4b | 2:39 | 2:35 | 1:29 | 0:57 |
| 5 | 4:25 | 4:19 | 2:24 | 1:24 |
| 5a | 2:39 | 2:35 | 1:29 | 1:58 |
| 6 | 1:43 | 1:43 | 1:00 | 0:26 |
| 7 | 3:27 | 3:24 | 1:55 | 0:44 |
| 8 | 4:34 | 4:31 | 2:27 | 1:10 |
| 9 | 4:51 | 4:45 | 2:40 | 1:09 |
| 10 | NR | 42:07 | 23:28 | 18:14 |
| 11 | NR | * | 479K | 224K |

APPENDIX H

| System: | 11/23 RT-SJ | 11/23 RT-FB | 11/23 RT-MTXM | 11/23 TSX+ | 11/73 RT-SJ | 11/73 TSX+ | 11/73 TSX+ (Caching) |
|---|---|---|---|---|---|---|---|
| Test | | | | | | | |
| 1 | 0:44 | 0:43 | 0:54 | 0:53 | 0:30 | 0:35 | 0:27 |
| 2 | 0:27 | 0:26 | 0:33 | 0:34 | 0:13 | 0:15 | 0:15 |
| 3 | 1:52 | 1:52 | 2:14 | 2:00 | 1:29 | 1:40 | 1:02 |
| 4 | 2:44 | 2:46 | 3:20 | 2:42 | 1:55 | 2:07 | 1:07 |
| 4a | 2:17 | 2:18 | 2:52 | 2:38 | 1:24 | 1:30 | 1:02 |
| 4b | 1:25 | 1:25 | 1:43 | 1:29 | 0:57 | 1:03 | 0:35 |
| 5 | 2:16 | 2:18 | 2:52 | 2:38 | 1:24 | 1:29 | 1:01 |
| 5a | 1:24 | 1:25 | 1:43 | 1:29 | 1:58 | 1:03 | 0:35 |
| @SPACE2 | X | X | X | X | X | X | X |
| 4 | 2:23 | 2:25 | 3:01 | 2:47 | 1:33 | 1:44 | 1:10 |
| 4a | 2:06 | 2:08 | 2:43 | 2:33 | 1:14 | 1:18 | 1:00 |
| 4b | 1:14 | 1:15 | 1:33 | 1:24 | 0:47 | 0:52 | 0:35 |
| 5 | 2:07 | 2:05 | 2:42 | 2:33 | 1:13 | 1:18 | 0:59 |
| 5a | 1:14 | 1:15 | 1:32 | 1:24 | 0:48 | 0:52 | 0:33 |
| @SPACE8 | X | X | X | X | X | X | X |
| 4 | 2:27 | 2:28 | 3:03 | 2:47 | 1:37 | 1:48 | 1:10 |
| 4a | 2:08 | 2:09 | 2:43 | 2:33 | 1:16 | 1:20 | 1:00 |
| 4b | 1:15 | 1:16 | 1:35 | 1:24 | 0:48 | 0:53 | 0:32 |
| 5 | 2:09 | 2:09 | 2:43 | 2:33 | 1:15 | 1:20 | 0:59 |
| 5a | 1:16 | 1:16 | 1:34 | 1:24 | 0:48 | 0:53 | 0:32 |
| 6 | 0:52 | 0:52 | 1:05 | 1:07 | 0:26 | 0:29 | 0:30 |
| 7 | 1:41 | 1:42 | 2:13 | 2:10 | 0:44 | 0:48 | 0:46 |
| 8 | 2:19 | 2:18 | 2:53 | 2:59 | 1:10 | 1:18 | 1:20 |
| 9 | 2:23 | 2:22 | 3:08 | 3:13 | 1:09 | 1:09 | 1:19 |
| 10 | 23:56 | 24:01 | 28:13 | 25:14 | 18:14 | 19:55 | 11:54 |
| 11 | 254K | 253K | 253K (VBGEXE) | 219K | 224K | 219K | 123K |

APPENDIX I

| System: | C | I6 | T1 | U1 |
|---|---|---|---|---|
| Test | | | | |
| 1 | 1:12 | 0:35 | 0:36 | 0:27 |
| 2 | 0:26 | 0:21 | 0:26 | 0:20 |
| 3 | 2:51 | 1:26 | 1:23 | 1:01 |
| 4 | 3:55 | 2:05 | 2:08 | 1:23 |
| 4a | 3:20 | 1:47 | 2:01 | 1:20 |
| 4b | 1:58 | 1:04 | 1:06 | 0:49 |
| 5 | 3:20 | 1:48 | 2:02 | 1:19 |
| 5a | 1:57 | 1:04 | 1:06 | 0:50 |
| 6 | 0:53 | 0:43 | 0:51 | 0:40 |
| 7 | 1:43 | 1:27 | 1:47 | 1:22 |
| 8 | -- | -- | 2:17 | 1:47 |
| 9 | -- | -- | 2:27 | 1:52 |
| 10 | -- | -- | 18:29 | 13:50 |
| 11 | 123K | 251K | 360K | 1042K |
| CPU | 23 | 23 | 23+ | 34 |
| HARDWARE | INTERLAN ETHERNET | PERITEK MEMORY | DATARAM BULK | DATARAM BULK |
| HANDLER | SPECIAL | SPECIAL | RF11 | RF11 |

APPENDIX L

Description of computer systems.

87A CPU type: 11/73 Memory: Camintonn (2 meg)

---

87A1 RT-11 operating system

Controller: EMULEX SC02/C, emulates RK07
Disk Brand: Fujitsu 2322
Rated "peak" data transfer rate: 1200K bytes/sec
Rated track-to-track seek time: 5 msec
Rated average seek time: 20 msec

---

87A2 TSX+ V6.1 operating system
1 meg data cache. Directory cache: 140 entries.
No other system users, but detached job WINPRT
(window print) running.
TSX+ started just before tests.

Controller: EMULEX SC02/C, emulates RK07
Disk Brand: Fujitsu 2322
Rated "peak" data transfer rate: 1200K bytes/sec
Rated track-to-track seek time: 5 msec
Rated average seek time: 20 msec

****************************************************************************

87B CPU type: 11/73 Memory: National 1 meg.

RT-11 operating system

Controller: Sigma RQD11/SCH
Disk Brand: CDC XMD 850
Rated "peak" data transfer rate: 3000K bytes/sec
Rated track-to-track seek time: 5 msec
Rated average seek time: 16 msec

---

87B1 TSX+ V6.01 operating system

Same as system 87B.

****************************************************************************

87C CPU type: 11/23 Memory: DEC MSV11-QA 1 meg.

---

87C1 Controller: RQDX2
Disk Brand: RD53

---

87C2 Controller: RLV12
Disk Brand: RL02
Rated "average" data transfer rate: 512K bytes/sec
Rated average seek time: 55 msec

****************************************************************************

87D CPU type: 11/73 Memory: Camintonn (512K)

Controller: DILOG DQ696
Disk Brand: MAXSTOR 4380E
Rated "peak" data transfer rate: 1250K bytes/sec
Rated track-to-track seek time: 3 msec
Rated average seek time: 16 msec

---

87D1 Controller: DILOG DQ246
Disk Brand: FUJITSU 2333
Rated "peak" data transfer rate: 2460K bytes/sec
Rated track-to-track seek time: 5 msec
Rated average seek time: 20 msec

```
********************************************************************************
87E     CPU type:  11/73               Memory:  CHRISLIN

                Controller:  ETHERNET (IRANET SOFTWARE)
********************************************************************************
87F     CPU type:  11/73

                Controller: ANDROMEDA SMDC (caching disabled)
--------------------------------------------------------------------------------
87F1    CPU type:  11/73

                Controller:  ANDROMEDA SMDC (caching enabled)
********************************************************************************

87G     CPU type:  11/23+              Memory:  DEC
-------------------------------------------------------------------------------

87G1    CPU type:  11/73               Memory:  DEC

                Controller:  RQDX1
                Disk Brand:  RD51
-------------------------------------------------------------------------------

87G2    CPU type:  11/73               Memory:  DEC

                Disk Brand:  RX50
-------------------------------------------------------------------------------

87G3    CPU type:  11/73               Memory:  DEC

                Controller:  RLV12
                Disk Brand:  RL02
******************************************************************************
```

APPENDIX M (1987)

THIS APPENDIX CONTAINS A BRIEF DESCRIPTION OF THE SYSTEMS

| SYS | CPU | MEMORY | | CONTROLLER | DISK | EMULATES |
|---|---|---|---|---|---|---|
| 87A | 11/73 | CAMINTONN | | | VM | |
| 87A1 | 11/73 | CAMINTONN | RT | EMULEX SC02/C | FUJITSU 2322 | RK07 |
| 87A2 | 11/73 | CAMINTONN | TSX | EMULEX SC02/C | FUJITSU 2322 | RK07 |
| 87B | 11/73 | NATIONAL | RT | SIGMA RQD11/SCH | CDC XMD850 | DU |
| 87B1 | 11/73 | NATIONAL | TSX | SIGMA RQD11/SCH | CDC XMD850 | DU |
| 87C | 11/23 | DEC | | | VM | |
| 87C1 | 11/23 | DEC | | RQDX2 | RD53 | |
| 87C2 | 11/23 | DEC | | DEC RLV12 | RL02 | |
| 87D | 11/73 | CAMINTONN | | DILOG DQ696 | MAXSTOR 4380E | DU |
| 87D1 | 11/73 | CAMINTONN | | DILOG DQ246 | FUJITSU 2333 | DU |
| 87E | 11/73 | CHRISLIN | | ETHERNET VIRTUAL DISK | | SPECIAL |
| 87F | 11/73 | | | ANDROMEDA SMDC | | DU |
| 87F1 | 11/73 | | | ANDROMEDA SMDC | CACHE ENABLED | DU |
| 87G | 11/23+ | DEC | | | VM | |
| 87G1 | 11/23+ | DEC | | RQDX1 | RD51 | |
| 87G2 | 11/23+ | DEC | | | RX50 | |
| 87G3 | 11/23+ | DEC | | RLV12 | RL02 | |

APPENDIX N (1987)

| System: | 87A | 87C | 87G |
|---|---|---|---|
| Test | | | |
| 1 | 0:18 | 0:38 | |
| 2 | 0:12 | 0:26 | |
| 3 | 0:42 | 1:27 | |
| 4 | 1:00 | 2:14 | |
| 4a | 0:54 | 2:06 | |
| 4b | 0:30 | 1:09 | |
| 5 | 0:54 | 2:05 | |
| 5a | 0:30 | 1:10 | |
| 6 | 0:23 | 0:52 | |
| 7 | 0:44 | 1:48 | |
| 11 | 406K | 262K | 278K |
| CPU | 73 | 23 | 23+ |
| MEMORY | CAMIN | DEC | DEC |

APPENDIX O (1987)

| System: | 87C1 | 87C2 | 87G1 | 87G2 | 87G3 |
|---|---|---|---|---|---|
| Test | | | | | |
| 1 | 0:59 | 0:56 | | | |
| 2 | 0:36 | 0:28 | | | |
| 3 | 2:20 | 2:27 | | | |
| 4 | 3:12 | 3:31 | | | |
| 4a | 2:39 | 2:45 | | | |
| 4b | 1:43 | 1:48 | | | |
| 5 | 2:39 | 2:45 | | | |
| 5a | 1:43 | 1:47 | | | |
| @SPACE2 | | | | | |
| 4 | 2:50 | 3:12 | | | |
| 4a | 2:29 | 2:37 | | | |
| 4b | 1:32 | 1:39 | | | |
| 5 | 2:29 | 2:37 | | | |
| 5a | 1:32 | 1:38 | | | |
| @SPACE8 | | | | | |
| 4 | 2:56 | 3:33 | | | |
| 4a | 2:32 | 2:48 | | | |
| 4b | 1:34 | 1:50 | | | |
| 5 | 2:32 | 2:48 | | | |
| 5a | 1:35 | 1:48 | | | |
| 6 | 1:05 | 0:56 | | | |
| 7 | 1:58 | 1:54 | | | |
| 8 | 2:40 | 2:28 | | | |
| 9 | 2:44 | 2:46 | | | |
| 10 | 27:29 | 29:56 | | | |
| 11 | 146K | 184K | 141K | 11K | 196K |
| CPU | 23 | 23 | 23+ | 23+ | 23+ |
| CONT | RQDX2 | RLV12 | RQDX1 | | RLV12 |
| DISK | RD53 | RL02 | RD51 | RX50 | RL02 |

APPENDIX P (1987)

| System: | 87A1 | 87A2 | 87D | 87D1 | 87E |
|---|---|---|---|---|---|
| Test | | | | | |
| 1 | 0:29 | 0:29 | 0:29 | 0:31 | 0:42 |
| 2 | 0:13 | 0:15 | 0:14 | 0:14 | 0:13 |
| 3 | 1:23 | 1:06 | 1:19 | 1:22 | 1:38 |
| 4 | 1:49 | 1:11 | 1:42 | 1:46 | 2:04 |
| 4a | 1:20 | 1:05 | 1:17 | 1:19 | 1:49 |
| 4b | 0:55 | 0:38 | 0:51 | 0:53 | 1:02 |
| 5 | 1:19 | 1:04 | 1:16 | 1:18 | 1:48 |
| 5a | 0:54 | 0:38 | 0:51 | 0:54 | 1:02 |
| @SPACE2 | | | | | |
| 4 | 1:30 | 1:13 | 1:24 | 1:29 | |
| 4a | 1:10 | 1:02 | 1:09 | 1:12 | |
| 4b | 0:45 | 0:34 | 0:42 | 0:45 | |
| 5 | 1:09 | 1:01 | 1:08 | 1:11 | |
| 5a | 0:44 | 0:35 | 0:43 | 0:45 | |
| @SPACE8 | | | | | |
| 4 | 1:33 | 1:15 | 1:26 | 1:31 | |
| 4a | 1:13 | 1:01 | 1:09 | 1:13 | |
| 4b | 0:47 | 0:35 | 0:44 | 0:47 | |
| 5 | 1:12 | 1:01 | 1:08 | 1:12 | |
| 5a | 0:47 | 0:34 | 0:44 | 0:47 | |
| 6 | 0:26 | 0:28 | 0:27 | 0:28 | 0:26 |
| 7 | 0:47 | 0:48 | 0:50 | 0:50 | 0:46 |
| 8 | 1:08 | 1:17 | 1:12 | 1:12 | |
| 9 | 1:07 | 1:05 | 1:12 | 1:12 | |
| 10 | 16:58 | 12:44 | 16:06 | 16:49 | |
| 11 | 236K | 120K | 483K | 632K | 524K |
| CPU | 73 | 73 TSX | 73 | 73 | 73 |
| CONT | EMULEX | SC02/C | DQ696 | DQ246 | ETHERNET |
| DISK | FUJITSU | 2322 | MAXSTOR 4380E | FUJI 2333 | VIRTUAL DISK |

APPENDIX Q (1987)

| System: | 87B | 87B1 | 87F | 87F1 |
|---|---|---|---|---|
| Test | | | | |
| 1 | 0:28 | 0:26 | 0:35 | 0:26 |
| 2 | 0:13 | 0:14 | 0:14 | 0:14 |
| 3 | 0:54 | 1:02 | 1:27 | 0:56 |
| 4 | 1:04 | 1:18 | 1:49 | 1:08 |
| 4a | 0:54 | 1:04 | 1:18 | 0:58 |
| 4b | 0:30 | 0:37 | 0:54 | 0:35 |
| 5 | 0:54 | 1:03 | 1:17 | 0:57 |
| 5a | 0:31 | 0:38 | 0:55 | 0:34 |
| @SPACE2 | | | | |
| 4 | 1:01 | 1:13 | 1:26 | 1:06 |
| 4a | 0:54 | 1:02 | 1:08 | 0:58 |
| 4b | 0:29 | 0:35 | 0:44 | 0:34 |
| 5 | 0:52 | 0:59 | 1:07 | 0:55 |
| 5a | 0:28 | 0:34 | 0:43 | 0:32 |
| @SPACE8 | | | | |
| 4 | 1:04 | 1:14 | 1:27 | 1:05 |
| 4a | 0:55 | 1:03 | 1:07 | 0:57 |
| 4b | 0:32 | 0:37 | 0:44 | 0:33 |
| 5 | 0:53 | 1:01 | 1:07 | 0:54 |
| 5a | 0:29 | 0:35 | 0:44 | 0:31 |
| 6 | 0:26 | 0:27 | 0:26 | 0:26 |
| 7 | 0:43 | 0:47 | 0:47 | 0:46 |
| 8 | 1:10 | 1:12 | 1:08 | 1:08 |
| 9 | 0:59 | 1:04 | 1:07 | 1:03 |
| 10 | 10:48 | 12:24 | 16:10 | 10:33 |
| 11 | 569K | 569K | 574K | 583K |
| CPU | 73<br>C ON | 73<br>TSX | 73<br>C OFF | 73<br>C ON |
| CONT | SIGMA RQD11/SCH | | ANDROMEDA SMDC | |
| DISK | CDC XMD 850 | | | |

# SITE, MANAGEMENT AND TRAINING SIG

# Diary of a Novice System Manager

**Mark Roark Chartier**
Systems & Logistics Corporation
1887 Business Center Dr., Ste. 1A
San Bernardino, CA 92408

**Abstract**

What are some of the tasks and challenges facing a person who is suddenly placed in charge of setting up and operating a computer system? What knowledge must he/she bring or strive to obtain quickly in order to successfully set-up, organize and maintain the facility and to provide the users with fine, reliable service? During his/her apprenticeship, formal or otherwise, what types of observations, routines and thoughts might help to avoid disastrous mistakes and to conquer the inevitable problems?

These themes are developed and explored through the experiences of Mr. Chartier, a still-fledgling system manager who, in spite of having an educational background and job experience in a field completely unrelated with computers, was suddenly confronted with the opportunity to learn on an operating computer system, with all its accompanying responsibilities.

Still profoundly awed by the computer, and without taking his developing skills too much for granted, Mr. Chartier offers with enthusiasm the story of his successes and failures to urge others on and give them confidence.

I have taken for a motto a phrase that I tell the people at our site: "I'd do anything for a user."

This paper is intended for newly-crowned System Managers who possibly have obtained their title through:

- company purchase of a new departmental computer
- vacancy of the previous system manager's post
- creation of a new post of system manager
- other vicissitudes of life.

I don't know how you obtained your job as system manager, whether by some bold desire, by some sheepish offering: "I'll try," or whether boss just pointed his finger and said: "It's you!" I got my job via "Other vicissitudes of life." I had absolutely no idea that I would be a system manager. I had spent the last ten years of my life in Spain restoring pipe organs. But my boss happened to know me from my University days and he knew that the company needed a system manager. He thought that I had a good technical mind and that I would have the right attitude towards the job. So in spite of the fact that my only previous computer experience was programming a small, programmable calculator, (an experience that would be of extreme use later in my job), he called me to see if I would be interested in the job. He called several times to Spain to explain what the job would entail. The initial description of the job made me acutely aware of the importance of the post to the company and with that, the accompanying responsibility. I was really nervous about taking on such responsibility. I thought: "Can I do it? Can I give a useful service? Can I learn to make responsible decisions operating the computer in terms of efficiency and of not making dreadful mistakes? Can I learn all that I will need to know to do the job?" Well my boss had some encouraging words: "Sure you can do it!" He said that the only way to learn about computers was to work with them.

After several months on the job, I reflected upon my special situation. Not everybody is going to be in as nice of a spot as I was. My post was a full time job, not a little hobby or extra duty I did only during off hours. My boss was computer knowledgeable; he could serve as my guru. Finally, the system was already running, so, although I did not have the headaches of getting a system installed and up, on the other hand, I did have the added responsibility: users' files were already out there and I had to take care of them.

Well, enough for this. I came to the door of the building for the first day of work and I was scared! There was no little-by-little build up to the job: I had just opened the door when boss whisked me into the computer room saying, "This is the computer ... Look at this! This is the backplane ... This is a DZQ-11 ... This is a modem/multiplexer ... This is a DB-25 connector!" Ad infinitum!!! And there I was frantically taking notes. Boy, all this stuff running through my head! It was an immedi-

ate exposure to the vocabulary and to the physical aspects and appearance of the equipment.

The day didn't end with that dizzying session: they needed to make an account for me. So boss said to the present system manager, "Make Mark an account." Reply: "With what privilege?" "Well, full system privilege— after all he's going to be the system manager." Then he turned around to me and said, "Don't crash the system!!!" That did wonders to calm my nerves! I had thought: "Maybe there'll be this little *practice* computer upon which I can make all my mistakes in a special environment apart from that of the users." Not so at all; all mistakes were paid for. I learned to log on and to log off. Then I was introduced to this two meters or more stretch of shelves which contained the formidable VAX/VMS Documentation Set. Now the "easy" assignment was: "Learn it!" I ended the day providing service by doing regular backups.

The next three weeks of the job were equally exciting. I had to deal with an erroneous maintenance manual that told one how not to load a ribbon into the printer. To rectify this I had to write a procedure telling other people how to change that ribbon, a fairly complicated task. In order to write the procedure, I had to get into the editor and learn how to create and save (or not save) my edits. On the third week I went to the Spring 1987 DECUS symposium at Nashville. I returned with such an exposure to the computer that, driving down the street in the car, I began seeing DCL commands in the license plates. When that happens you know that your life will never be the same—you are hooked!

Now some additional observations: I learned quickly that there would be two qualities very important to the job, 1) exquisite observations of events that I saw and 2) very careful thinking. An example that illustrates this occurred to me when I tried to enroll a new user. I looked with the `AUTHORIZE` command `SHOW` , and I got a User Authorization record on the screen. I saw that it had a field labeled "DEFAULT" and I thought: "That must be the default directory." Well, unknown to me, that field was to be input as two separate parameters: device and directory. So I typed `/DEFAULT=` thinking that I was going to specify the user's login default directory in his UAF record. But I observed that the field didn't change. And when I exited `AUTHORIZE` a message came up: **`Rights Database Modified.`** That made me sort of nervous, so I logged back on and saw that I had corrupted the `DEFAULT` record thinking all the while that I was changing the default of the user. Boy, was I embarrassed to explain that to my boss. But it was the close observation of the message that gave me a clue: Mark, you blew it!!

### What is the System Manager's Job?

Now the system manager's job is very simple, at least to explain. Only two words: provide service. There are two general types of services: direct and indirect. The direct services, examples of which might be the enrollment of new users, the mounting of volumes for processing and the modification of account parameters, are usually visible to users. The indirect services, such as the installation of operating system upgrades and new software, the arrangement for and supervision of maintenance and the performance of regular backups, are more transparent to users. One might ask the perhaps unfortunate riddle: How can one distinguish between direct and indirect service to users? Well, one doesn't notice the results of indirect services ... until they're not there.

The correct employment of the two types of services gives a great service to the company, giving a return on the company's investment in the computer, aiding the productivity of the computer's users and offering them a pleasant working environment.

I learned quickly that the system manager has a variety of tasks, many of which are not done from the operator's console. In my job, for example, I have donned overalls to climb over ceilings and crawl beneath furniture to install cabling; I've read schematics, soldered wires and assembled connectors; taken apart and repaired floppy drives and printers, devices which I had never seen before; I've built bookcases, been a librarian; telephoned all over the country trying to solve problems; written letters, some of them complaints; organized and overseen maintenance; read catalogues, ordered supplies; worked on fixing the overheat alarm for the computer; I've been a draftsman; re-inked ribbons, (a tremendously messy job); all these, aside from the computer activities.

I was a greenhorn as a system manager and I really didn't know too many of the acronyms. Our maintenance company brochure said: "One individual at each site should be designated as SM." I really didn't know what SM meant and I immediately thought of sado-masochist. But, reading along, something about the context showed me that that was just not quite right. I said: "Oh no, you fool! That means system manager." But on further reflection maybe my first guess was not so far out of line.

See Figure 1 for a diagram of the Learning/Service Process of a System Manager.

A system manager uses information obtained from a great variety of sources, of which the most important are Users and Failures. He takes this information, processes it, using some of it directly, immediately, giving user benefit. Some of the information goes first to make him a better system manager, which in the end results in user benefit.

See Figure 2 for a diagram of the Responsibilities of the System Manager.

### Interest in the Users

The system manager has many responsibilities. One of the more important ones is to have interest in the users. For most users the computer is only a tool, it forms only part of the process needed to get their job done. The system manager should understand the users' needs and problems in a greater context. This will allow him to make the correct decision, sometimes by-passing computer difficulties to get the greater job done and out the door.

### Job Attitude

Ethics is an important responsibility. The system manager, having the most privileged account, must use his powers exclusively for service to the users, always respecting the privacy of others' data. The system manager's behavior can gently enforce among the employees a more professional attitude towards the computer and its capabilities, not a sloppy "everybody knows everybody's password" approach, not a naively trusting "we're all friends here" attitude.

Curiosity, that perpetual "Why?" and "How?" is a great asset, especially for troubleshooting.

What do you do when the buck stops at the desk of the system manager? You get up and you have an "I can do it!!!" attitude, which does not mean to proceed blindly but it also means not to evade the challenge and responsibility, responsibility which is part self-confidence and part carefulness. I failed in this respect. One day we were under pressure to meet a deadline for a report when the system printer failed. We had a procedure known to work for changing over a print queue to the operator's console when the printer was broken. The procedure failed for some unknown reason. Although my boss was out of town and in spite of my limited experience with this type of problem, I had indeed thought out the work-around. I would have been able to implement it, moving the queue to the second of two identical serial port boards that we have, but having to move the cables in a manner that was new to me, and possibly having a need to use a null modem. I balked, in part because of the difference of the cable maneuvers and doubts about the null modem, in part by a desire to get to the cause. I made the diagnosis of the cause far more important than getting my colleague's reports out on time. That was a mistake. I had lacked self-confidence in my work-around and I had permitted myself the luxury of "diddling around."

But I learned from that experience. The second time a situation like that came up was similar to the first: boss was out of town and I was all alone. My assignment was to load some new software. Following the manufacturer's instructions to the letter, the installation procedure failed. I first thought it was my mistake, so I went over to my account and made some experiments with symbols and logicals. I realized by this experiment and by reading the *Error Messages and Recovery* manual that I, not the manufacturer, knew the proper way to install the software. I then had to trace down the manufacturer's error. (I used the `SET VERIFY` command in his installation command file). I was then able to install the software. I called the firm and they acknowledged that yes, I was right and that they weren't too accurate in the use of DCL. I succeeded with the installation without having to have my users wait around while I located and talked to the guru. After all, I'm paid to know.

### Acquisition of Knowledge

The knowledge needed for the job is not always acquired in the "right" order. Sometimes you'll really be burning hot learning something and the next thing you know a user will carry you way off to another problem and you will be frantically tearing the pages out of the documentation set trying to find where the information is. The documentation set is really not a book you can read from cover to cover, but one should be acquainted with what's there.

### Transmission of Knowledge

My boss kept putting me up to many tasks, one of which was to organize users' meeting. I waited some time before starting them, mostly because I considered myself too inexperienced to offer any pertinent information or advice. At our site I decided to have a standard format of topics for these meetings: present status of the computer; current changes; near future changes; status of previous problems, if resolved or still in research; some type of presentation; and new problems and requests. I've scheduled these meetings every two weeks late Friday afternoon, so that our employees talk about something useful instead of their weekend outing plans. The result is turning out to be very positive, but if you try this out at your site, be sure to insist that the other computer users also make presentations.

The transmission of knowledge via memos and manuals brings back the story of the printer ribbon. I had written a procedure for changing the ribbon on the printer and had tried it out with three people at our site. Each of the three had a problem at a different place in the procedure. From that I learned a very interesting lesson: even simple English words that have nothing to do with the computer are not always interpreted in the same way by all people. It was a success though, as, months later, one of those employees remembered, without looking at the procedure, all of the details involved. In fact, she remembered it better than I did.

User education can be misinformation. I had an embarrassing experience when I, with great confidence, told a user that he could not delete his own directories. This was a very curious statement, but it stems from the fact that with the privileges of system manager, I always deleted my directories via the BYPASS privilege and I had completely forgotten about the `SET FILE/PROTECTION=(O:D)` command, whereby, of course, all users can delete their own directories.

## Tasks and Skills

In the next part of the paper are listings of topics, most of which will be encountered by all system managers. I cannot give the order in which one must learn things, as this is dependent on the person and on site needs. But these lists serve as starting points, checklists, overviews of the realm of the system manager. As a background process

in thinking about these topics, the system manager must keep system security always in his mind: if I do a certain command, if I make a certain change, physical or logical to the system, what changes might this bring to the security of the files?

### Software-related Tasks and Skills

Fluency with DCL is gained by experience. Not all of its wondrous effects are documented. The previous system manager had made some beautiful printouts of directory files and I wanted to imitate these. So I went over to the hard-copy terminal and I typed **TYPE DIRNAME.DIR**. Suddenly, the printhead went jerking all over the page. The paper just flew out of the machine, making a terrible racket. And there I was, helpless, as the paper tore and the machine continued trying to print. Those great interrupt commands CTRL-Y...nothing! I finally found, not the on/off switch, but the plug. I looked there sadly at that thing and then at the other's beautiful printouts and wondered what had I done? I thought "Oh Lord, I've broken the printer." It was the most embarrassing thing. Of course, you do not type or print .DIR or .EXE files, among certain others. **PRINT** is a qualifier to the **DIRECTORY** command, not **.DIR** a parameter of a PRINT or TYPE command.

Much can be learned by just skimming through the manuals, getting some idea of what's in them, what's available. This saved my life one night when I was working alone. We had changed the label on the second disk drive's pack. I tried to reboot the system and when the site-specific startup procedure was executing, it stumbled when it could not mount the second disk and it immediately exited from the procedure. System messages showed me that the installation was not complete; many commands did not work correctly, some not at all. I needed to edit the startup file, commenting out the second disk mount, in order to avoid a very lengthy process of taking the disk packs out, putting new ones in, purging the drives and getting the system back to the way it previously was. I gave the EDIT command and I got a very strange prompt. I knew from previous messages that the EDT editor had not been installed. To make things difficult, the INSTALL utility was one of those that did not work because of this false boot. But, from having skimmed through the manuals, I knew that there was something called nokeypad editing. So I took out the manual, read some instructions, edited the file and got home at a more decent hour.

I was very satisfied of my being able to quickly and accurately think this problem out. As one's computer knowledge matures, one's efforts are directed more rapidly to the correct answer. One's thinking—computer thinking—gets more refined. Now this does not mean that one understands everything, but it allows one to make intelligent guesses and reasonable predictions before issuing commands.

The catalyst for learning is not always business. My users had requested a more personalized login message and this request suddenly put into focus all sorts of things I had been reading about: lexical functions, symbols, symbol substitution, things like that. So I worked hard on this DCL program to give everybody something other than "Welcome to MicroVMS." (It was in the logic flow and economy of code of this program, where the experience with the small calculator was so useful). I learned so much from writing that program that I could then write programs really useful to the company. But the jokes continued. In one of the cuter ones I wrote, there comes flashing across the screen: "Two, four, six, eight, who do we appreciate???" Then in double height letters and in blinking, reverse video, (by which I learned to send control sequences to the terminal), appears: "SLC, SLC" (that's the name of my company), "Rah!, Rah!, Rah!" Well, those things are cute, but then you can really, really learn a lot by doing them and the users love them.

### Hardware-related and "Hands on" Tasks and Skills

The hardware-related and hands-on tasks are an area in which previous mechanical and electrical knowledge can be very useful. Be sure to observe your maintenance personnel when they come to work. For one, you get a view of your equipment on the inside and, two, you can help them avoid making dreadful mistakes. Be sure to learn the names of the parts so you can describe them when you need maintenance.

### Organizational Tasks

The organizational tasks can be much more important than they may seem. Our ability to purchase new equipment suddenly depended upon urgently producing a "wishlist." We were able to sweat one out in just four hours because I had all the advertisements filed in an orderly fashion by topic and/or device type.

The task itself of filing advertisements and organizing literature gives one an exposure to products and helps keep one up-to-date.

More important are the inventories of equipment, wiring and cabling diagrams and Computer Room Request forms, all of which might be included in a Computer System Standard Practices and Procedures document. This text should describe the facility and its operations to such detail as is needed to be able to operate the computer in the absence of the system manager. It might also include sections on disaster recovery and system security practices.

### Nice Luxuries

Many of the "nice luxuries" have to do with electricity. Much of this, and indeed, most of the topics herein mentioned, can be self-taught from numerous, readily-available books. If you don't know something, learn it!

Whereas knowledge of electricity can help one save time and money in trouble-shooting and in making custom cables, for example, other knowledge is more associated with the particular applications being run at your

site. Some will want to learn about mathematics and logic, others will need to understand spreadsheet software. Still others might want to study graphic arts, with a view towards good page layout and lettering, an aspect which can highly influence your customers.

## Conclusion

The yardstick for measuring personal performance as system manager is the service rendered to a user: "Anything for a user." And it's nice to know that one's knowledge does not have to be complete nor exhaustive in order to provide that good service. The variety of tasks, the challenge, the possibility to help others and the measure of personal growth make the job of system manager so much fun.

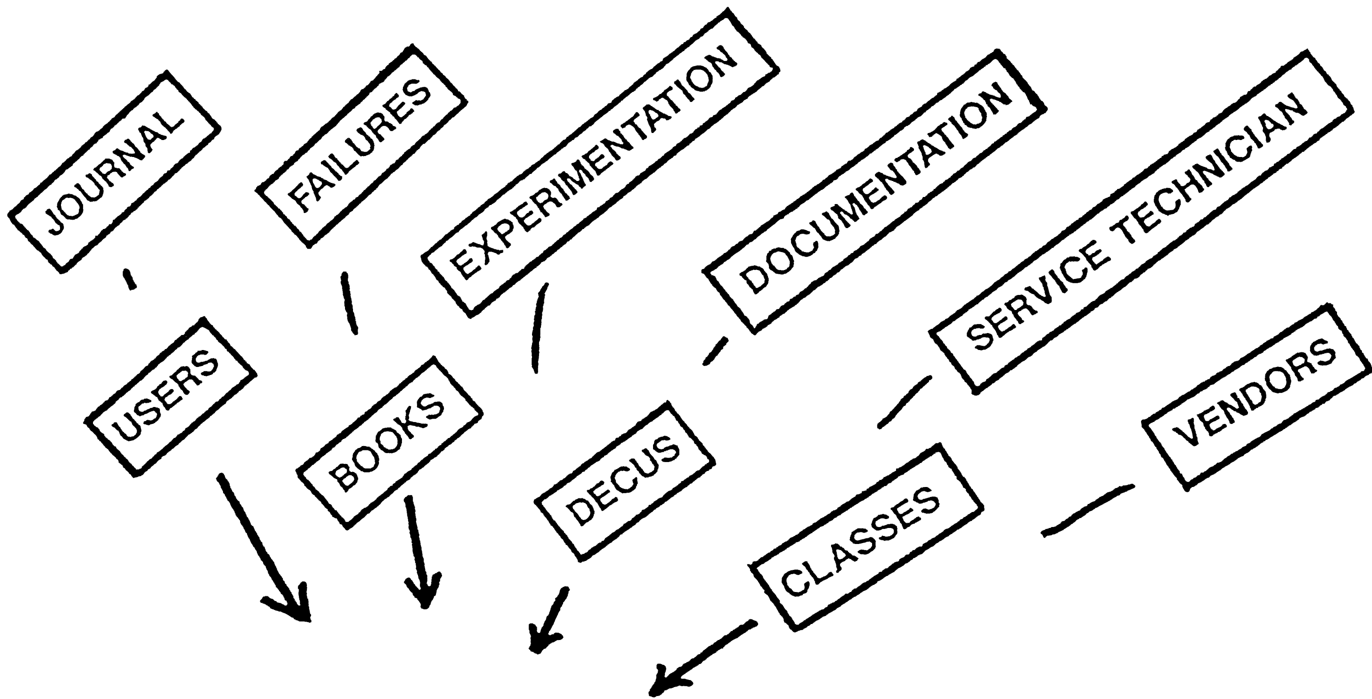
And in spite of any fears or problems, maintain the "I can do it!" spirit.

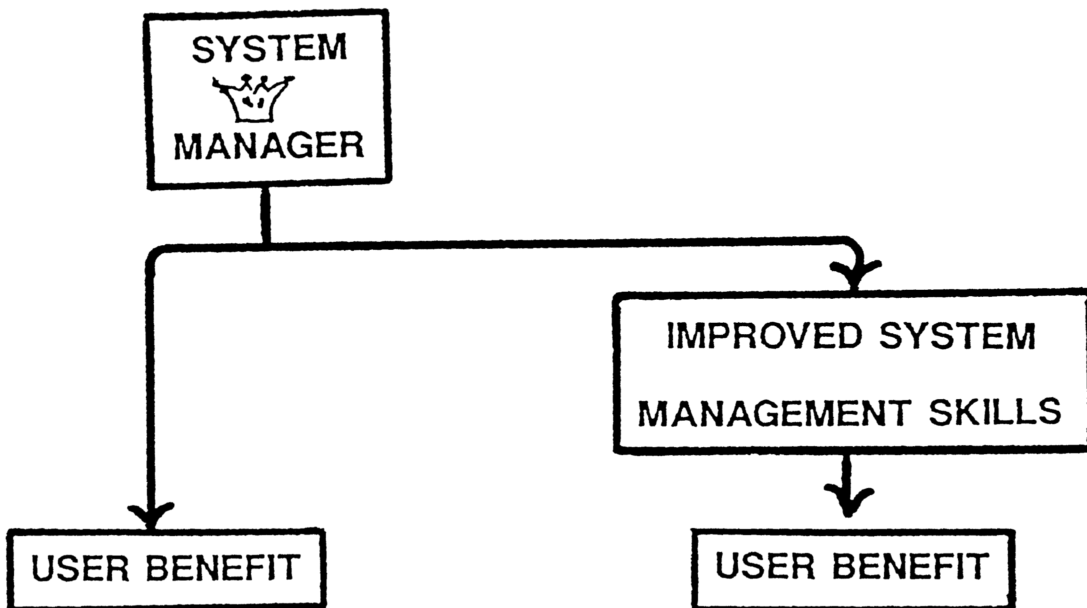
I did it, and so can you!!!

## Acknowledgments

I would like to acknowledge the two people who have most influenced the successful outcome of my venture into the computer world: my boss, Ted A. Jenkins, whose patience and wisdom have guided me through all my learning experiences, and my father, Ben F. Chartier, who, from my earliest childhood, inspired me to always give my best to the project.

# Learning/Service Process of a System Manager




* User education
* User environment tailored to meet individual needs
* Help in solving application/program problems

* More efficiency in handling user requests
* Automation of certain tasks
* Easy work environment lessens frustration and aids productivity

Figure 1.

# Responsibilities of a System Manager

**Interest in the Users**

- Understand the applications/programs which they are running.
- Understand the problems which they are facing.

**Job Attitude**

- Ethics
- Curiosity
- "I can do it!" spirit

**Acquisition of Knowledge**

- Keep abreast of new products and developments.
- Sharpen present skills.
- Better one's skills in weak areas.
- Learn completely new things.

**Transmission of Knowledge**

- User's meetings
- Memos and manuals
- User education
- Correct use of computer facility

**Application of Knowledge**

- Direct intervention in computer facility operations
- Contacts with maintenance companies, s/w and h/w technical support, supply houses, et al.
- Solutions to Users' problems

Figure 2.

# Software-related Tasks and Skills

(The basic command name is printed in bold capital letters).

**Fluency with DCL**

- Vocabulary
- Syntax
- General overview of VMS operation
- General overview of file organization
- General overview of user environment: UIC, UAF, username
- System startup and shutdown
- Interrupts: **CONTROL-Y**, **CONTROL-Z**, et al.
- **HELP**
- **SHOW**
- Directory listing: **DIRECTORY**
- Moving about in directory hierarchy: **SET DEFAULT**
- Creating files and directories: **CREATE**
- Use of editor in both line and keypad mode
- File protection: **SET FILE/PROTECTION**
- **TYPE** and **PRINT**
- Careful use of **DELETE** and **PURGE**
- Thoughtful use of wildcards: "*", "%" and "..."
- Communicating with users: **PHONE**, **MAIL**, **REPLY**
- Authorizing new users: **RUN AUTHORIZE**
- **MOUNT** and **BACKUP**

**Ability to Install New Software and Software Upgrades**

- VMS INSTALL Utility
- Formatting media
- Mounting media
- Temporarily changing a system parameter for installation procedure: editing file MODPARAMS.DAT then running AUTOGEN.

Figure 3.

Continued

**Security**

- UAF: **RUN AUTHORIZE**
- Privileges: **RUN AUTHORIZE**
- **SET PASSWORD**
- Removing users: **RUN AUTHORIZE**
- ACL: **SET** "Object"**/ACL**
- Audit trails: **SET AUDIT/ENABLE**

**Device set-up**

- **SHOW** "device"
- **SET** "device"

**Establishing and Managing Print Queues**

- In order to obtain different output formats.
- To permit multiple printers and locations
- **START/, STOP/, ABORT/**

**Development of DCL Command Procedures**

- Symbols and symbol substitution
- Lexical functions
- Logical tests
- Getting user input: **INQUIRE**
- Error handling
- How to call command procedures: "@"

**System Tuning**

- General: editing file MODPARAMS.DAT then running AUTOGEN
- User specific: modifications of user's UAF; **RUN AUTHORIZE**

**Accounting**

- Be able to generate accounting reports: **ACCOUNTING/**

Figure 3, cont.

# Hardware-related and "Hands on" Tasks and Skills

**Site installation**

- Layout of computer room
- Environmental considerations: temperature and humidity
- Alarms
- Layout of work areas
- Power conditioning and power cable routing
- Communications cable assembly and/or routing
- Installation of computer and mass-storage devices
- Installation of terminals, printers and other peripherals.

**Post-installation tasks**

- Routine upkeep and cleaning of site, hardware and peripherals
- Troubleshooting
- Hardware maintenance and repair
- Procurement, installation and storage of supplies: magnetic media, paper goods, printer/plotter materials, etc.
- Storage of recorded backup and software media

Figure 4.

# Organizational Tasks

- Documentation Library: organized and located for easy access by users.
- Literature and Journal Library: organized and located for easy access by users.
- Advertisement and Catalog File: organized by type of object or service; each item dated!!
- Backup media: clearly labelled; chronologically ordered; safely stored.
- Software media: clearly labelled; safely stored.
- Inventory of all hardware and software specifying their location and ID numbers.
- Write a Computer System Standard Practices and Procedures document.
- Write a Computer Center Request form.
- Maintain separate files for each device and software product. These will contain purchase, warranty, maintenance, etc. papers.
- Learn to type.

Figure 5.

# Nice Luxuries

**Knowledge of Electricity**

- Understanding of schematic symbols and conventions.
- Reading schematic diagrams (at least for cable routing).
- Component identification: type, physical appearance, units of measure, value/rating codes.
- Use of volt/ohm meter (for voltage and continuity checks).
- Cables and connectors: assembly and color-codes.
- Site power: breakers, phases and ground.
- Telephone/modem installations.
- Insulation and short-circuits.
- Basic understanding of semi-conductors.

**Knowledge of Mathematics**

- Binary notation and binary logic.
- Algorithms and formulae.

**Knowledge of Higher-order Computer Languages**

**Knowledge of Application Software**

- Word-processing
- Spreadsheet
- Desktop publishing
- Database management
- Graphics
- Accounting

**Knowledge of the English Language**

- Vocabulary and spelling
- Grammar

**Knowledge of Graphic Arts**

- Formatting
- Font types
- Optical illusions and layout

Figure 6.

# VAX SYSTEMS SIG

# Viruses, Worms, and Trojan Horses—Part II[1]

**Robert A. Clyde**
Clyde Digital Systems
Orem, Utah

### Abstract

Trojan horses, viruses and worms make an effective attack upon a system's security through the insertion of covert logic into otherwise innocent programs. Mandatory access controls under VMS provide a partial defense against them. However, mandatory access controls are not available on most VMS systems, and even when they are, Trojan horses, viruses, and worms can still pose a threat. Other defensive techniques are available which can combat Trojan horses, viruses, and worms.

Increased security awareness on the part of users and system managers will make users more cautious when loading and executing new software. The use of read-only memory precludes the insertion of a virus or worm into a program that is in read-only memory. Checking the integrity of programs on the system enables a site to detect inserted covert logic. Surveillance of system use is essential if a vigilant system manager or security administrator is to detect and possibly prosecute intrusion into the system. Appropriate analysis and reporting of the surveillance data greatly assists a site in detecting security problems.

## The Enemy

This paper focuses on attacks on a computer system by Trojan horses, viruses and worms. Viruses and worms are extensions of the Trojan horse theme. These attacks center around a seemingly innocent program that contains covert logic. When this program is invoked by an unsuspecting user, it tampers with objects that the user can access so that the intruder can achieve some ultimate goal. A good way to understand the relationship between Trojan horses, viruses and worms is to view them in the context of a concerted attack using all three.

A prospective intruder creates a Trojan horse—a program containing covert logic—and entices other people to use it. The Trojan horse's covert logic also contains a virus. When an unsuspecting user executes the Trojan horse, the virus spreads itself to other programs to which the user has write access [2,3]. Furthermore, each time one of the infected programs runs it also spreads the virus. Each virus also looks for an opportunity to insert a worm into a particular system program. This worm contains the necessary logic to allow the intruder to penetrate the system at will. For example, the worm might be placed in the LOGINOUT program so that whenever the intruder types a certain sequence for the password, he is logged in with all privileges.

[1]This is an updated version of the paper published by the same author in the *1987 Spring DECUS Proceedings* [1].

In summary, the relationships between these three enemies are as follows:

- Trojan Horse – Contains covert logic and can introduce a virus via unsuspecting users.
- Virus – Spreads the virus to other hosts (e.g., programs), thereby breaking down the defenses of the system so that eventually a worm will be inserted. The worm will be inserted when someone with write access to the targeted system program runs an infected program.
- Worm – Penetrates the security of the operating system.

Note the insidious nature of a Trojan horse attack on your system. Removing the worm that allows penetration will not provide a complete remedy since the virus will be active and will eventually insert another worm. Complete recovery from such an attack will require neutralizing the virus. The purpose of this paper is to discuss measures that could be taken in order to prevent and detect such attacks.

It is possible to have a Trojan horse without a virus and a virus without a worm. For example, if the goal of the intruder is simply access to certain files rather than penetration of the operating system, then a virus or Trojan horse alone will suffice. However, even though a virus spreads itself, it is not an inherently stronger attack than

a Trojan horse. This is because a Trojan horse can perform the same covert action that a virus does. Therefore, methods that will combat Trojan horses will be effective in combatting viruses and worms as well [4].

### Security Awareness

Increased security awareness on the part of system managers and users must be the initial focus for controlling this and most security problems. In particular, users and system managers should be properly suspicious when presented with a *gift horse* program. If there are any questions at all, the source code should be reviewed by a reliable expert.

Users should periodically check the protection codes and access control lists set on their files to see if there have been any changes. Generally, executable programs should be set so that there is no write access. This will help limit the spread of a virus.

Terminals must be locked up or logged out when not in use. If this is not done, an intruder can avoid the use of a Trojan horse and directly insert a virus or a worm via a terminal left logged in to someone else's account. For example, if a user with SYSPRV leaves his terminal logged in, an intruder could insert a worm directly into the LOGINOUT program from that terminal. In reality unattended logged in terminals probably account for more security breaches than Trojan horses. A terminal should be locked in cases where logging out would be difficult or very inconvenient. Although Digital's terminal server has a locking mechanism, VMS does not. Nevertheless, it is possible to write a program to perform terminal locking at the operating system level.

### Discretionary Access Controls

The standard VMS operating system comes equipped with discretionary access controls in the form of access control lists and protection codes that can be set on most objects. These access controls are known as *discretionary* since the owner of the object is able, at his *discretion*, to modify the access control lists and protection codes.

Because of this, discretionary access controls are not an effective defense against Trojan horses, viruses and worms. Consider the following Trojan horse scenario:

1. A user named JOHN writes a game called XTREK and sets its protection so that anyone can execute it.
2. JOHN places covert logic in XTREK so that it sets any files to which the user has access to world read and write.
3. JOHN then sends a mail message to everyone on the system proclaiming the wonders of his XTREK program and inviting all to try it.
4. SAM reads the message and decides to run XTREK. The game runs fine, but also changes SAM's files so that anyone can read or write to them. XTREK is able to do this since VMS allows the owner of the files to modify the access controls.
5. Now JOHN is able to read and write SAM's files.

### Mandatory Access Controls

Historically, mandatory access controls have been touted as the primary defense against Trojan horses, viruses and worms[6]. In practice, mandatory access controls are set in place by the system security officer—they cannot be modified by non-privileged users. If SAM owns files which have had mandatory access controls placed on them so that only TOP SECRET users can read or write to them, then SAM cannot lower the files' classification to allow users at the SECRET level to access them. So if SAM runs the XTREK program, it will be unable to lower the classification on TOP SECRET files. Thus, the mandatory access controls have defeated the Trojan horse in this example.

VMS does have a latent capability for providing mandatory access controls [5]. A separate product from Digital, the VMS Security Enhancement System (VMS SES), enables the mandatory access controls. This product is designed to raise the security of the system to the Department of Defense's B1 level [6], although the product has not yet been evaluated by the DoD's National Computer Security Center. Yet even with SES, mandatory access controls are not a cure-all for two reasons.

First, VMS with mandatory access controls contains numerous covert channels [6]. Covert channels are communication channels inherent in the system which were not originally designed as such. Covert channels are only significant on systems which have mandatory access controls and which are running multiple security levels. Therefore, a Trojan horse in a program run by a TOP SECRET user could use a covert channel to transmit TOP SECRET information to an intruder at the SECRET level. For example, if VMS allowed the SHOW USER command to be issued by users at any level, the process name field could be used as a communication channel. Thus the Trojan horse in the XTREK program could change its process name to contain TOP SECRET data. JOHN at the SECRET level could then read this data by issuing a SHOW USER command.

The method for handling the covert channel threat is as follows:

1. Identify as many of the covert channels as possible.
2. Remove as many covert channels as possible.
3. Monitor the remaining covert channels.

Second, mandatory access controls are not really mandatory for all users. On a VMS system there are privileges (e.g., BYPASS and READALL) which are able to bypass the mandatory access controls. When a program with a Trojan horse is executed by a user with those privileges,

it can access protected information and make it available to an intruder. Therefore, a complete penetration scenario involving viruses and worms, like the one described earlier involving LOGINOUT, would give the intruder all privileges—*including the BYPASS and SECURITY privileges.*

### Use of Read-Only Memory

Using Read-Only Memory (ROM) can be very effective in reducing the risk of a Trojan Horse attack [3], since data stored in ROM cannot be overwritten. Examples of possible ROM devices are

- ROM chips
- Optical disks (i.e., CD-ROM)
- Magnetic tape with the write ring or tab removed

ROM can be effective in the prevention and detection of Trojan horses when used in one or more of the following ways:

- Store executable programs
- Store data for comparison during integrity checking
- Store integrity checking routines for protection from tampering

Executing programs from ROM prevents anyone from inserting a Trojan horse into that program other than when the program is first placed in ROM. However, devices like optical disks can be much slower than more conventional media. When speed is important, it may be more cost effective to use the ROM for storing comparison information and integrity checking routines. This provides an unmodifiable standard for use in detecting covert logic that may have been placed in programs.

In order to truly use ROM to defend against a Trojan horse attack, however, a site must carefully certify each program before placing it into ROM. Otherwise, the site incurs the added expense of ROM without actually minimizing the chance of a Trojan horse attack.

### Controlled Program Creation

Another method for reducing a system's exposure to Trojan horses, viruses, and worms is to restrict the insertion and creation of programs on the system. This can be partly accomplished by acquiring software only from known, reliable sources. On particularly sensitive systems it may be necessary to perform a source code review and certification before placing an outside program on the system.

As an additional precaution, it may be necessary to restrict the creation of executable programs. This can be partially accomplished by controlling access to the various compilers, assemblers and linkers on the system. However, executable code could still be downloaded from a PC or some other system. By monitoring terminal input, it would be possible to detect downloading of executable code and flag this as a potentially suspicious event.

### Integrity Checking

Trojan horses, viruses and worms function by compromising the integrity of programs and files on the system. Consequently, their presence may often be detected by checking to see if any programs have been changed or if any file protections have been modified. Such an integrity check could consist of the following:

1. Compare the protection codes and ACLs of system files to a previously determined standard.
2. Check for viruses and worms by comparing system programs and files to a previously determined standard (i.e., perform a CRC).
3. Perform such a check upon each execution of a program image.
4. Store programs as encrypted images and then decrypt upon execution. This approach requires that the problems of encryption key storage and management be solved first.

While these procedures will detect that system integrity has been compromised, they do not provide sufficient information for identifying the source and method of the initial intrusion. Collecting such information requires surveillance.

### Surveillance of System Use

As indicated by the earlier sections, it may be difficult to totally prevent the insertion of a Trojan horse onto a system. Nevertheless, a vigilant system security officer can employ surveillance in an effort to discover one of the following conditions [7]:

- Original insertion of a Trojan horse, virus or worm.
- Abnormal use or access as a result of a Trojan horse, virus or worm making certain files or services available.

The following are potential sources of surveillance data:

- VMS security alarms.
- VMS accounting log.
- Monitoring terminal I/O.
- Monitoring system service use.

Only the first two sources are inherently available with VMS [6]. The other two would require additional system-level programming.

Once surveillance data has been collected, it must be analyzed. This can either be done manually or automatically. If it is done manually, the volume of data would most likely preclude any type of review other than spot checking. A computerized analysis, on the other hand, could greatly reduce the burden on the system security officer.

### Analysis of Surveillance Data

The purpose of collecting and analyzing surveillance data is to detect any type of suspicious activity—not just Trojan horses, viruses, and worms. Nevertheless, this method should also be effective against these particular intrusions into the system. (Of course, the surveillance system must have tamper resistant mechanisms of its own.)

For example, the VMS alarms could flag any use of the AUTHORIZE program. A security officer, knowing who is supposed to be able to run AUTHORIZE, could check these alarms and recognize if AUTHORIZE had been run by an intruder. (Note, however, that if the intruder acquired the SECURITY privilege, he could disable the alarms before running AUTHORIZE.)

Monitoring system service requests would make it possible to detect such things as

- Increases in privilege level
- Use of executive and kernel mode

The analysis program could search through the monitored terminal I/O and perform pattern matching in order to detect such things as follows:

- Browsing through directories
- Execution of AUTHORIZE, SYSGEN, INSTALL, etc.
- Displaying of sensitive information
- Downloading of executable code

Perhaps the most important benefit of monitoring terminal I/O is that it provides the system security officer with a complete record of what a particular user did at a terminal. Thus if a user entered a Trojan horse via the terminal, this action would be recorded. If a Trojan horse has made certain sensitive data available, the terminal surveillance would contain a record of what the intruder did with that data. Thus, the terminal surveillance data may constitute valuable evidence if disciplinary action or prosecution becomes necessary.

### Conclusion

Trojan horses, viruses and worms function by compromising the integrity of programs and files on the system. A concerted attack using all three can be particularly troublesome. Although mandatory access controls provide some defense against Trojan horses, viruses and worms, they may not provide a sufficient defense. Surveillance and integrity checking may be implemented on a VMS system with or without mandatory access controls. The use of surveillance coupled with integrity checking can provide a potent defense against Trojan horses, viruses and worms.

### References


[1] Clyde, R. "Defending Against Trojan Horses, Viruses, and Worms." *Proceedings of Digital Equipment Computer Users Society.* Nashville, TN: Spring 1987, pp. 381-386.

[2] Pozzo, M. and Gray, T. "Managińg Exposure to Potentially Malicious Programs." *Proceedings of the 9th National Computer Security Conference.*

[3] Young, C. "Taxonomy of Computer Virus Defense Mechanisms." *Proceedings of the 10th National Computer Security Conference.* Baltimore, MD: September 1987, pp. 220-225.

[4] Israel, H. "Computer Viruses: Myth or Reality?" *Proceedings of the 10th National Computer Security Conference.* Baltimore, MD: September 1987, pp. 226-230. Gaithersburg, MD: September 1986, pp. 75-80.

[5] Blotcky, S., Lynch, K. and Lipner, S. "SE/VMS: Implementing Security in VAX/VMS." *Proceedings of the 9th National Computer Security Conference.* Gaithersburg, MD: September 1986, pp. 47–54.

[6] U.S. Department of Defense. DoD Computer Security Center. *Department of Defense Trusted Computer System Evaluation Criteria*, CSC-STD-001-83 (Aug. 15, 1983).

[7] Stoll, C. "What do you Feed a Trojan Horse?" *Proceedings of the 10th National Computer Security Conference.* Baltimore, MD: September 1987, pp. 231-237.

[8] Digital Equipment Corporation. *Guide to VAX/VMS System Security.* Maynard, Massachusetts: July 1985.

# REMPRINT
## Remote Printing for VAX/VMS

Marty Adkins
Westinghouse Electric Corporation
Baltimore, Maryland

**ABSTRACT**

Although DECnet-VAX provides a rich set of features, Digital has yet to give the VAX user a simple way to print files on another node, while retaining all print qualifiers. Over the years, DECUS members have attempted elegant solutions in the form of distributed print symbionts, each with significant drawbacks. REMPRINT takes a simpler approach by implementing this capability (mostly) in DCL as a DECnet requester/server object.

### INTRODUCTION

As long as VMS and DECnet-VAX have existed, users have asked for the ability to print files to a printer on another DECnet node, **and** with all the relevant qualifiers of the PRINT command:

$ PRINT *filelist* /QUEUE=LASER -
/FORM=SPR /SETUP=*module1* -
**/NODE=*node***

VMS offers primitive remote printing in several steps:

$ COPY *filelist node*::
$ PRINT /REMOTE *node*::*filelist*
Wait until printed ...
$ DELETE *node*::*filelist*

Besides being quite tedious, this method does not permit any PRINT qualifiers to be specified. Also, the print job is charged to either the default account field of the proxy username, or to the default DECNET account, if no proxy exists. For sites with resource chargeback, this is not acceptable. An abbreviated form is easier,

$ COPY *filelist node::device:jobname*

but only works if the remote queue name is the same as the remote printer device name (LPA0 => LPA0:).

Over the years, a number of custom and modified print symbionts have attempted to solve this deficiency. Although these have been elegant solutions, they have had their drawbacks. Symbionts contain sophisticated code which may break with future VMS releases. Moreover, if your printer already uses a modified print symbiont supplied by DEC or a third party, there is currently no way to merge their functions. Examples include LATSYM for terminal servers, TFMSMB for Talaris laser printers, SPRINT for classification markings, and several Postscript conversion symbionts. Lastly, these distributed symbionts require that every queue be defined on every node, which is impractical in a large network.

### DEC RESPONDS

Digital has been listening, but with impaired hearing. At first glance, the newly-announced Distributed Queue Service appears to be what we have awaited for eight years. All relevant print qualifiers are supported, with automatic retry on the file transfer, plus remote queue manipulation. However, closer inspection shows that DQS also requires a queue to be defined on both the client and server nodes. Also, the remote print job accounting record has the account field set to the client node name, and the node name field set to blank, thereby defeating chargeback systems!

Digital has also ignored the many requests to bundle such a capability within DECnet-VAX. For this V1.0 release, DQS is a layered product, requiring a license (plus maintenance) on these nodes. Perhaps future releases of DQS will address these shortcomings, by using the Distributed Name Service, plus possible changes to the job controller.

## POOR MAN'S SOLUTION

REMPRINT uses the "KISS" method to achieve a reasonable compromise. Its syntax supports most print qualifiers, one notable exception being /NOTIFY:

> $ REMPRINT *node filelist* -
> [/ACCOUNT=*account*] [/NOWAIT] -
> [/*qualifiers* ...]

Filespecs may include full wildcarding. Remote queues need not be defined on the client node, and REMPRINT does not interfere with the symbiont of the remote target device. The /NOWAIT qualifier permits lengthy file transfers to be performed in a subprocess. The /ACCOUNT qualifier can be used to override the default of the current account.

Remote queue operations include display:

> $ REMPRINT *node* /SHOW_QUEUE

and job abortion:

> $ REMPRINT *node* /ABORT=*nnn queue*

## DCL TASK-TO-TASK

REMPRINT is implemented as a DCL DECnet client/server. Before examining its details, let's look at a simple example of using DCL for DECnet task-to-task functions (figure 1). The client expects to invoke a DECnet object on the remote node called SERVER, which is really just a command procedure called SERVER.COM. The server procedure opens the logical name SYS$NET to complete the DECnet logical link with its client.

Although this example employs two command procedures, the client and server functions could easily be combined into a single procedure:

```
$ IF F$MODE() .EQS. "NETWORK" -
      THEN GOTO Be-server
$!  Do Client part here
$!       ...
$!
$Be-server:
$!  Do Server part here
$!       ...
```

## REMPRINT IN DEPTH

REMPRINT exploits this feature of DCL and DECnet to accomplish its aim, but in a more complex fashion. Figure 2 depicts the simplified interaction between REMPRINT's two personalities.

Note that REMPRINT is defined as a DCL verb. ("What? You just said it was done with DCL!" Well, that's almost true.) The REMPRINT command has many qualifiers plus three syntax variants - show_queue, abort, and regular printing. Although it's possible to parse this in DCL, this author would never attempt it, especially when the CLI utility routines do it so nicely. The same reasoning applies to parsing file lists with wildcards. So REMPRINT.EXE parses the command line, and verifies file existence. It segregates the qualifiers into those that determine file selection, and those related to printing, and then builds COPY and PRINT commands into DCL symbols. The supported qualifiers are:

File selection:
/Since /Before /Created /Modified
/Backup /Expiration /Confirm
/By_owner /Exclude

Print attributes:
/After /Lower_case /Burst /Flag
/Trailer /Header /Space /Feed
/Form /Setup /Copies /Job_count
/Queue /Device /Passall /Note
/Characteristics /Parameters /Name

The /Delete qualifier is always appended so that the temporary files are deleted on the server node.

Now that all the preprocessing is done, REMPRINT.EXE exits by chaining to REMPRINT.COM. The first step is to open the logical link to the server, and to tell it which function we would like to perform - in this case, printing. The client passes along its username and account, and the server,

with the aid of a privileged image, validates the account, and sets its username and account to match. This will produce the proper accounting record for the print job, and cause the desired name to appear on the flag page. The server now creates a unique subdirectory to hold the files to be copied. The name, generated by the client, is the concatenation of the client's node name, username, and time of day. The server informs the client that it is ready to receive the files.

The client uses the previously formed COPY command to push the files to the remote subdirectory. The PRINT command follows, and is executed verbatim by the server. The resulting status message is deflected and displayed on the client's SYS$OUTPUT, usually the user's terminal. The server now restores its original username and account, and both parties close their logical link.

At this point, the user is back to a "$" prompt, and believes everything is finished. However, on the server side, some housekeeping remains. Although the temporary files will be deleted after printing, the subdirectory will not. The server is unable to delete it now because it contains files! So we compromise - by deleting all files that have resided here longer than 24 hours, which means today's invocation cleans up yesterday's litter. Other approaches were considered, such as a batch job that runs each night, but all had flaws.

## RESOURCE CHARGEBACK

A bit more should be said about NETJOB and our accounting system. At login time, a user must supply an account, or charge number, which will pay for the session. Not only must the account be valid, and open, but that user must be authorized to charge it. In VMS, print jobs inherit the account field of the submitter, and so should remote print jobs. Therefore, NETJOB modifies the account field of the surrogate submitter to produce the desired result. Before it does, it still verifies that, on the server node, the account is valid and open. Should the client node not be running our accounting system, or for some other reason, the client's account is rejected by the server, the /ACCOUNT qualifier may be used to specify a valid one. For this case only, NETJOB also insists that the user be authorized to charge the account on the server node. An image that can change one's username and account invites abuse, so NETJOB has several security checks. For example, it makes sure the job mode is "network", and that the DECnet object is REMPRINT.

## ERROR HANDLING

When writing any task-to-task application, the approach to error handling is simple - leave nothing to chance and trap **everything!** Use /TIME_OUT on READ to avoid hangs. If one party detects an error condition, it should inform its partner to permit a graceful cleanup. And if the connection to the server fails, check node reachability. This last item is harder than it sounds. A DCL OPEN statement that takes a /ERROR branch gives no error display, nor does it make the RMS status codes available. There is no DCL lexical function to test node reachability, so one's first inclination might be to redirect SHOW NET or NCP results to a scratch file. But beware - both give erroneous answers! These commands access the same NETACP volatile database, which is as stale as the last routing update. Moreover, if the remote node is in another DECnet area, then a level one router will always show it as reachable. Frankly, the only way to know for sure if a node is reachable at a given moment, is to "touch it". REMPRINT does this with a subroutine which is summarized below:

$ Assign *scratch-file* Sys$output, Sys$error
$ OPEN /Read *Node*"":: "0="
$ READ *scratch-file* looking for:
    "NOSUCHOBJ" - success
    "NOSUCHNODE" - typo
    "UNREACHABLE" - bashful
$ DELETE *scratch-file*

## PERFORMANCE FACTORS

To eliminate the large overhead of VMS process creation, permanent NETSERVER processes should be used. For the default DECNET username, these are the same ones used for Mail, Phone, etc. With that assumption, REMPRINT's transfer delay approximates the COPY time. Small files will typically transfer in 3-5 seconds for DECnet links of 56Kbps or greater. The DCL COPY utility performs block-mode transfers, so more elegant solutions should attain minimal improvement.

## INSTALLATION

REMPRINT's installation consists of two steps. On the client node, REMPRINT.CLD must be interpreted by SET COMMAND and stored in DCLTABLES, with the image specification pointing to the location of REMPRINT.EXE. On the server node, REMPRINT.COM must be defined as a DECnet object:

```
NCP> Define Object REMPRINT -
        Number 0 -
        File Loc:REMPRINT -
        Proxy Outgoing -
        User user Password pass -
        Account account
```

Lastly, NETJOB must be Installed with CMKRNL privilege to modify the username and account, and with SYSPRV to access our accounting authorization files:

```
INSTALL> Loc:NETJOB /Open /Header
          /Priv=(Sysprv,Cmkrnl)
```

## CAVEATS

Every poor man's solution has its limitations, and REMPRINT is no exception:

1) The DCL READ /TIME_OUT qualifier does not currently work for logical links due to a deficiency in the VMS mailbox driver. This has been SPR'd to Digital.

2) REMPRINT does not provide a way for other applications to spool files to a remote queue, since it creates no local server queue.

3) REMPRINT does not currently support Unix/Ultrix systems.

| CLIENT | SERVER |
| --- | --- |
| $ OPEN /Read /Write /Error=No-server -<br>DCLserver *Node*::"Task=SERVER" | $ OPEN /Read /Write /Error=False-alarm -<br>DCLclient SYS$NET |
| | $ DEFINE SYS$OUTPUT DCLclient<br>$ DEFINE SYS$ERROR DCLclient |
| $ WRITE DCLserver /Error=Disappeared -<br>"SHOW USERS" | $ READ DCLclient /Error=Goodbye -<br>/Time-out=30 COMMAND |
| $! Result displayed on SYS$OUTPUT | $ 'COMMAND |
| $ CLOSE DCLserver /Error=Continue | $ CLOSE DCLclient /Error=Continue |

Figure 1: DCL for DECnet Task-to-Task

| CLIENT | SERVER |
| --- | --- |
| REMPRINT.EXE - Parse command line,<br>verify file existence, and store COPY<br>and PRINT commands in DCL symbols | |
| $ OPEN Server | $ OPEN Client |
| $ WRITE "PRINT" | $ READ Option |
| $ WRITE Id, Username, Account | $ READ Id, Username, Account<br>NETJOB.EXE - Set Username, Account |
| | $ CREATE /DIRECTORY -<br>[.PRINT.'ID'] /PROT=O:RWED |
| $ READ State | $ WRITE "READY" |
| $ COPY *filelist node*::[.PRINT.'ID'] -<br>/PROT=(S:RD,O:RD,G,W) | |
| $ WRITE Print-command<br>"Job 99 started on queue LASER ..." | $ READ Print-command<br>$ 'Print_command |
| | NETJOB.EXE - Restore Username, Account |
| $ READ State<br>$ CLOSE Server | $ WRITE "DONE"<br>$ CLOSE Client |
| | $ DELETE /MODIFY /BEFORE="-1-" -<br>[.PRINT...]*.*;* |

Figure 2: REMPRINT Client/Server Interaction

# SOFTQUOTA
## A Diskspace Management Utility

Shari Dishop
VAX Support Group
Westinghouse Electric Corporation
Baltimore, Maryland 21203

Abstract

Since the introduction of disk quotas in VMS 2.0, the VAX system manager has been empowered to manage disk storage. Unfortunately, the implementation of "overdraft" is inflexible, and does not track the typical development scenario. During the life of a process (or login session), a user will likely create a number of sizable, temporary files through the actions of editing, compiling, linking, and executing programs. At the end of the session, most of these files are purged, printed and deleted, or rolled out to magnetic tape. SOFTQUOTA adds a third threshold to the quota system, a "soft" quota. This utility is invoked at login time to check a UIC's permanent disk usage against its soft quota, while still permitting growth to the regular hard quota during the session. Parameters for each UIC (or identifier) are maintained on a per-volume basis, and may be displayed or modified with the SOFTDB utility. In a VAXcluster system, coordination between nodes is done via the distributed lock manager.

Note: This is a repeat of a paper presented at the Spring 83 DECUS. It has been updated to reflect changes since that time.

### Introduction

In VMS release 2.0, Digital introduced the disk quota mechanism and in VMS 4.0 they introduced ownership of files by identifiers in addition to by UIC's. This disk quota mechanism permits the system manager to establish an absolute limit on disk storage for each user (UIC or identifier) on a per-volume basis. The quota for a user is specified by two parameters:

| | | |
|---|---|---|
| PERMANENT quota | - | ceiling on total usage |
| OVERDRAFT quota | - | margin above PERMANENT quota to extend an already open file |

The USAGE value for a user is constantly updated at every file creation, deletion, and extension. If the requested allocation would increase USAGE above quota, the operation is not performed, and the user receives the familiar message:

```
%SYSTEM-F-EXDISKQUOTA, disk quota exceeded
```

Unfortunately, the implementation of "overdraft" is inflexible, and does not track the typical software development scenario. During the life of a process (or login session), a user will likely create a number of sizable, temporary files through the actions of editing, compiling, linking, and executing programs. At the end of the session, most of these files are purged, printed and deleted, or rolled out to magnetic tape (or at least they should be!). To augment the standard permanent or "hard" quota, we established a "soft" quota with the following relationship:

| | | |
|---|---|---|
| SOFT quota | = | static disk storage |
| PERMANENT quota | = | static disk storage + workspace area |

So as not to interfere with Digital's implementation, we created a separate database to contain the additional parameters, and a SOFTQUOTA utility which is invoked for the user at login to:

1. Display current usage statistics
2. Optionally lower hard quota to equal soft quota
3. Restore original hard quota if soft quota no longer exceeded

A SOFTDB utility was also developed for ease in maintenance and display of the

database.

## SOFTQUOTA UTILITY

The SOFTQUOTA utility is defined as a verb with the following syntax:

```
$ SOFTQUOTA   [/MODIFY]     [/QUIET]
              [/NOUNIQUE]   [/START]
              [dcnn,dcnn,...]
```

| | | |
|---|---|---|
| dcnn | - | Volume to check (defaults to SYS$DISK) |
| /MODIFY | - | Allows modification of hard quota |
| /QUIET | - | Suppresses informational messages (e.g., "quota file not active") |
| /NOUNIQUE | - | Don't check for unique UIC |
| /START | - | Start the listener for remote nodes on clusters |

Examples are:

```
$ SOFTQUOTA
You have used 3488 blocks of 5000 block softquota on
USER$DISK

$ SOFTQUOTA DBA1,DBA2
%SOFTQ-I-QFNOTACT, quota file not active on DBA1
You have overdrawn 224 blocks of 3000 block softquota
on DBA2

$ SOFTQUOTA /MODIFY/QUIET DBA1,DBA2
You have overdrawn 224 blocks of 3000 block softquota
on DBA2
You will not be able to create new files until you have
cleaned up.

        ... User cleans up on DBA2 ...
    User must invoke softquota to reset quotas

$ SOFTQUOTA
You have used 2634 blocks of 3000 block softquota on
DBA2
```

### When to run it

At our sites, SOFTQUOTA is invoked at login time in the system-wide command procedure SYS$SYLOGIN.

```
$ IF F$MODE() .NES. "INTERACTIVE" THEN -
$ GOTO DONE
$ SOFTQUOTA /MODIFY/QUIET
$DONE:
```

We do not run it at logoff time because:

1. The user should never be inhibited from logging off.
2. Logoff procedures are still easily circumvented.
3. It interferes with processes of same UIC.

The third item deserves more discussion. Each interactive login permits SOFTQUOTA to conditionally "lower the boom". If User1 has a batch job running which has exceeded the soft quota for that UIC, and User1 then logs in, SOFTQUOTA will lower his hard quota, which may cause problems for the batch job. Similarly, if User2 with the same UIC logs in, he will cause problems for User1. This is why checks are performed in the program to determine if there is another active process with the same UIC. As long as NOUNIQUE is not specified, SOFTQUOTA will detect possible interference and will display a status message, but will not modify the user's hard quota. The same action is taken for batch logins; if the user is not around to correct the problem, there is little to be gained by impacting his batch job.

### SOFTQUOTA Databases

If quotas are enabled on a disk volume, the parameters for each user are stored in the file [000000]QUOTA.SYS, which is managed by the ACP. To prevent performance degradation, a portion of the entries are cached in main memory. To examine or modify an entry, SOFTQUOTA issues ACP QIO requests specifying a function code of IO$_ACPCONTROL and parameter lists for the File Information Block (FIB), and the File Transfer Block (FTB). The SOFTQUOTA data items are maintained on a per-volume basis in the file [000000]SOFTQUOTA.DAT, which is indexed by UIC. The record definition is:

Softquota Record Structure

| | | |
|---|---|---|
| UIC | I4 | Primary Key 0 |
| Group | | Key 2 |
| Member | | Key 1 |
| SOFTQUOTA | I4 | Amount of residual space allowed |
| HARD-SAVE | I4 | Saved PERM quota from QUOTA.SYS |
| MESSAGE-COUNT | I4 | Number of times user has exceeded softquota |

### SOFTQUOTA General Algorithm

The following structured English describes the SOFTQUOTA algorithm in a slightly simplified form:

```
if USAGE .LE. SOFTQUOTA then
  Display "usage" message
  if HARD-SAVE .NE. 0 then  {User cleaned up}
    PERMQUOTA := HARD-SAVE
    HARD-SAVE := 0
else if PERMQUOTA .LE. SOFTQUOTA then
  Display "Still Overdrawn" message
  MSG-COUNT := MSG-COUNT + 1
else
  Display "Overdrawn" message
```

```
if "/MODIFY" specified AND "unique UIC" then
   HARD-SAVE := PERMQUOTA
   PERMQUOTA := SOFTQUOTA
   MESSAGE-COUNT := MESSAGE-COUNT + 1
   Display "Must Cleanup" message
```

Error Handling

SOFTQUOTA checks the status returns from all system calls. Certain errors are handled gracefully, and display a message to the user. Other messages are displayed only if the "/QUIET" qualifier is not specified.

Quotas not enabled -- if "/QUIET" not specified, display %SOFTQ-I-QFNOTACT, quota file not active on <volume>

No entry in QUOTA.SYS -- if "/QUIET" not specified, display %SOFTQ-W-NODISKQUOTA, no HARD disk quota entry on <volume>

No entry in SOFTQUOTA.DAT -- use DISK DEFAULT %SOFTQ-W-DEFSET, no SOFTQUOTA entry on <volume>, using default

Record locked in SOFTQUOTA.DAT -- retry 50 times

All other errors are fatal and a rather noticeable error exit is taken:

```
25-Mar-1983 11:05:34.10

        IMPORTANT:

Please tell SYSMGR that the following error occurred in
SOFTQUOTA
 Thank you.

%SOFTQ-F-QIOERR, initial QIOW failed
%SYSTEM-F-DEVOFFLINE, device is offline
```

Clusters: A Special Challenge

A problem was discovered with the Softquota utility upon the introduction of clustered machines in VMS 4.0. Although the database files are shared among the nodes of a cluster, each machine is only aware of the processes local to it. The check for a unique process no longer worked. Users quickly complained about having a job running on one node in a cluster, where it had accumulated substantial amounts of CPU time and temporary disk space, then logging on to another node in the cluster and having their first job killed. It was discovered that when they logged on to the second node they were informed by softquota that they were over their quota and since they had a unique process on that node their permanent quota was lowered. This caused the process on the first node to be killed for violating their quota. Thus all of the time spent by the first process was wasted.

A solution to this problem was needed and it required very fast inter-processor communication. It should not take longer to log on to a cluster than to log on to a non-clustered machine. The solution also had to have an extremely clean way to synchronize the inter-processor communication. Timing races between machines are no fun!

We chose to use the distributed lock manager as a way to implement our solution. It provides the fast inter-processor communication desired and has a 16 byte block (lock value block) for passing data between processes. It also provides a method of synchronization to avoid the timing races.

A Look at the Distributed Lock Manager

The lock manager is a means of controlling shared access to resources (files, data structures, databases, executable routines, etc.). A request to access a resource is called a lock and has a level of access and sharability or mode associated with it. These modes are:

NL Null Mode
- grants no access, indicates interest in the resource

CR Concurrent Read
- grants read access, allows others to read or write

CW Concurrent Write
- grants write access, allows others to read or write

PR Protected Read
- grants read access, allows others to read but not to write

PW Protected Write
- grants write access, allows others concurrent read access but no write access

EX Exclusive
- grants write access, prevents others from accessing the resource

The compatability of the lock modes is summarized in the table following.

Compatability of Lock Modes

| Requested Locks | Granted Locks | | | | | |
|---|---|---|---|---|---|---|
| | NL | CR | CW | PR | PW | EX |
| NL | Y | Y | Y | Y | Y | Y |
| CR | Y | Y | Y | Y | Y | N |
| CW | Y | Y | Y | N | N | N |
| PR | Y | Y | N | Y | N | N |
| PW | Y | Y | N | N | N | N |
| EX | Y | N | N | N | N | N |

A lock can be in one of three states: it can be in the granted state, it can be waiting in the queue to be granted, or it can be waiting in the queue to be converted to a different mode. Lock conversion is used to provide the synchronization of the inter-processor communications. Converting a lock to a higher mode that is incompatible with an existing lock will cause the first lock to be placed on the conversion queue until the second lock is dequeued or is converted to a compatible mode. This can be used to cause one process to wait for another process to complete a desired task. Lock conversion can also cause the contents of the lock value block to be read from the master copy or to be written to the master copy. The table below summarizes this aspect.

Effect of Lock Conversion on Lock Value Block

| Held Mode | Mode Converted to | | | | | |
|---|---|---|---|---|---|---|
| | NL | CR | CW | PR | PW | EX |
| NL | R | R | R | R | R | R |
| CR | N | R | R | R | R | R |
| CW | N | N | R | R | R | R |
| PR | N | N | N | R | R | R |
| PW | W | W | W | W | W | R |
| EX | W | W | W | W | W | W |

Unique User Algorithm

The following is the algorithm used to determine if a user has a process on another node in the cluster.

```
Look in local process table for same UIC
If match found then
 return non-unique user status
 exit algorithm
Else
 If member of a cluster then
  For each node in the cluster loop
   If not the local node then
    Issue lock on resource to awaken remote node
     Remote awakened by blocking AST on resource
     Remote reads UIC from lock value block
     Remote looks in its process table for match
     Remote puts match value in lock value block
     Remote converts lock to release resource
     Remote hibernates till next request
    If error occurs in communication then
     Continue
   Else
    Read lock value block
    If match found then
     return non-unique user status
     exit algorithm
```

New Version

Some advantages to this new version of the softquota utility are that the same software runs on a single node or on a cluster and in a cluster environment it dynamically adjust to nodes entering or leaving the cluster. But one drawback is that this current version of the utility requires a continuously running batch job on cluster nodes to provide fast execution time for processing unique user check requests.

## SOFTDB UTILITY

Just as the DISKQUOTA utility is used to maintain the hard quota file, some method is required to edit the SOFTQUOTA indexed file. In the early days of SOFTQUOTA, we utilized an existing in-house forms/update tool which supported only character data fields; hence, SOFTQUOTA.DAT was a formatted character file. Within the past year, the file has been converted to a binary format and the SOFTDB utility has been enhanced and expanded. The SOFTDB utility will now report the information in the QUOTA.SYS file, and will also handle adding and modifying entries in the QUOTA.SYS file. Its command set resembles the familiar syntax of the AUTHORIZE utility.

SOFTDB Command Set:

```
USE       <volume> - Defaults to SYS$DISK
ADD       <UIC>     [/SOFTQUOTA=n]
                    [/HARDSAVE=n]
                    [/MESSAGE=n]
                    [/PERMQUOTA=n]
                    [/OVERDRAFT=n]
MODIFY    <UIC>     [/SOFTQUOTA=n]
                    [/HARDSAVE=n]
                    [MESSAGE=n]
                    [PERMQUOTA=n]
                    [/OVERDRAFT=n]
REMOVE <UIC>
SHOW [/USEHARD] <UIC> - Paginated display.
LIST [/USEHARD] <UIC> - lists specified
                        record(s) on
                        FOR$PRINT
                        (defaults to
                        SOFTQUOTA.LIS)
HELP <Keyword1> <Keyword2> etc.
@<filespec> - command file, default type is
              ".COM". If logical name
              SOFTDBINI is defined, that file is
              executed at startup.
```

EXIT

<UIC> - [g,m], [*,m], [g,*], [*,*], identifier, USER, or DISK

/USEHARD uses the QUOTA.SYS file as the reference file on wildcard operations.

## CURRENT LIMITATIONS & PLANNED ENHANCEMENTS

Currently the remote process for clustered systems only runs from a batch job. We are planning to make it run as a detached process to remove it from the batch queue. Currently the SOFTDB utility will accept identifiers and add entries for them to the database but the SOFTQUOTA utility only checks a user's UIC when they log in. We wish to also check the quotas of all of the identifiers a user holds. In the SOFTDB utility the REMOVE function does not support access to the QUOTA.SYS file. We have not determined whether to remove a quota for a UIC that still owns files or not.

## SUMMARY

No quota system ever won a popularity contest with users, and SOFTQUOTA is no exception. At our sites, it is accepted as one of life's necessary evils, and is considerably less offensive than ordinary hard disk quotas. We have been using this tool since 1981, and unless the cost of disk drives plummets, or VMS provides an equivalent capability, SOFTQUOTA should enjoy a long life.

Keywords: VAX/VMS, DISK QUOTA, SOFTQUOTA, SOFTDB

Acknowledgments: The author would like to credit Martin J. Adkins for presenting the original paper in 1983 and additional help in designing the solution for the clusters. The author would also like to credit Art Moorshead for developing the SOFTDB utility. SOFTQUOTA was a product of design by committee, with the initial implementation by Brad Schafer (now with DEC). Robert A. Koppelman added support for multiple volumes, and ancillary routines and assistance were provided by Almon T. Sorrell. This author implemented the cluster support for checking for unique UIC's and the latest enhancements to the SOFTDB utility.

# FAST RESPONSE ON OVERLOADED SYSTEMS
## (or the alchemy of the VMS scheduler)

**Silvano de Gennaro**
European Organization for Particle Physics Research (CERN)
Geneva, Switzerland

**Abstract**

This report describes the features and the ...non-features (euphemism) of the VMS scheduler, and presents the analysis work we have done at CERN, trying to make scheduling more effective, particularly in situations of CPU saturation. By modifying the logic of the scheduler we could obtain remarkable results in interactive response time under conditions of computing overload.

The purpose of this presentation is to show you how we managed to obtain a fast interactive response out of our overloaded VAX computer systems.

By "Fast Interactive response", I mean the response you typically get from an empty system, and by an overloaded VAX I mean a well tuned and reasonably well configured one, running at 100% of its CPU capacity.

A VAX is first of all an interactive machine, and therefore it should guarantee a constant response time to people who want to use it as such.

Users who are very interactive will use such a small portion of the CPU, that it is logical on a timesharing system to give them easier access to it.

Typically you may think of a full screen editor as a very interactive application, where you spend most of your time moving through the file with the cursor. It seems unreasonable that a cursor movement or a page scroll must wait for someone else to finish computing the 17th root of 3454. But unfortunately this happens, because of the way the scheduler works.

In fact the design of the VMS scheduler is full of good intentions, and uses a mechanism of dynamic priorities which tries to reward terminal bound people as opposed to CPU bound ones.

The implementation was simple and proved sufficient when VAX computers were minicomputers, but now it can no longer cope with the complexity of the production environment supported by the new large systems.

Our analysis led us to conclude that the problem was in the handling of the dynamic priority, so finally we developed, in our Alchemy lab, a witchcraft that turned a saturated VAX into a responsive machine.

Before introducing you to the mysteries of our alchemy and revealing the magic formula of instant response time, I must initiate you to the infernal rites of the VMS scheduler.

The unitary entity considered by the Scheduler is a Process. The process parameters that influence the Scheduler decisions are the Process state and priority. Process state and priority change according to process behavior and with the occurrence of System Events.

The process running in the CPU is called the Current (CUR) process. There is at any one time only one Current process per processor. To be eligible to become Current, a process must be Computable (COM), which means waiting for the CPU. If it is not Current, nor Computable, a process can be in a wait state: Suspended (SUSP), Hibernating (HIB), Local Event Flag wait (LEF), Common Event Flag wait (CEF), or generic Resource Wait (RWxxx).

A process in COM, LEF, HIB or SUSP may be outswapped if the system is short of free memory, in which case its state is changed to COMO, LEFO, HIBO, SUSPO respectively.

Apart from being in COM state, to get the CPU a process must also have the highest priority in the system. In fact the VMS scheduler always picks up the first COM process at the highest priority. If there are more processes with the same priority, then they are queued together in the priority ring relative to that priority. The scheduler takes jobs from the top of their priority ring, and puts them back at the bottom.

Normally a process executes at its base priority, which is fixed and assigned at process creation.

On top of this Base priority, a process scores an extra grant, called "Priority Boost", for every I/O operation completion it experiences.

Once boosted up, the priority will start decaying back to the Base amount via decrements that take place at every CPU reassignment.

In standard VMS the values assigned to the priority boosts are: 6 for terminal input, 4 for terminal output, 3 for resource available, and 2 for disk I/O completion. These boosts are added to the base priority and are

not cumulative. So, for instance, a process with a base priority of 4 will be boosted to 10 after a Terminal Input completion, but will be executing at 9 next time it gets the CPU, because the Scheduler decrements by 1 at CPU reassignment time.

When it gets the CPU, a process may keep it for the duration of a QUANTUM, (SYSGEN parameter; by default 200ms on all machines!) unless it issues a resource request or is pre-empted by another process which becomes Computable at a higher priority.

The drawback of pre-emption is that the pre-empted process not only loses the CPU, but it also loses priority, as pre-emption causes a rescheduling, therefore the Scheduler will subtract 1 next time it reassigns the CPU to the pre-empted process, which also ends up at the tail of the ring for this lower priority level.

So in a busy system, where the hammering of pre-emption is high, your priority boosts may vanish very rapidly. And of course, the lower your priority gets, the faster it decays towards your base priority. And what is really dramatic here is that the speed of your priority decay is independent from your sins or virtues, and becomes basically random.

And what did you do to be punished so crudely?

Nothing. You were just unlucky to bump into someone stronger than you. This fundamental law is called "The law of the jungle", and the method is called "Wild scheduling".

Apart from this basic mechanism of priority adjustment, the VMS scheduler has three joker cards, which make it even more unpredictable.

The first important joker is what we call the PIXSCAN boost.

What is given to common mortals to know is that every second the system scans through the COMputable processes and boosts one or more of those to the same priority as the highest non-realtime computable process in the system. The reason to do this is to prevent possible deadlocks caused by processes at a low priority keeping locks for a long time.

The parameter PIXSCAN defines the number of Process blocks to scan. This parameter plays an important role in system performance. In fact processes which are most of their time COM are usually compute-bound low priority ones (most likely batch). So, if PIXSCAN is too high, you risk to give them too easy access to the CPU, with bad results for interactive performance.

A typical side effect of this is that if you have privileges and you need to raise your priority up for a legitimate reason, like investigating a system problem, or getting a decent response out of PACMAN or STARWARS, then be aware that you will bring up together with you a bunch of people at random, including batch jobs every second.

The second joker in the list is the special parameter IOTA.

A process normally runs for a CPU QUANTUM. But most I/O operations require so little CPU that an I/O bound process may take too long to reach its Quantum End. So, because some important actions like Working Set Adjustment happen at Quantum End, the parameter IOTA was introduced to cut the quantum shorter for I/O bound processes. The value of IOTA (def. 20 ms) is subtracted from the quantum time left for every I/O operation.

The third joker is the fact that all this nice setup goes completely berserk if there is one or more processes in COMO state. In this case everyone will be taken down to his base priority at every Quantum End.

The reason for that much violence is to give a chance to the COMO guy to get back in memory sometime. This may be correct in a machine which is totally interactive, but causes incredible degradation in a wide job mix which includes batch. It is absolutely unreasonable to slow down 100 people editing just to swap back in a 12 hours computing batch elephant.

To resume what we said so far:

A priority boost is linked to events that depend on process or system context (I/O completion, PIXSCAN). Priority decrement instead goes along with CPU assignment, which in busy systems becomes random due to a high pre-emption rate. In these conditions, the priority decay speed becomes uncontrollable. This narrows the distance between CPU and I/O intensive jobs, lining them all back to their base priority too soon.

So, to make scheduling more effective and controllable by the system manager, we tried to remove some of these random factors.

In fact, the major effort for us was not in modifying, but in understanding the dynamics of things; what was going on exactly. The documentation tells you almost nothing about it, and there are no utilities in the sky or on earth that can help you monitor the reactions of the VMS scheduler to different environments and parameters settings.

VMS performance manuals kindly suggest to "acquire more CPU capacity" if the CPU becomes saturated. That in reality would not be necessary with a better scheduler.

Other major operating systems, that we have experience of, don't kill interactive performance so drastically when the CPU is hogged, and the reason is that they have a deterministic scheduler, not a random wild dog.

To understand and solve the problem we set up a team composed essentially of wizards with a long background of experience on different operating systems for large scale production.

We started writing some simple utilities to analyze priority changes and measure their effects.

We finally found the solution to the VMS scheduler problem in an ancient book of black magic written by Hermes Trismegistos, who was the father of modern Alchemy. In fact although Alchemy and Black Magic have nothing to do with each other, the amount of fantasy and blind faith which existed in both does bear a remarkable resemblance to the techniques commonly used

by computer manufacturers when producing operating systems.

Therefore like all good alchemists we attacked the problem using the method of syllogism.

Syllogism is a demonstration method invented by Aristotle, by which you can prove that a predicate is true if obtained by inference from two true hypothesis.

All animals are mortal
Man is an animal
Man is mortal.

The first two statements are true, so the resulting one is true as well.

All stones are made of elements
Gold is an element
All stones are made of gold.

In fact, the problem of turning stones into gold, and the one of getting a good response time out of an overloaded VAX are very similar, with the difference that stones to gold is easier to do.

So we needed a slightly more powerful syllogism, and therefore we used this multi-threaded, para-inferential, phylo-exoteric syllogism of the 3rd kind:

A decrement is a punishment.
A punishment is to a crime.
A crime is abuse of CPU.
CPU is given in quantums.
A crime is abuse of quantums.
A decrement is for a quantum.

In other words, the more quantums you use, the more you are CPU intensive, therefore criminal, therefore the scheduler must punish you by decreasing your priority.

It is not a crime to lose the CPU when you are pre-empted, but it is a crime to ask for it again and again at every end of quantum. So we took our magic wand and turned the random frog into a deterministic prince, by decreasing the process priority at Quantum End instead of at CPU assignment.

This makes the priority decay much more controllable through SYSGEN parameters, and makes it easy for the system manager to enhance the difference in decay speed between CPU and I/O bound processes, through the length of Quantum and the setting of IOTA.

To measure the results of our transmutation we performed a "margin conditions" benchmark.

**Figure 1: Benchmark**

| | |
|---|---|
| PITAGORA | 100% cpu |
| ARCHIMEDES | 90% cpu + disk |
| EUCLID | 90% cpu + t/out |
| HERACLIT | 90% cpu + t/in |
| DIOGENES | 50% cpu + disk |
| ARISTOTELES | 50% cpu + t/out |
| PLATO | 50% cpu + t/in |
| SOPHOCLES | 10% cpu + disk |
| EURIPIDES | 10% cpu + t/out |
| DEMOSTENES | 10% cpu + t/in |

We used 10 jobs executing concurrently, and requiring a fixed percentage of CPU, plus some sort of I/O (Fig. 1). These jobs were run all at the same time, in an empty VAX 780 with default SYSGEN settings, and observed running for 5 minutes by an home-written monitoring program which was able to compute the real percentage of CPU obtained by each of the jobs in the lot, as well as its average priority.

**Figure 2: VMS Scheduler**

| User | CPU% | Priority |
|---|---|---|
| ARCHIMEDES | 6.21 | 5.00 |
| EUCLID | 6.18 | 5.19 |
| HERACLIT | 4.99 | 6.00 |
| DIOGENES | 5.65 | 5.31 |
| ARISTOTELES | 6.74 | 5.88 |
| PLATO | 8.66 | 6.71 |
| SOPHOCLES | 5.75 | 5.19 |
| EURIPIDES | 7.15 | 6.88 |
| DEMOSTENES | 14.74 | 8.35 |

Figure 2 shows the results obtained running with the standard VMS scheduler, and Figure 3 those obtained by the CERN modified version.

You can see that in standard VMS the differences in percentage of CPU obtained are minimal, even between jobs that have totally different attitudes: PITAGORA is 100% CPU bound and gets 6.18% of the VAX CPU, while PLATO, who is 50% CPU and 50% Terminal Input oriented, gets a mere 2% more. The only job that gets a reasonable attention by standard VMS is DEMOSTENES, which is in fact an endless loop on keyboard read, with a 1 byte buffer. Hopefully there is nothing like that in real life.

| Figure 2: CERN Scheduler | | |
|---|---|---|
| User | CPU% | Priority |
| PITAGORA | 3.38 | 8.63 |
| ARCHIMEDES | 3.22 | 8.69 |
| EUCLID | 4.66 | 9.13 |
| HERACLIT | 11.28 | 9.25 |
| DIOGENES | 4.00 | 9.19 |
| ARISTOTELES | 5.24 | 10.13 |
| PLATO | 30.92 | 11.50 |
| SOPHOCLES | 3.19 | 9.38 |
| EURIPIDES | 5.47 | 10.63 |
| DEMOSTENES | 16.68 | 12.06 |

In the CERN version instead (Fig. 3), you can see that the larger percentage of the CPU goes to Terminal Input bound users. These are the most interactive ones, because a terminal input normally requires human (or generally a speaking animal) intervention by a finger (and brain too sometimes).

| Figure 4: Relative speed | |
|---|---|
| PITAGORA | 0.54 |
| ARCHIMEDES | 0.51 |
| EUCLID | 0.75 |
| HERACLIT | 2.26 |
| DIOGENES | 0.70 |
| ARISTOTELES | 0.77 |
| PLATO | 3.57 |
| SOPHOCLES | 0.55 |
| EURIPIDES | 0.76 |
| DEMOSTENES | 1.13 |

Figure 4 shows the "relative speed" (i.e. fig. 3 divided by fig. 2). You see that all Terminal Input jobs run up to 3.5 times faster. In particular, PLATO and HERACLIT represent a closer approximation to a full screen editor, or a data entry system.

As a consequence, all the other Greeks slow down by half or one third on our scheduler. What this shows really is the limit which Terminal Input oriented applications can get if working at a Kalatchnikov typing speed, and proves that our scheduler is more sensitive to interaction than the standard one.

Please note, however, that this is a "margin conditions" benchmark, not done with real life applications, but with impressionist programs that are very specialized in only one kind of operations.

PITAGORA, who lives full time in the CPU, with no I/O whatsoever, exists only in a mathematician's mind, and DEMOSTENES can only exist if someone falls asleep in EDT, with his nose on the RETURN key.

A real life job consists in fact of a mixtures of these types of I/O operations, so it will get different kind of boosts, at varying intervals, therefore smoothing down the difference in the size of the buildings.

Also note that this benchmark was done with a quantum value of 200 ms. A lower value will give extra control on the CPU re-partition.

Our mod to the scheduler consists of a binary patch to VMS. Which is a terrible thing to do.

Don't forget that patching the system makes you blind, or worse, you lose the DEC Software Warranty.

So by this mortal sin we sold our soul to buy user satisfaction.

And now we are wandering restless in the Depths of the Fiery Caverns for System Managers.

I thank Eric Mc Intosh and Les Robertson for their contribution in work and ideas, Hermes Trismegistos (currently reincarnated in the body of Mike Dawkes) for the magic formulas, and Alan Silverman for letting us all infest his machines.

# Evaluation of Third Party VAX/VMS Disk Compression Products

Marian K Iannuzzi
Westinghouse Electric Corporation
Baltimore, Maryland 21203

## ABSTRACT

A number of third party vendors have introduced disk compression utilities which promise to eliminate the use of BACKUP/RESTORE for disk optimization. This paper presents a detailed evaluation of three products with emphasis on disk integrity, algorithms used and degree of optimization obtained. Description of evaluation methods used and benchmarks are included.

## INTRODUCTION

This evaluation began in November 1986 as an attempt to obtain an apples-to-apples comparison of third party disk optimization products. Any product to be tested in-house needed to meet the following requirements:

- Must permit other disk access
- Interactive and batch modes
- No file characteristics changed
- Data integrity preserved
- Runs quickly
- Concurrent processing of multiple disks
- Volume and shadow sets handled
- Likely to work with future VMS updates
- Gracefully handles unexpected events
- Optimizes free space and files
- No spare disk required

In December 1986, three products met the pretest criteria, Squeezpak, Diskeeper, and Rabbit-7 (figure 1). Defrag V2.0 was no longer considered because information from H&E Concepts stated "All file characteristics (except FILE ID) are unchanged for moved files...". Diskit/VMS was not further considered since it operated only in an offline mode. Demonstration copies of the remaining products were obtained for a two part evaluation: initial test and stress test.

## INITIAL TEST

The initial testing was designed to quickly identify any problems in the following areas:

- Product installation
- Ability to handle unexpected events (power outages, etc)
- File header modifications
- Data integrity
- File and free space optimization

The preliminary test environment consisted of a standalone VAXstation II/GPX with two RD54s, one for all logging and the other as the test disk. The author was the sole user of the system. The initial test was comprised of two portions:

1) A bit-by-bit comparison of the disk before and after optimization
2) Recovery from unexpected events.

A command procedure initialized the test disk with a cluster size of one, then created 31 files:

- Twenty of various sizes placed by specific logical block number (LBN)
- Ten one-block files placed by VMS
- One 101-block file split into nine one-block chunks plus one 92-block chunk.

The resulting disk was less than 1% full with one non-contiguous file (NCF). A DUMP of each file on the disk was recorded and each product was allowed one pass or run. DCL DIFFERENCES was used to compare the before and after DUMP files. Anything flagged in the file header area was carefully examined. Changes in the map area, for example, would be acceptable while a new creation date would not.

One item worth mentioning is that Squeezpak "tags" all files that it moves, as the relocated files are moved by exact placement. Squeezpak sets a file characteristic bit in the file header area to indicate on successive passes that a file was placed by Squeezpak and not by a user. VMS sources indicate that this bit is user-definable.

For the second portion of the initial test, each product was interrupted during the file copy phase by pressing the halt button, pulling the power plug, and typing control-Y in successive trials. The vendor's specified recovery procedure was performed and the disk was scrutinized for any anomalies.

All three products were easy to install, successfully handled unexpected events, did not damage any data, and files were manipulated to eliminate the non-contiguous file and to consolidate the free space.

## ALGORITHMS

In an attempt to better understand how each product attacked the optimization problem, the strategy utilized by each was inspected.

### Squeezpak

Squeezpak's simplified algorithm has a two-step approach - all the file fragments are collected to make as many files contiguous as possible, and then the free space is consolidated (figure 2). In the first step, a list of all the non-contiguous files is made, ordered by block size from the largest to smallest. Scanning begins with the largest file and continues through the entire list. The file is moved only if enough free space is available to make the file contiguous (figure 3). If more than one such free space exists, the file is moved to the one closest to logical block zero.

In the second step, free space collection begins at logical block zero and the entire disk is scanned, stopping at each free space (figure 4). When a free space is located, one of two things can occur:

A) A file which has the same number of blocks as the free space is moved to fill the free space

OR

B) The file immediately following the free space is "slid" or relocated into the free space. This sliding of files continues until two free spaces are joined.

The overall result is that contiguous files are positioned near the front of the disk and free space near the end.

### Diskeeper

Diskeeper's simplified algorithm (figure 5) moves through the disk by file id. Contiguous files are not moved unless doing so makes the free space more contiguous. If more than one area of free space exists where a file could be made contiguous, the area closest to the end of the disk is chosen. Files that are not able to be made contiguous are only moved if doing so results in less file fragments. The overall result is that contiguous files are positioned near the end of the disk and free space near the beginning.

### Rabbit-7

Rabbit-7's simplified algorithm (figure 6), begins with LBN zero and scans to the largest LBN. When a free space is located, the largest file that fits is moved into the free space and the scanning continues. If a file to occupy all or a portion of the free space does not exist, the file immediately following the free space is moved, contiguously if possible, toward the end of the disk. A larger free space is created and again an attempt is made to find and move the largest file that fits into the space.

When a file is located, if it is contiguous, the scanning continues. If the file is non-contiguous, it is moved toward the end of the disk, contiguously if possible. This creates two or more free spaces. The free space with the lowest logical block number is

treated as any other free space and the processing continues. The overall result is that contiguous files are positioned near the end of the disk and free space near the beginning.

## STRESS TEST

The stress test was performed under controlled conditions to provide a valid comparison. This environment consisted of a VAX 86XX, part of a three node cluster, two RA81s, and an HSC70. The disks, when used for the test, were mounted privately. Actual user disk snapshots were copied from tape through the HSC70 so each product could have the same starting point. The test cases were:

A) Single RA81
B) Shadowed RA81
C) Two-RA81 volume set
AND
D) Different single RA81
E) Different two-RA81 volume set

Test A was performed on a VAX 8600, all others on a VAX 8650. Tests A, B, and C were concluded in June 1987, and test cases A, B and C destroyed. Since these snapshots were no longer available when new versions of Squeezpak and Diskeeper were released in November 1987, the three products were again tested using D and E.

Command procedures were used to collect data before, during and after every pass of the compression product. All journal, log and report files were not written to the disk(s) being optimized. No spare disk was provided for optimization. Directory files were allowed to be moved and none of the test disks contained any open files.

The disk analysis utilities provided by each product, as well as VAX SPM and DCL commands, were used to collect additional and duplicate information. Each product was allowed six consecutive passes on each test case. The batch jobs were executed in a dedicated, interactive priority, system queue during light load periods - evenings, nights and weekends.

## RESULTS

Using the mythical "perfect" disk compression product as a standard, it was hoped that the following could be obtained: on a disk at least 85% full, in one pass, return zero NCFs, one free space equal in size to the total number of free blocks on the disk, in minimal run-time without user disruption or system manager interaction. Conversely, the number of NCFs and free spaces should not increase and the largest free space should remain constant. That is, a product should NOT make a bad situation worse.

Some trends to keep in mind while reviewing the results (figures 7-10):

- Mean extents/file should approach 1.00.
- Number of free spaces should approach 1, except on the volume sets.
- Mean blocks/free space and largest free space should approach the same number, the total number of free blocks on the disk.
- File transfers are an indicator of the amount of "work" done to accomplish the results.
- Wall clock time is included only as a rough guideline of what to expect in similar situations.

## CONCLUSIONS

The test results for the earlier versions and the most recent versions demonstrate rapid product development. Diskeeper version 1.3, instance, made the free space less optimized (figures 7 and 8), but this trend was improved in version 2.0 (figures 10 and 11).

The actual number of non-contiguous files on a disk still remains a mystery. Although discussion with each vendor revealed no differences in the definition of a non-contiguous file, it is obvious that the variance is non-trivial. These results should be used to better understand each product's strengths and weaknesses and how each might be exploited in a particular environment.

While perfection remains elusive, safe, reasonable disk optimization is attainable, and offers significant advantages over the traditional method of BACKUP/RESTORE.

| Product | Vendor | Tested |
|---|---|---|
| Squeezpak V1.2 | DEMAC Software Ltd. | YES |
| Diskeeper V1.2 | Executive Software, Inc. | YES |
| Rabbit-7 V1.0 | RAXCO Rabbit Software | YES |
| Defrag V2.0 | H & E Concepts | NO - changes FID |
| Diskit/VMS | Software Techniques Inc. | NO - offline mode only |

Figure 1: Products Considered - Dec 1986

# Squeezpak Algorithm (Simplified)

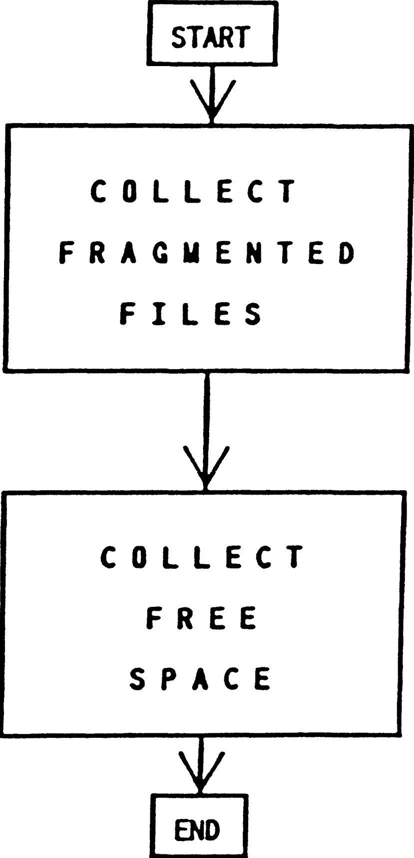



Figure: 2

# Squeezpak -- Collect Fragmented Files

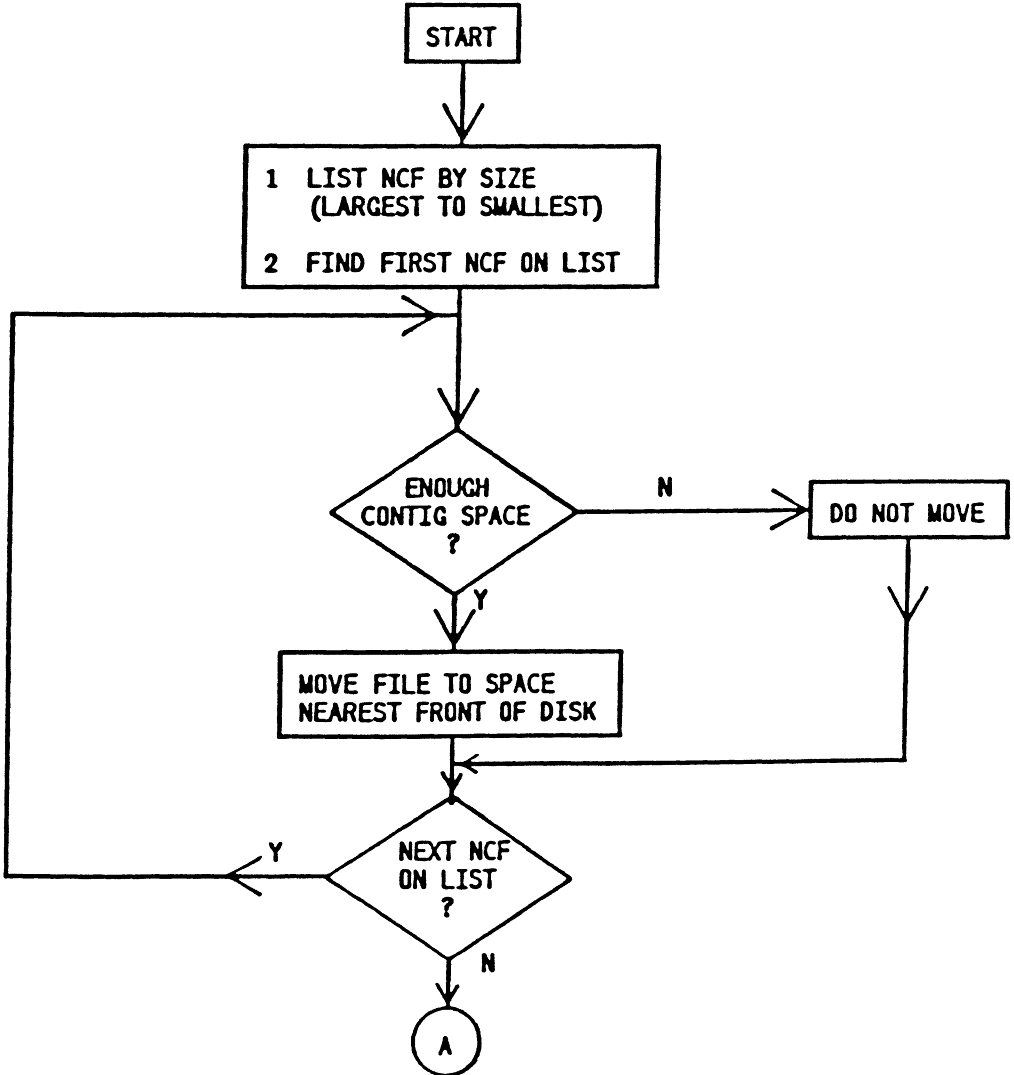



Figure: 3

# Squeezpak -- Collect Free Space

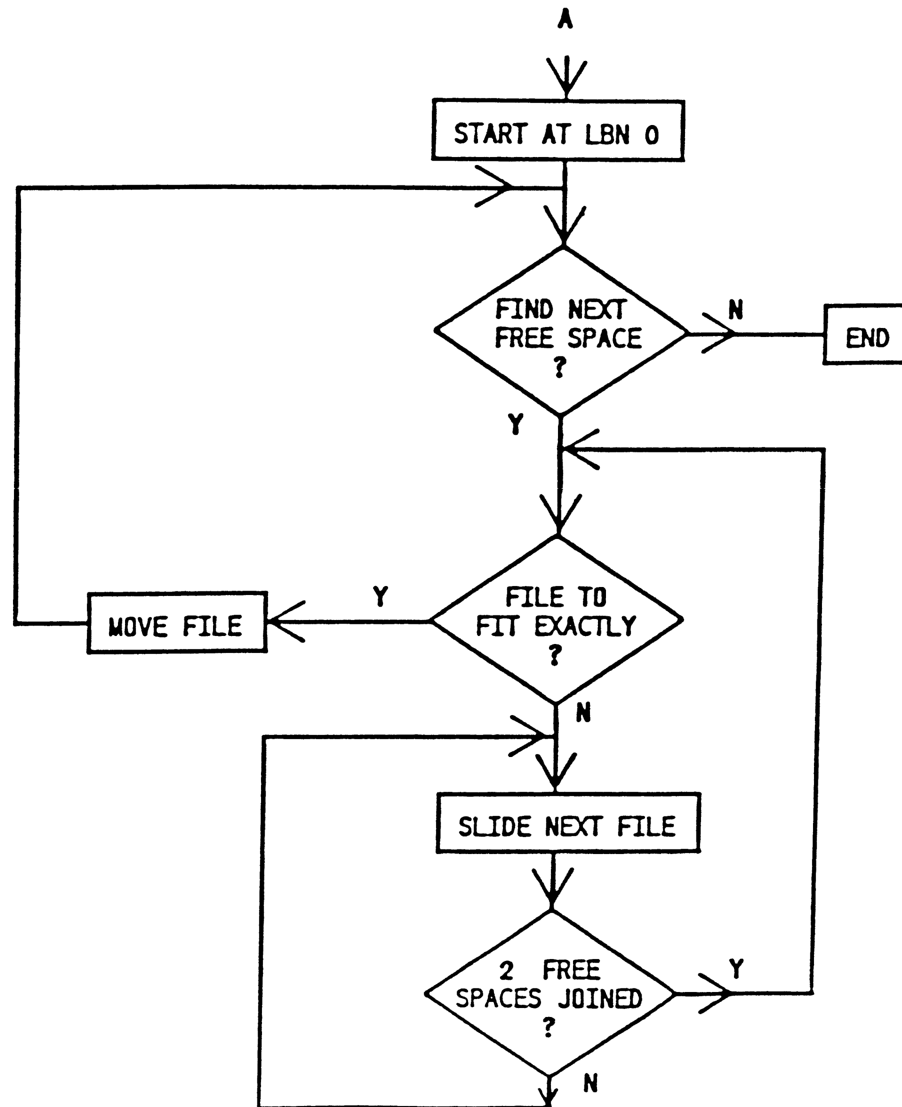



Figure: 4

# Diskeeper Algorithm (Simplified)

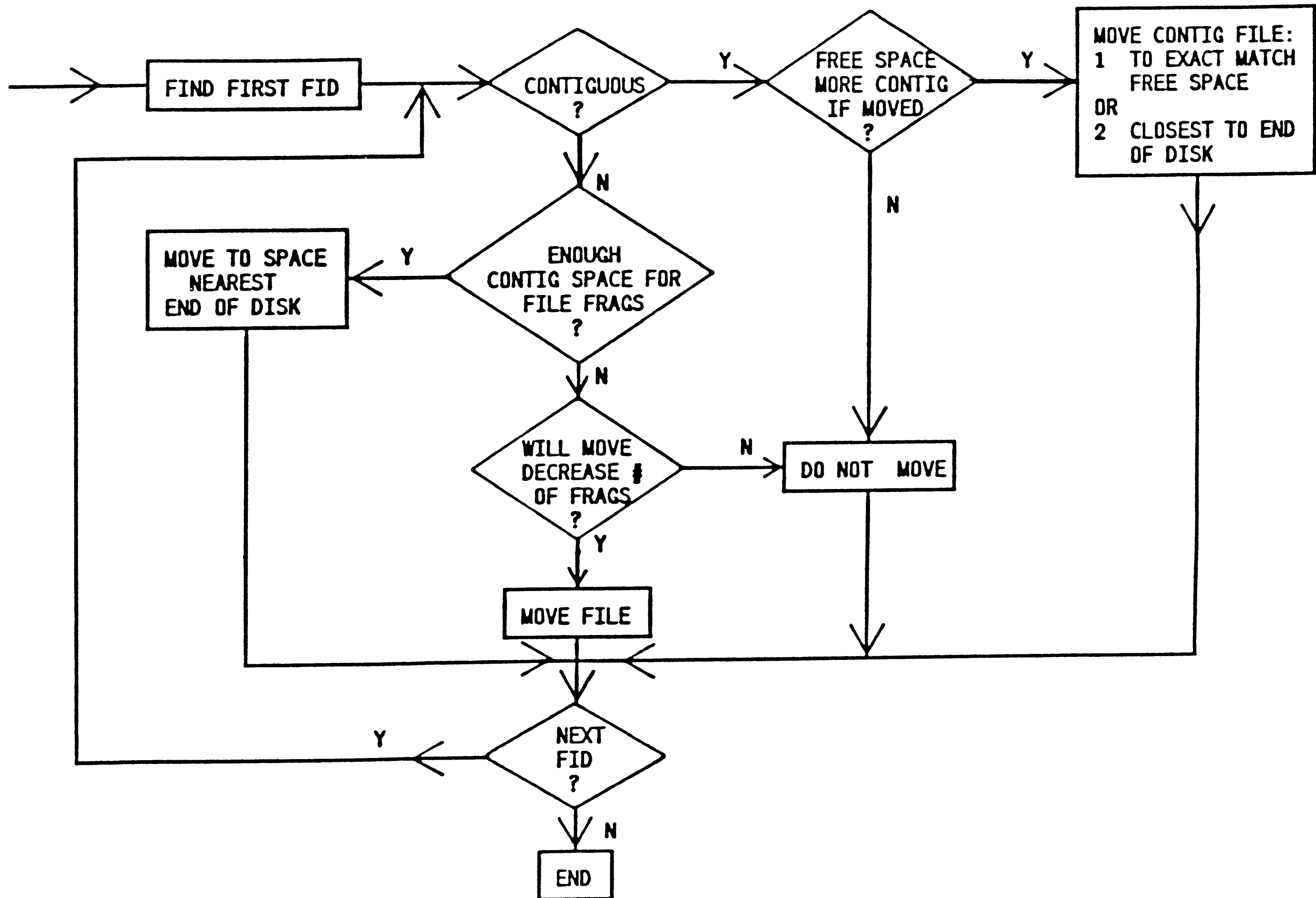



Figure: 5

# Rabbit-7 Algorithm (Simplified)

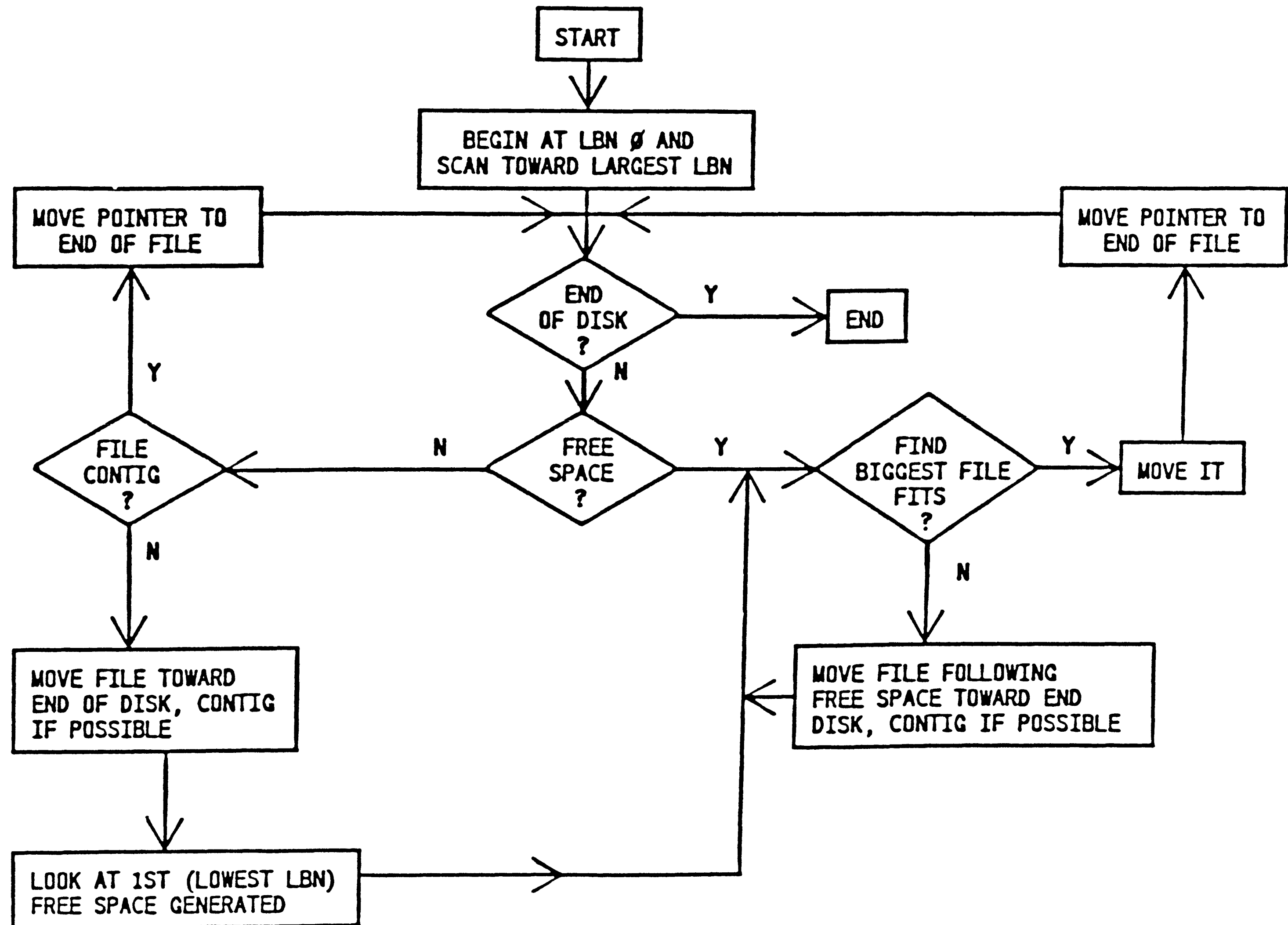



Figure: 6

| | PASSES | | | | | | |
|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| # NCF | 4,404 | 765 | 674 | 671 | 671 | 670 | 670 |
| Mean Extents/File | 2.21 | 1.75 | 1.70 | 1.69 | 1.69 | 1.69 | 1.69 |
| Free Spaces | 7,809 | 7,474 | 7,391 | 7,361 | 7,357 | 7,350 | 7,347 |
| Mean blocks/Free Space | 15 | 16 | 16 | 16 | 16 | 16 | 16 |
| Largest Free Space | 240 | 901 | 901 | 1026 | 1026 | 1026 | 1026 |
| File Transfers | --- | 16,101 | 2,855 | 688 | 238 | 235 | 173 |
| Elapsed CPU Time * | --- | 33:44 | 24:51 | 13:39 | 13:32 | 13:29 | 12:56 |
| Elapsed Wall Clock * | --- | 6:00:43 | 6:30:39 | 3:33:18 | 3:20:38 | 2:20:24 | 1:43:52 |

Squeezpak Results Version 2.0a - Single RA81

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| # NCF | 4,453 | 408 | 353 | 335 | 334 | 331 | 322 |
| Mean Extents/File | 2.21 | 1.57 | 1.53 | 1.52 | 1.52 | 1.52 | 1.51 |
| Free Spaces | 7,809 | 7,925 | 5,868 | 5,595 | 5,591 | 5,628 | 5,723 |
| Mean blocks/Free Space | 15 | 14 | 20 | 21 | 21 | 20 | 20 |
| Largest Free Space | 240 | 196 | 434 | 240 | 240 | 242 | 240 |
| File Transfers | --- | 17,997 | 11,236 | 8,244 | 5,148 | 389 | 3,271 |
| Elapsed CPU Time * | --- | 13:28 | 11:33 | 9:36 | 8:47 | 7:43 | 8:16 |
| Elapsed Wall Clock * | --- | 3:51:22 | 2:19:04 | 1:41:13 | 1:03:32 | 13:09 | 43:34 |

Diskeeper Results Version 1.3 - Single RA81

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| # NCF | 4,405 | 1 | 1 | 1 | 1 | 1 | 1 |
| Mean Extents/File | 2.21 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| Free Spaces | 7,809 | 17 | 17 | 17 | 17 | 17 | 17 |
| Mean blocks/Free Space | 15 | 6,929 | 6,9296 | 6,929 | 6,929 | 6,929 | 6,929 |
| Largest Free Space | 240 | 60,720 | 60,720 | 60,720 | 60,720 | 60,720 | 60,720 |
| File Transfers | --- | 13,860 | 29 | 0 | 0 | 0 | 0 |
| Elapsed CPU Time * | --- | 17:41 | 2:31 | 2:28 | 2:32 | 2:31 | 2:28 |
| Elapsed Wall Clock * | --- | 3:31:50 | 13:48 | 13:14 | 13:14 | 13:16 | 13:14 |

Rabbit-7 Results Version 1.03f - Single RA81

Environment: Run on VAX 8600 - system batch queue; disk 86.7% full; cluster size=1; analyze/disk/repair performed before compression.

* rounded to the nearest second

Figure 7: TEST CASE A RESULTS

| | PASSES 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| # NCF | 290 | 4 | 2 | 2 | 2 | 2 | 2 |
| Mean Extents/File | 1.03 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| Free Spaces | 175 | 44 | 37 | 38 | 33 | 32 | 38 |
| Mean blocks/Free Space | 219 | 869 | 1,034 | 1,007 | 1,159 | 1,195 | 1,007 |
| Largest Free Space | 8,240 | 8,819 | 8,502 | 7,111 | 7,046 | 9,072 | 14,662 |
| File Transfers | --- | 2,576 | 424 | 148 | 130 | 136 | 140 |
| Elapsed CPU Time * | --- | 5:37 | 3:43 | 3:31 | 3:33 | 3:31 | 3:28 |
| Elapsed Wall Clock * | --- | 1:09:04 | 21:40 | 17:57 | 16:26 | 17:51 | 17:24 |

Squeezpak Results Version 2.0a - Shadowed RA81

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| # NCF | 292 | 5 | 4 | 4 | 4 | 4 | 4 |
| Mean Extents/File | 1.03 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| Free Spaces | 175 | 525 | 163 | 169 | 169 | 169 | 169 |
| Mean blocks/Free Space | 219 | 73 | 235 | 226 | 226 | 226 | 226 |
| Largest Free Space | 8,240 | 3,246 | 3,246 | 3,246 | 3,246 | 3,246 | 3,246 |
| File Transfers | --- | 4,968 | 2,818 | 380 | 0 | 0 | 0 |
| Elapsed CPU Time * | --- | 3:00 | 3:48 | 2:17 | 2:18 | 2:18 | 2:17 |
| Elapsed Wall Clock * | --- | 1:11:38 | 34:04 | 6:25 | 2:42 | 2:40 | 2:40 |

Diskeeper Results Version 1.3 - Shadowed RA81

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| # NCF | 291 | 3 | 3 | 3 | 3 | 3 | 3 |
| Mean Extents/File | 1.03 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| Free Spaces | 176 | 3 | 2 | 2 | 2 | 2 | 2 |
| Mean blocks/Free Space | 217 | 12,750 | 19,124 | 19,124 | 19,124 | 19,124 | 19,124 |
| Largest Free Space | 8,240 | 35,855 | 35,854 | 35,854 | 35,854 | 35,854 | 35,854 |
| File Transfers | --- | 2,330 | 2 | 0 | 0 | 0 | 0 |
| Elapsed CPU Time * | --- | 3:37 | 2:06 | 2:04 | 2:06 | 2:08 | 2:05 |
| Elapsed Wall Clock * | --- | 38:26 | 11:10 | 11:00 | 11:05 | 11:02 | 11:01 |

Rabbit-7 Results Version 1.03f - Shadowed RA81

Environment: Run on VAX 8650 - system batch queue; disk 95.7% full; cluster size=1; analyze/disk/repair performed before compression.

* rounded to the nearest second

Figure 8: TEST CASE B RESULTS

| | PASSES 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| # NCF | 3,318 | 450 | 0 | 0 | 0 | 0 | 0 |
| Mean Extents/File | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Free Spaces | 7,420 | 1,320 | 109 | 65 | 58 | 44 | 40 |
| Mean blocks/Free Space | 35 | 202 | 2,437 | 3,988 | 4,474 | 6,091 | 6,628 |
| Largest Free Space | 2,348 | 20,197 | 100,585 | 51,047 | 47,228 | 61,137 | 59,172 |
| File Transfers | --- | 39,951 | 13,808 | 538 | 406 | 394 | 364 |
| Elapsed CPU Time * | --- | 30:33 | 18:34 | 7:56 | 7:50 | 7:40 | 7:39 |
| Elapsed Wall Clock * | --- | 6:17 | 3:26 | 46:39 | 42:09 | 40:42 | 39:30 |

Squeezpak Results Version 2.0a - Vol Set

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| # NCF | 3,334 | 132 | 34 | 29 | 29 | 29 | 29 |
| Mean Extents/File | 1.21 | 1.04 | 1.02 | 1.02 | 1.02 | 1.02 | 1.02 |
| Free Spaces | 7,420 | 6,042 | 3,976 | 3,850 | 3,826 | 3,820 | 3,821 |
| Mean blocks/Free Space | 35 | 43 | 64 | 66 | 67 | 67 | 67 |
| Largest Free Space | 2,348 | 7,385 | 6,991 | 5,717 | 5,707 | 5,717 | 5,717 |
| File Transfers | --- | 36,322 | 16,567 | 6,249 | 864 | 73 | 6 |
| Elapsed CPU Time * | --- | 16:28 | 15:08 | 11:18 | 11:47 | 14:12 | 13:53 |
| Elapsed Wall Clock * | --- | 3:38:58 | 1:36:38 | 42:46 | 29:35 | 59:38 | 34:44 |

Diskeeper Results Version 1.3 - Vol Set

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| # NCF | 3,328 | 11 | 10 | 10 | 10 | 10 | 10 |
| Mean Extents/File | 1.22 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| Free Spaces | 7,420 | 5 | 2 | 2 | 2 | 2 | 2 |
| Mean blocks/Free Space | 35 | 129,589 | 129,589 | 129,589 | 129,589 | 129,589 | 129,589 |
| Largest Free Space | 2,348 | 123,239 | 129,589 | 129,589 | 129,589 | 129,589 | 129,589 |
| File Transfers | --- | 21,069 | 3,016 | 0 | 0 | 0 | 0 |
| Elapsed CPU Time * | --- | 23:34 | 10:01 | 7:44 | 7:43 | 7:45 | 7:51 |
| Elapsed Wall Clock * | --- | 4:11:39 | 1:06:50 | 42:32 | 42:37 | 42:33 | 42:38 |

Rabbit-7 Results Version 2.0 - Vol Set

Environment: Run on VAX 8650 - system batch queue; volume 92.7% full; cluster size=1; analyze/disk/repair performed before compression.

* rounded to the nearest second

this data represents 2 RA81s, therefore perfection in free spaces = 2

Figure 9: TEST CASE C RESULTS

| | PASSES 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| # NCF | 5,896 | 387 | 161 | 128 | 122 | 117 | 117 |
| Mean Extents/File | 3.02 | 1.96 | 1.60 | 1.54 | 1.54 | 1.50 | 1.49 |
| Free Spaces | 12,198 | 10,369 | 8,295 | 7,919 | 7,861 | 7,478 | 7,399 |
| Mean blocks/Free Space | 17 | 19 | 24 | 25 | 25 | 27 | 27 |
| Largest Free Space | 690 | 2,371 | 6,215 | 1,802 | 2,228 | 1,012 | 1,183 |
| File Transfers | --- | 18,692 | 1,989 | 323 | 26 | 438 | 12 |
| Elapsed CPU Time * | --- | 42:10 | 19:58 | 13:33 | 12:41 | 14:12 | 12:28 |
| Elapsed Wall Clock * | --- | 5:49:47 | 3:11:56 | 1:59:59 | 1:54:02 | 3:43:52 | 2:17:40 |

Squeezpak Results Version 2.1y - Single RA81

| | PASSES 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| # NCF | 6,029 | 666 | 162 | 30 | 2 | 1 | 1 |
| Mean Extents/File | 3.02 | 1.24 | 1.01 | 1.01 | 1.00 | 1.00 | 1.00 |
| Free Spaces | 12,198 | 5,487 | 1,047 | 261 | 96 | 36 | 19 |
| Mean blocks/Free Space | 17 | 37 | 194 | 781 | 2,123 | 5,662 | 10,729 |
| Largest Free Space | 690 | 392 | 1,832 | 3,632 | 12,077 | 24,677 | 34,798 |
| File Transfers | --- | 10,452 | 4,276 | 1,934 | 694 | 1,544 | 302 |
| Elapsed CPU Time * | --- | 38:01 | 8:55 | 3:00 | 00:48 | 1:44 | 00:22 |
| Elapsed Wall Clock * | --- | 3:19:25 | 1:09:06 | 32:13 | 13:48 | 16:30 | 4:50 |

Diskeeper Results Version 2.0 - Single RA81

| | PASSES 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| # NCF | 5,902 | 34 | 20 | 20 | 20 | 20 | 20 |
| Mean Extents/File | 3.02 | 1.19 | 1.19 | 1.19 | 1.19 | 1.19 | 1.19 |
| Free Spaces | 12,198 | 1,098 | 1,042 | 1,014 | 1,040 | 1,039 | 1,038 |
| Mean blocks/Free Space | 17 | 186 | 196 | 196 | 196 | 196 | 196 |
| Largest Free Space | 690 | 13,489 | 13,489 | 13,489 | 13,489 | 13,489 | 13,489 |
| File Transfers | --- | 15,860 | 103 | 70 | 15 | 15 | 15 |
| Elapsed CPU Time * | --- | 15:56 | 2:40 | 2:40 | 2:39 | 2:37 | 2:38 |
| Elapsed Wall Clock * | --- | 3:05:36 | 14:28 | 14:02 | 13:36 | 13:37 | 13:42 |

Rabbit-7 Results Version 2.0g - Single RA81

Environment: Run on VAX 8650 - system batch queue; disk 77% full; cluster size=1; analyze/disk/repair performed before compression.
* rounded to the nearest second

Figure 10: TEST CASE D RESULTS

| | PASSES | | | | | | |
|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| # NCF | 12,095 | 1,178 | 687 | 632 | 507 | 506 | 463 |
| Mean Extents/File | 2.04 | 1.55 | 1.42 | 1.37 | 1.36 | 1.35 | 1.35 |
| Free Spaces | 17,879 | 14,172 | 14,046 | 12,975 | 12,665 | 12,400 | 12,303 |
| Mean blocks/Free Space | 10 | 13 | 13 | 14 | 15 | 15 | 15 |
| Largest Free Space | 966 | 5,729 | 3,749 | 1,782 | 1,000 | 1,406 | 1,298 |
| File Transfers | --- | 33,138 | 24,970 | 16,147 | 12,788 | 9,498 | 7,583 |
| Elapsed CPU Time * | --- | 78:12 | 43:02 | 30:57 | 26:40 | 25:54 | 25:29 |
| Elapsed Wall Clock * | --- | 11:50:58 | 11:03:36 | 5:23:08 | 4:31:42 | 4:48:18 | 7:22:00 |

Squeezpak Results Version 2.1y - Vol Set

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| # NCF | 12,183 | 1,613 | 1,182 | 941 | 794 | 697 | 614 |
| Mean Extents/File | 2.04 | 1.6 | 1.15 | 1.06 | 1.04 | 1.03 | 1.03 |
| Free Spaces | 17,879 | 9,430 | 3,804 | 1,890 | 1,320 | 1,145 | 706 |
| Mean blocks/Free Space | 10 | 19 | 47 | 96 | 138 | 192 | 258 |
| Largest Free Space | 966 | 1,001 | 504 | 573 | 1,013 | 962 | 1,805 |
| File Transfers | --- | 122,158 | 8,397 | 5,702 | 4,582 | 4,170 | 3,724 |
| Elapsed CPU Time * | --- | 1:09:34 | 29:26 | 18:30 | 11:11 | 8:12 | 6:17 |
| Elapsed Wall Clock * | --- | 1:09:34 | 29:25 | 18:30 | 11:11 | 8:12 | 6:17 |

Diskeeper Results Version 2.0 - Vol Set

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| # NCF | 12,129 | 27 | 17 | 17 | 17 | 17 | 17 |
| Mean Extents/File | 2.04 | 1.04 | 1.04 | 1.04 | 1.04 | 1.04 | 1.04 |
| Free Spaces | 17,879 | 564 | 339 | 340 | 340 | 340 | 340 |
| Mean blocks/Free Space | 10 | 343 | 3,356 | 3,355 | 3,355 | 3,355 | 3,355 |
| Largest Free Space | 966 | 46,102 | 99,837 | 99,837 | 99,837 | 99,837 | 99,837 |
| File Transfers | --- | 31,906 | 3,094 | 1 | 0 | 0 | 0 |
| Elapsed CPU Time * | --- | 41:47 | 8:59 | 6:45 | 6:40 | 6:43 | 6:46 |
| Elapsed Wall Clock * | --- | 7:54:27 | 1:08:49 | 35:38 | 35:35 | 35:35 | 35:35 |

Rabbit-7 Results Version 2.0g - Vol Set

Environment: Run on VAX 8650 - system batch queue; volume 89.7% full; cluster size=1; analyze/disk/repair performed before compression.

* - rounded to the nearest second

this data represents 2 RA81s, therefore perfection in free spaces = 2

Figure 11: TEST CASE E RESULTS

# VMS DISK PERFORMANCE

Wef Fleischman
UIS/Software Techniques, Inc.
DECUS Anaheim, Fall 1987

**ABSTRACT**

Disk performance under VMS is determined by many factors. This article surveys the most important factors under the system manager's control. The efficiency of the VMS file caching system also contributes to file throughput and so this topic is covered in some detail. By understanding the basis of disk performance you can make the most intelligent tuning decisions about managing your VAX system. The following text is a transcript of a DECUS presentation given at the Anaheim, Fall 1987 symposium.

## INTRODUCTION

It seems that no matter how many advances are made in computer technology one thing always surfaces as a top wish-list item: *make the system run faster.*

Today, I would like to talk about several aspects of VMS disks that probably interest you. At every DECUS, for the past several years, I've talked about different aspects of disk optimization, none quite the same. I don't like to repeat material because it gets tiring to both you and me. Therefore, those of you who may have attended my sessions before are in luck because I plan to address some topics that have not been discussed in detail before. These are areas that will become commonly debated in the future.

Figure 1 summarizes the basic areas of my talk today.

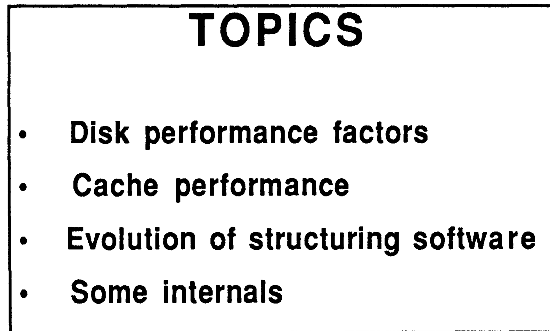



*Figure 1*

As compared to a few years ago, when there was very little concern about disk performance, today we've grown up and realized that disk performance problems arise on *all* systems, from time to time, and good system managers must maintain a constant vigil. The hope is that by learning to recognize the tell-tale signs of disk performance problems, we will be prepared to deal with the problems effectively.

Why has disk performance become such a hot topic? Part of the reason is that VAXes are being used in increasingly demanding environments, more challenging than even two years ago. And as we become more demanding, the more performance becomes a decisive factor. With these increasing needs, system managers have become more skillful and creative in finding techniques to *squeeze* out more throughput.

My first objective today is to review five basic disk performance factors that you may or may not have already heard something about. Second, most of you, if not all, are aware that there are third-party tools available to help optimize disks in various ways. But you need answers to the questions of: "Which tool is best for me?" and "What features will make the job of system management easier?"

As one of the veterans who's studied disk performance for many years, I hope to be able to share some of the history and evolution of disk structuring software with you now. I think it's interesting to see how far we've come in this technology, and how the most recent techniques in-use are becoming increasingly important.

The technology that is just breaking today is what you and I have come to DECUS to hear about. It is my hope that you will come away from this session with a good working knowledge of the technical issues involved in disk performance management.

## FILE PLACEMENT

The location of active files has an important effect on disk performance. Disks are mechanical devices, so it takes time to move the heads to the correct cylinder and to wait for the desired sector to rotate into position for reading or writing.

If two files, that are often used (such as those shown in Figure 2), are located right next to each other, the throughput of the disk is the best possible. Consequently, all users of the disk benefit.

However, if these same two files are located many cylinders apart, the disk makes the user wait for the heads to seek from one position to another, lowering the disk's throughput and tying it up from performing other user's requests.

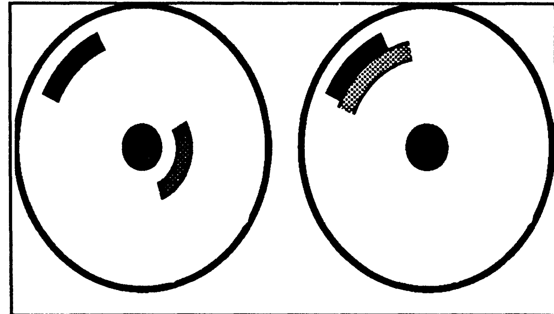


*Figure 2*

I would like to share a benchmark which I performed to quantify the effect of file placement on disk throughput.

A small test program repetitively read one block from the disk at one location, and then a block from another location on the disk. This simulated the disk operations that occur when accessing two files located at different places on the disk. After 1000 iterations, the elapsed time was recorded for each of three different disks on a MicroVAX.

The first disk tested was a Fujitsu 2242 disk, 61 megabytes in size. Note in Figure 3 that when the two files were close to each other it took around 30 seconds to complete 1000 iterations. As the separation distance was increased, this elapsed time increased to about 8 times this amount. The time increased roughly proportional to the seek distance between the files.

I also wanted to demonstrate the effect on a Digital disk, so I then benchmarked a 71 megabyte RD53 drive. Again, very similar results were observed, as shown in Figure 4.

Lastly, I performed the identical benchmark on a very high performance drive, a CDC 9772 XMD. This drive is very large (858 megabytes) and known to be quite fast. This disk showed basically the same characteristics as in the previous two examples. When the separation of the data was minimal, it still required about 30 seconds to complete the benchmark. When widely separated, it took longer.

It's worth noting that in the last test, the worst separation was faster than that for the other disks, despite the fact that the CDC 9772 XMD disk is over 800 megabytes while the other disks were both less than 100 megabytes.

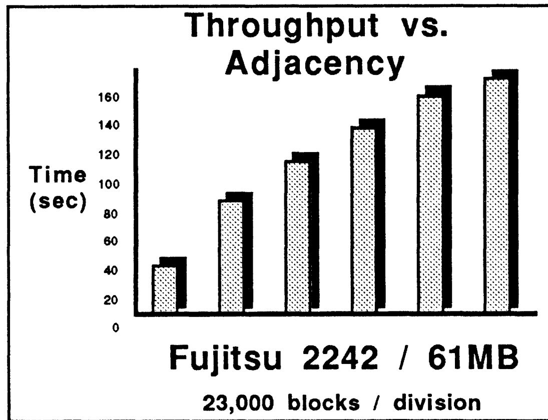



*Figure 3*

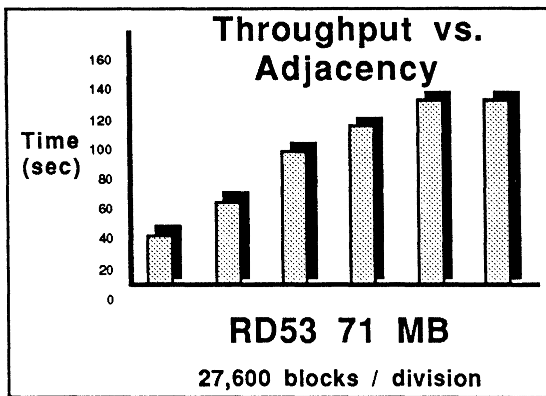



*Figure 4*

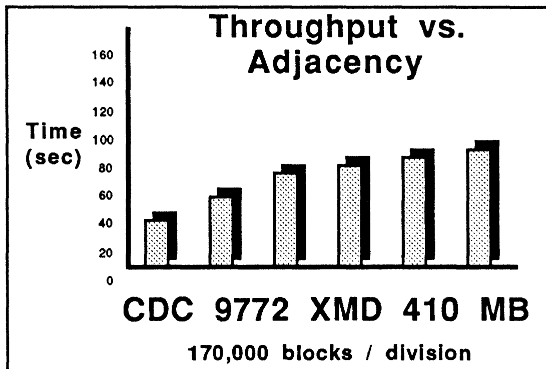



*Figure 5*

The moral to be learned here is: *faster disks can better hide the effects of randomly separated files.* Faster disks can be left with more scattered data and the performance impact will be less painful. The converse is also true: *the slower a disk is, the more important it is to worry about file placement.*

Now that we have established the importance of file placement, the next task in turn is to identify the files that are being accessed the most. This allows you to concentrate your efforts on their placement. You should, of course, first use your own knowledge about your particular applications and how users run them on your system. This will probably lead you to the conclusion that certain key files are accessed regularly; write these files down as deserving special attention. Then consider a second category of files: those that the VMS operating system accesses often. These files also benefit from being placed optimally. The files to consider are summarized in Figure 6.

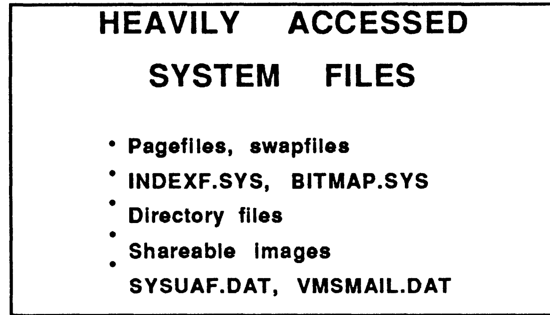



*Figure 6*

The single most-used files on a VMS system are the pagefiles. If your system lacks bountiful memory resources, the swapfiles may be utilized often as well. The volume directory structure, stored in .DIR files are also often referenced. Those files common to many users (such as system directories like SYSEXE) may be re-read many times. Note that directory files are accessed *so* often that the Files-11 Extended QIO Processor (the XQP) stores as much of them as possible in the directory cache. The cache is not large enough to store all directories, however, and so placement of the directory files should not be overlooked.[1]

As shown, other files that tend to be accessed regularly are the system and network authorization files, the proxy database and VMSMAIL.DAT.

When you have identified the files to be placed for optimum accessing, the next question you must ask is: "Where?" The *center* (see Figure 7) of the disk is the most optimal file placement. This is because it takes the least amount of time and distance, *on average,* to reach the center of the disk. In addition, when a heavily accessed file is located centrally, it takes less time to make excursions to other files.

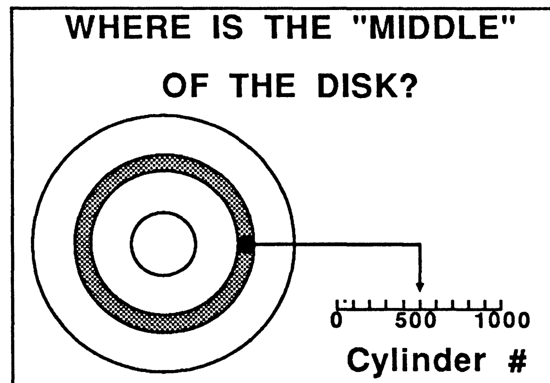



*Figure 7*

Once you have determined the important files and you've decided on the best location for them, how do you get them there? VMS is, unfortunately, somewhat incomplete in its ability to allow you to accomplish this.

Creation of template files using File Definition Language (FDL) can be used. The POSITION keyword allows you to declare where on the disk to locate your file. Figure 8 below shows a minimal FDL file that could locate a specific file precisely at block 25000 of a disk. But FDL cannot help you if some file is already located at the desired location. It also is unable to define and move directory files, so their placement cannot be accomplished at all using this method.

[1]For more on directory cache performance see below.

```
PLACING A FILE USING
    CREATE/FDL

FILE
         NAME                  "NAME"
AREA 0
         ALLOCATION            <size>
         EXACT_POSITIONING     YES
         POSITION LOGICAL      25000
```

*Figure 8*

The lack of standard VMS tools was one of the prime motivations for the development of third-party software to perform such functions.

## FILE FRAGMENTATION

The next performance factor I would like to discuss is the level of file fragmentation: i.e., the breakup of single files into many smaller pieces that are distributed across the disk. This is a normal outcome during normal VMS operation for disks that undergo file creations, deletions, extends and truncates (which includes just about every disk ever attached to a VMS system). All of these operations are carried out by the XQP, so as to be completed as *quickly* as possible, rather than spend undue time and overhead finding the *best* available location. This is necessary because it can sometimes take a considerable amount of extra time to determine the *best* place to locate a new file. Generally, the user wants a file allocated as *fast* as possible. An undesirable side effect, however, is that individual files become scattered across the entire surface of the disk. Believe it or not, a new file will be broken into multiple fragments in many cases even if there's room available to allocate it contiguously.[2]

Fragmented files slow disk performance due to two effects: **split I/O** and **window turns**. Each is discussed briefly below, along with the best method to measure each.

Split I/O occurs when a portion of a file that you wish to read or write is not located on the disk adjacently. This is not apparent to the application program because VMS supports the notion of *virtual* disk storage. The thinking that the blocks of data files are located together is only an illusion, however. File segments that a user thinks are contiguous are sometimes split across the disk in a thousand pieces. To support the illusion, VMS performs multiple I/O operations behind the scenes to satisfy what the user thinks of as a single operation.

How much does split I/O affect your system? It depends on your system, but it is not unusual for split I/O to lengthen processing by 300% on a badly fragmented file or more.

Unfortunately there is no convenient utility in VMS to measure the number of split I/Os. You can, however, measure the total number of split I/Os that have been processed by your entire system since it was booted and thus collect at least a coarse measure. This procedure simply involves using ANALYZE/SYSTEM and examining a special statistics cell maintained by VMS. The procedure for this is shown below:

```
$ SET PROCESS/PRIVILEGE=CMKRNL
$ ANALYZE/SYSTEM
SDA> EXAMINE PMS$GL_SPLIT
PMS$GL_SPLIT: 00005243 "8..."
```

This example shows that 5243 hex (or 21,059 decimal) split I/Os have occurred on this system since it was booted. By observing this statistic over time, and subtracting the difference in the PMS$GL_SPLIT value, you can begin to know if split I/O is a problem on your system and further understand what particular user operations might be leading to it.

File fragmentation also leads to a second form of avoidable overhead: window turning. This is an operation the XQP performs when it does not already know the whereabouts of a requested file block. When this happens VMS stalls the user temporarily and reads the volume index file for the necessary retrieval information.

In practice, high window turn rates are unusual but can be measured with the command:

```
$ MONITOR FCP
```

By the time the system window turn rate exceeds five per second, though, disk files are probably so highly fragmented that the split I/O rate has soared, too. In such a case, severe performance degradation is almost always observed.

A specific file's level of fragmentation can be ascertained with the DUMP/HEADER command. This utility produces a list of each fragment and shows where each is stored on the disk.

In addition, we have written a utility we call FRAG that produces a report of the fragmented files on a given disk, sorted so that the worst fragmented files (the ones we're most concerned about), appear first. (See Figure 9.) Notice that for the worst fragmented files, a number is shown inside parentheses after the number of fragments: this designates the number of additional file headers (known as *extension headers*) needed by the XQP to store all the retrieval information for the file. These extension headers are generally required when a file has a hundred or more fragments.[3] The presence of extension headers is also a strong indication that moderate to severe file fragmentation is present.

**FRAG Displays the Worst Fragmented Files**

| Fragment count (ext) | Worst 100 Fragmented Files: |
|---|---|
| 252(2) | [00,001]BCKMGR.LOG;348 |
| 101(1) | [SYSTEM]SWAPFILE.SYS;1 |
| 27 | [001,001]BCKMGR.LOG;347 |
| 22 | [001,001]BCKDUB1.LOG;3 |
| 22 | [SYSTEM]ERRLOG.SYS;1 |
| 21 | [WEF]DISKIT_DSU.EXE;10 |
| 18 | [DMP[TEST0.PHYS;2 |

*Figure 9*

A number of remedies can be used to correct file fragmentation. One simple way is to use the COPY command. In the process of copying a file from one location to another, the file may become less fragmented--depending on the nature of the free space available on the volume at the time. CREATE/FDL allows much more control in allocating files as contiguous (as well as placing a file at a particular location on the disk, as mentioned before). But CREATE/FDL is still not very convenient for use on regular basis for all the files that need to be defragmented. Again, this formed the motivation to develop special software specifically designed to perform file defragmentation.

## FREE SPACE COMPACTION

The organization of a disk's free space can be important, as well, for two main reasons: First, when a new file is created, VMS has no choice but to allocate whatever space is left on the volume for the file. If the free space is fragmented at the time that the allocation request is received, VMS has no choice but to fragment the file from its very first allocation.

Second, some application programs require contiguous free space for operation, and lack of contiguous free space is a constant nemesis.[4] On such systems, the free space must be carefully managed to allow the application software to do its job.

The traditional technique for dealing with free space fragmentation is to take the disk out of service and use the BACKUP utility to perform a time consuming backup and restore operation, yielding yet another explanation for the development third-party software to ease this inconvenient management chore.

## DISK CHARACTERISTICS

Last, but not least, the performance characteristics of the disks you

---

[2]This is largely the result of the operation of the extent cache, discussed in more detail in a later section.

[3]Extension headers can also appear if you use ACLs or RMS journaling extensively on your system.

[4]The most notable example of this is Intergraph VAX systems which require contiguous free space for hardware-mapped graphic *design* files.

have on the system, and those that you are considering purchasing for your system in the future, have significant bearing on disk throughput. There are a vast array of features and capabilities (see Figure 10), offering performance that can range within an entire order of magnitude. The most important performance parameter is average access time.

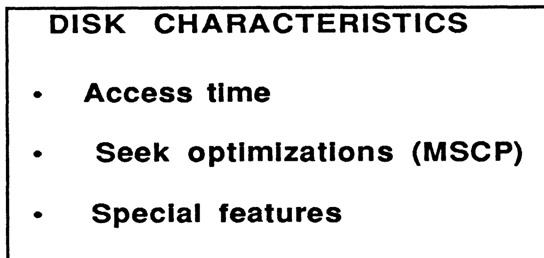



*Figure 10*

Some disks offer special performance-related features as well. Digital disks that use the MSCP protocol incorporate a feature known as "elevator seek optimization." This feature comes into play on disks that handle a continuous and heavy load of I/O (e.g., a minimum of three pending requests at all times). Some more exotic disks offer integral RAM caches for speeding access to disk blocks that are used often and repeatedly.

Most disks, nowadays, also incorporate error correction features. This is a big advantage in making drives more reliable, but something you may not have thought about is that error correction can slow throughput. This sense of security can lull the unsuspecting system manager into believing that all is well with the disk when, in fact, it is progressively nearing a day of disaster. If you've ever grappled with an apparent performance loss on a disk and a user innocuously asked you why there's a five digit number in the error count field of a $ SHOW DEVICE display, you may have just solved your performance problem.

## CACHE TUNING

The last area of disk performance optimization I'd like to talk about is the tuning of the file system cache. This is a subject that deserves your attention, especially if the system you manage is not typical, because this is the only (and rare) case for which VMS comes already optimized. As you will see, VMS does *not* make all-knowing adjustments to cache sizes for you[5] -- you must incorporate your knowledge of how your system is configured, and applied by your users, to effectively maximize your system efficiency.

If you call Digital and ask their advice on cache adjustments they're very likely to recommend that you just run AUTOGEN and accept its sole advice. Many software support specialists consider it absolute heresy to try to improve on these calculations. However, in reality, AUTOGEN is quite primitive, and works best if it works along with *you.*

AUTOGEN bases all of its decisions concerning cache on two factors, and is oblivious to other important information: 1) the BALSETCNT parameter (the maximum number of processes concurrently kept in memory), and 2) the number of disks on the system. Both of these are a good starting point-- after all, the higher your balance set, the greater your probable need for cache space to accommodate the buffers for file system requests. Likewise, the greater the number of disks, the greater the probable need for cache space for processing each. But this is fallible and simplistic.

The need for cache space is not simply based on the number of users on the system-- it depends, more accurately, on the number of file requests the system must process. One user who repetitively opens & closes files represents a far greater load to the system than ten who stay logged in to the same application program all day. Also, counting the number of disks does not necessarily reflect the correct amount of cache space needed. The number of disks that are actually *mounted* is important. What happens if you have only one disk and it is heavily used? What happens if you forget to run AUTOGEN after you've added disks? AUTOGEN doesn't take into account MSCP-served disks on other nodes of a cluster, yet they occupy cache space just as do local disks. It is quite possible for your cache space to be underestimated for a number of such reasons.

So, if your users submit a higher than average number of file system requests (e.g., a typical educational site with many interactive student accounts), or if you are in a cluster, you may benefit significantly from increasing the size of your caches. Even if you consider your system to be normal, you may be able to achieve noticeable improvement through cache enlargement.

Why not increase the cache sizes infinitely? If your system is low on memory (i.e. page faults heavily) then cached data ties-up paged pool. This may lead to heavier system working set paging or user process paging. On a typical 11/780 with 3 RM05's and a balance set of 30, AUTOGEN selects a cache of 512 pages, or about a 1/4 of a megabyte. If the 780 has less than 4MB of total memory-- its probably best to stick with AUTOGEN's computation. If you have 6MB or more, though, you can probably increase performance by increasing the cache size.

The best way to determine if you have enough free memory to consider dedicating some to additional cache is to monitor the size of the free page list. If substantial numbers of free pages exist in excess of the SYSGEN parameter FREELIM, then you probably should be considering an increase to the size of cache.

Increased cache size can also be detrimental due to another effect: greater numbers of cache buffers imply increased searching, and higher demands on the CPU. This is generally not an observed problem, however, since most machines have an ample surplus of CPU time (usually several percent) to use for cache management. This may become even less of a concern in the future as more and more VMS systems operate in the multi-processing environment. On such systems, surplus CPU time should be even more available.

In summary, with additional unused memory on the FREELIST,[6] and several percent extra CPU resources, there is no detriment to doubling or quadrupling the cache sizes computed by AUTOGEN.

Note that if you increase the demand on paged pool by increased the cache sizes, you may be defeating yourself unless you also adjust the system working set size. It would be ironic to create a larger cache only to have most of it paged out when most needed! This is AUTOGEN's main function in life: to ensure that the SYSGEN parameters are consistent with one another. The proper procedure to ensure this is always to make your trial adjustments in MODPARAMS.DAT, and then run AUTOGEN to allow it to adjust any other affected parameters and so be consistent with your changes. In this way, you and AUTOGEN can work together to achieve better performance.

I will be the first one to admit that I'm far from knowing everything about VMS's caching system, but I think that by sharing some of my experiences with you I can *demystify* the various file system caches so you can better understand their functions. This can help you predict the changes in performance that should result from tuning.

I should mention how the following benchmark numbers were collected. First, the benchmarks were taken with everyone off the system. This made it possible, in some cases, to use *elapsed* time as the performance indicator. Second, in order to control the caching parameters, I mounted the test disks test with the /PROCESSOR=UNIQUE qualifier. This sets up a private cache just for the disk under test. (Normally, the XQP shares the cache area across *all* disks mounted on the system for easier management of cache memory.)

One last note: if you run the SYSGEN utility and use the SHOW/ACP command, you receive a tidy list of all of the cache control parameters. In addition, note that most of these are "dynamic" parameters. This means that they can be adjusted while the system is up and running. This is true in one sense and false in another. True, the XQP will honor changes you make to the caching parameters the next time it builds a new cache, but this is false in the sense that it will not effect any caches that are already built. This means that, in general, the parameters are not really dynamic because the single central cache is built one time only-- when the system is booted. (Which was fine for my purposes because I wanted to rebuild the cache every time I mounted a new test disk anyway.)

## THE BITMAP CACHE

There are actually two "caches" associated with the volume allocation bitmap, BITMAP.SYS (see Figure 11). A BITMAP.SYS file on every disk denotes the blocks are currently free or in use. The BITMAP is the on-disk record of what *is* and *is not* currently in use on the disk.

[5]Digital *has* confirmed that in the next major future version of the VMS (V5.0) AUTOGEN will adjust cache sizes somewhat, based on actual user activity. However, this will be done in a conservative fashion, and will preferentially reduce cache sizes rather than expand them.

[6]See $ MONITOR MEMORY for this statistic.

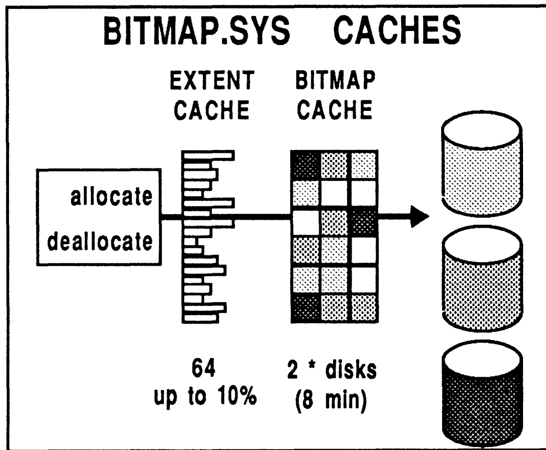



*Figure 11*

The file system keeps blocks of the bitmap in memory (in the BITMAP cache) to speed new file allocation and file extension. When an allocation request is received, the system must scan this bitmap, looking for enough free bits to satisfy the request. If the block of the BITMAP that is contained in cache doesn't have sufficient space, other blocks of the BITMAP file are read and scanned until the request can be satisfied.

Because this scanning process itself can require significant time, the XQP keeps a list of recently deallocated space. It uses this list first, if it can, to satisfy new allocation requests. This list is called **the extent cache**, although this term is somewhat of a misnomer because it's not a cache in the traditional sense at all. More aptly, it might be described as a "look-aside" list.

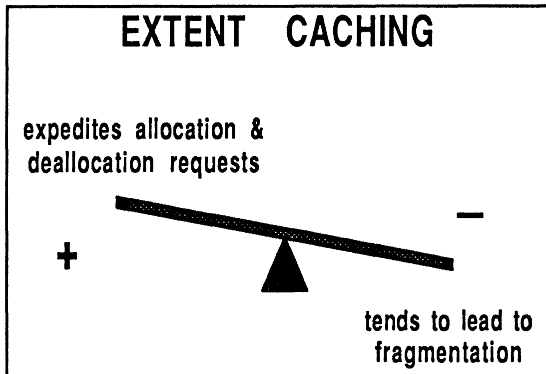



*Figure 12*

An interesting property of the extent cache is that it tends to create file fragmentation. Since the extent cache is checked *first* for space to allocate to new files, the space allocated may not be the best suited for that file. It is not uncommon to see the XQP allocate several non-contiguous pieces of free space from the extent cache for a new file, even when enough space would have been available for a contiguous allocation, if the XQP had bothered to check the bitmap file.

I performed a benchmark to see the effect that the size of the bitmap cache (controlled by the ACP_MAPCACHE SYSGEN parameter) had on allocation and deallocation requests. I chose a disk that would not be accessed by other users during the benchmark. The disk had about 15,000 free blocks that were known to be dispersed over the entire disk. I then allocated and deallocated a 15,000 block file 30 times and measured the total time required. The results of this benchmark are presented in Figure 13.

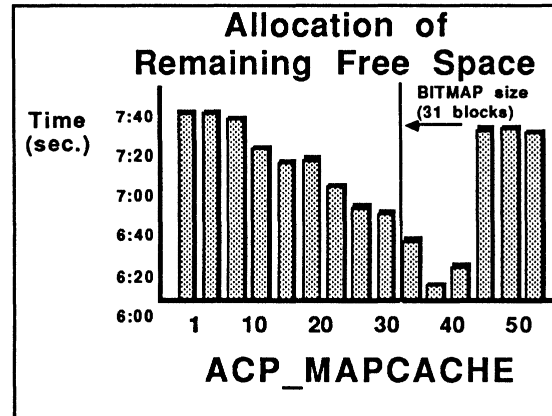



*Figure 13*

As can be seen from Figure 13, the time decreased as more and more blocks of the BITMAP file were allowed to become resident in the cache. For some unexplained reason, however, the performance continued to improve even past the point where the entire bitmap file should have become resident (this disk's BITMAP file was 31 blocks in size).

AUTOGEN does not adjust the extent cache size and limit. These usually set at 64 extents and 10% of the disk size simply because these are the default values that come with all VMS and MicroVMS systems. AUTOGEN allocates 2 blocks per disk it sees attached to the system for ACP_MAPCACHE.

My general recommendation is to allow a larger ACP_MAPCACHE value than the one that AUTOGEN computes. It is highly likely that the system will need more than two blocks of each bitmap file, especially if space on the disk is nearly exhausted (which also induces more frequent and more extensive bitmap file searches).

When free space becomes scarce, allocation requests become more difficult to satisfy because the XQP must check all blocks of the bitmap file to find a sufficient amount of space. This is another reason why the extent cache has become an integral part of the bitmap caching system. The extent cache alleviates the need for the system to scan the bitmap file for space, especially if the same size allocations and deallocations are routinely requested.

## THE HEADER CACHE

There are two caches for file headers from the INDEXF.SYS file (see Figure 14) that are similar to the BITMAP caches. These caches are referenced for many system operations, including file creation, deletion, extension, truncation, access, deaccess, and modify. **The header cache** contains copies of recently read file headers (including extension headers). The number of headers, for all disks that will fit, is controlled by the SYSGEN parameter ACP_HDRCACHE.

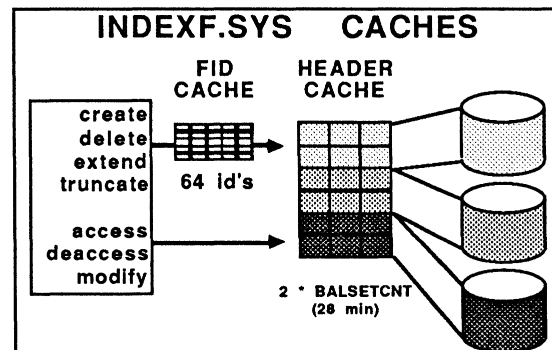



*Figure 14*

By default, AUTOGEN sets the size of header cache to 2 headers per process in the balance set. This implies that each user will place a demand on only 2 file headers. This default is probably too small in most cases and could be doubled without any troublesome side effects.

Just as the extent cache is really a list of recently deallocated storage in the bitmap file, the file-id, or "FID" cache is a list of recently deleted headers that are available for reuse. The FID cache enables VMS to quickly assign a new header when one is needed without having to scan the index file's header bitmap. As with the ACP_EXTCACHE, AUTOGEN does not adjust ACP_FIDCACHE, which controls the number of headers that are listed in the FID cache. Instead, this parameter is generally set to 64-- the default value that is shipped with all VMS distributions.

I performed two benchmarks to confirm the effect that the FID cache and header cache sizes should have on file processing performance. Figures 15 and 16 summarize these results.

To test the FID cache, 30 files were created and then deleted. The time required to complete this was measured and plotted for various sizes of ACP_FIDCACHE. Approximately a 10% improvement in create/delete performance resulted from the availability of enough FIDs to accommodate all files to be created. A significant benefit was seen by allocating just a few FIDs to the FID cache. Then, a tapering but increasing amount of further improvement was seen as more FIDs were added. The conclusion I have drawn here is that a relatively small FID cache will deliver a significant level of optimization. The default ACP_FIDCACHE value of 64 is quite generous, however, and probably quite adequate for most systems. It needs to be increased only for large systems with a great deal of expected file creation and deletion.

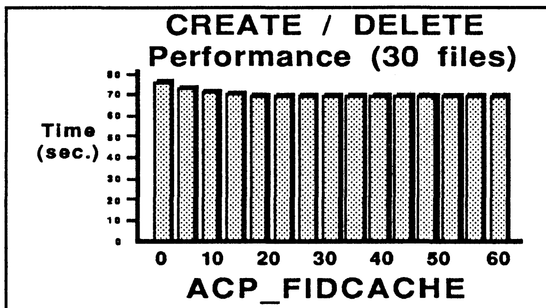



*Figure 15*

The size of the header cache, on the other hand, should affect the performance of many file operations. This is due to the near universal need for file headers to complete file system operations. The SYSGEN parameter ACP_HDRCACHE controls the size of this cache.

The header cache size was tested by opening and closing 18 pre-existing files, measuring the time required to complete this task for various header cache sizes. Note that the smallest possible header cache size is 3 headers-- this is a minimum value imposed by VMS. The size of the FID cache was held at zero during this benchmark to remove any effect it might have had on the results. As expected, the performance improved markedly at the point where all of the required headers could be expected to be loaded in cache, rather than VMS having to read these from the volume index file.

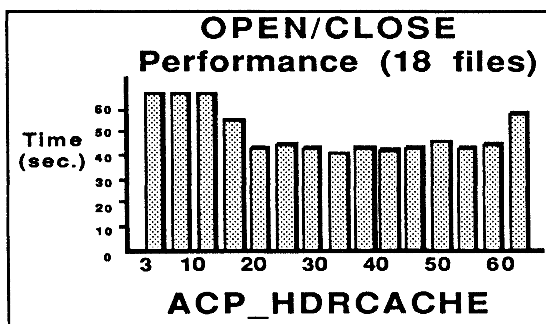



*Figure 16*

An unexpected result was observed as the size of the header cache was increased to even greater sizes, however. At about 60, nearly all of the performance benefit of having any header cache at all seemed to disappear. Not shown on the graph, this degradation continues to be observed till about a size of 80 when the performance again improved as if all headers were installed in cache.[7]

## DIRECTORY CACHE

The directory cache is constructed slightly differently than either the bitmap or header caches. The main directory cache contains actual blocks of directory files. These are referenced by VMS as it performs file name lookups.

It is important to note that when a file is opened, such as the file [SYS0.SYSEXE]SYSUAF.DAT, VMS must not only find the SYSUAF.DAT file name in the [SYSEXE] directory, but must also find the SYSEXE.DIR directory name in [SYS0], and the SYS0.DIR directory in [000000]. Therefore, quite a significant amount of directory processing is implicitly required owing to each subdirectory level used.

The directory cache is used for all operations that locate files by the file names: this includes file creation, deletion, renaming, access and lookup. The directory file itself is organized as an alphabetically sorted list of all file names stored in the directory with the file-ID of each file stored in it.

In order to speed directory processing, a **directory index (DINDX) cache** is used. Like the FID cache and the extent cache, the DINDX cache is not a true cache, nor is it like a "look-aside" list. Instead, it contains a small portion of data from each block of the pertinent directory file. A separate DINDX cache is built from each directory file as the file is used. Its function is to speed up certain aspects of directory processing. Figure 17 depicts the directory and DINDX cache structure.

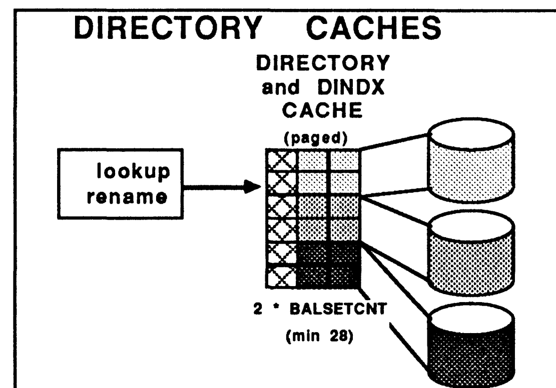



*Figure 17*

The use of the directory and DINDX caches are best illustrated by an example:

A telephone book is similar to a directory file in that both are alphabetically, sorted lists of names. Generally, when searching for a particular name in a telephone book, you use the index printed at the top of the page, which contains the first and last entry for that page. You can then scan the phone book fairly quickly without searching the entire page. Once the correct page is determined, you can then search the page in detail.

The DINDX cache is analogous to the index printed at the top of each page in the telephone book. It contains a copy of the last file cataloged by each block of its directory file. This list allows the XQP to determine quickly which block of a directory file contains a particular file name without having to go to the trouble of reading other blocks of the directory file. The DINDX cache for a particular directory is built *on-the-fly* the first time a directory is scanned for use. Any time the directory changes, such as by the addition of a new file entry, the DINDX cache is discarded and rebuilt on the next use.

[7] A representative from the VMS Development Group confirmed that some aberrations in the cache performance are currently under investigation. I hope to present a better explanation of this phenomenon at the Cincinnati DECUS symposium in May 1988.

For instance, assume there is a four block directory that catalogs 26 files with the names "A" through "Z." If you ask VMS to locate file "Z," it first reads block one of the directory file. It then examines the last entry and notice that the file "F" is the last file cataloged in that block. Before moving on, however, the XQP stores the fact that the last file name in block one is "F" in the DINDX cache. The search then continues. Block two, three and four are read in search of file "Z." Each time the last file name of each block is inserted in the DINDX cache for future reference. Finally, file "Z" is located and VMS returns to the user.

Next, assume that we ask VMS to look up file "Q." This time the XQP notices that file "Q," if cataloged at all, is in the DINDX cache and should be located in block three of the directory. So VMS goes directly to block three without re-scanning blocks one and two.

AUTOGEN sets the size of the directory cache (parameter ACP_DIRCACHE) to at twice the balance set count or 28, whichever is greater. It sets the DINDX cache (parameter ACP_DINDXCACHE) to be 25% of this size (in addition to the directory cache). These values are likely to be too small for many systems, especially those that have a large number of directory and subdirectory files, or have very large directory files.[8]

Figures 18 and 19 demonstrate the effect of directory and DINDX cache size on directory processing. For the benchmark, a directory was created with 3 subdirectories. Each of the subdirectories contained 3 subdirectories of their own, and each of these contained 3 subdirectories of their own. This created a total of 40 directory files. (Note that all directory files were one block in length.)

Finally, a file was created in each of the bottom level directories to create a structure four levels deep. The time taken to complete a full scan of the entire subdirectory tree was measured under varying cache sizes.

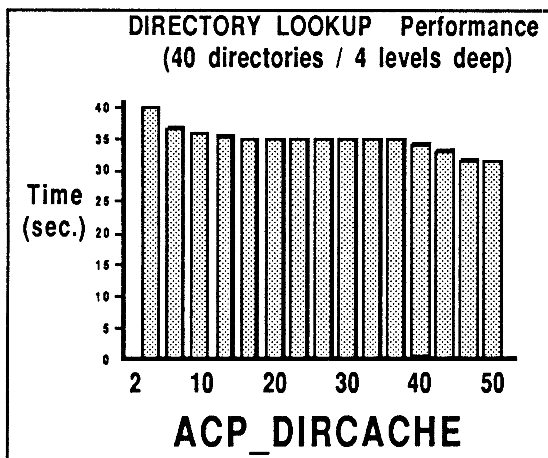



*Figure 18*

For the first benchmark, the size of the directory cache was varied but the size of DINDX cache was held at a minimum constant size. Performance improved in two stages until the point at which all 40 blocks of all of the directory files should have been resident in cache. After about 10 blocks of cache were available, a noticeable performance improvement of 8-10% resulted. When the cache reached a size where all directory blocks were resident, another 5% improvement was observed. The reason for the two-stage nature of the observed data is presently unknown.[9]

Next, the same directory scanning benchmark procedure was applied while holding the directory cache size constant at a small, intermediate and large values of 2, 27, and 47. For each size of directory cache, the size of the DINDX cache was varied and performance differences noted. (Note that the smallest value for directory and DINDX cache tested was 2 blocks-- the minimum value allowed by VMS for these parameters.)

[8]Directory files greater than 127 blocks in length are to be avoided. Aside from general slowness resulting from scanning directories of this size, certain optimizations that RMS normally provides are precluded in this instance.

[9]Again, a topic to hopefully be explained in Cincinnati.

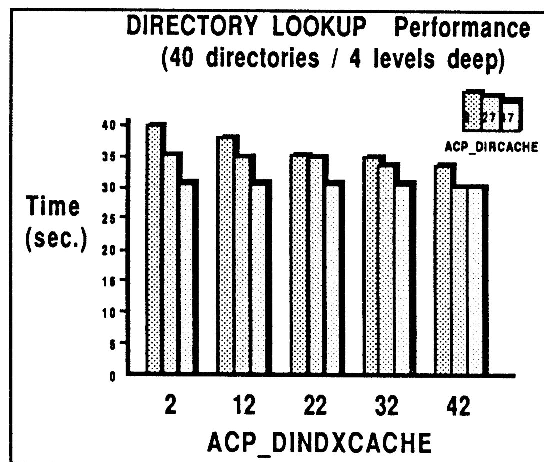



*Figure 19*

This benchmark showed the marked performance advantage provided by the availability of DINDX cache when the directory cache is constrained. Contrarily, the advantage gained by an ample DINDX cache was negligent when the entire directory tree was resident in directory cache.

The role of DINDX cache then can be seen to be quite important in the case that not all of the directory file is resident in directory cache. This is an important conclusion because it most closely describes most VMS systems. In the normal case, there are far more directory blocks competing for residency in the directory cache than can be accommodated. This is because users generally access a large number of directories, but also because directories are relatively large files, sometimes containing hundreds of blocks. The DINDX cache reduces greatly the performance degradation that otherwise would be felt if only directory cache were available, and thus is an important tool.

## THE EVOLUTION OF STRUCTURING TOOLS

As a last topic, I would like to survey the evolution of disk structuring software. It is interesting to review how software of this type originated, and what features evolved later. In the process, I want to share with you some of the more advanced techniques that have been developed recently.

### THE 1ST GENERATION

The first efforts at disk structuring were accomplished with no more than glorified COPY programs. The utilities of this type, by-and-large, never saw widespread distribution and were mostly only an experimental attempt to understand the problem better and achieve at least *some* improvement.

The primary aim of such 1st Generation software was to make fragmented files contiguous. Unfortunately, however, they gave up rather easily if free space was unavailable-- a case that is quite common in the real world. Other subtle problems also needed to be conquered. In the worse attempts, the files were processed without considering that users might be in the middle of processing, causing various undesired conflicts. In the better attempts, the files were locked during processing, but that carried some undesirable behavior too, as users unaware of the restructuring software were locked-out from their files for periods of time.

One other underestimated pitfall was the fact that restructured files received new file-ID numbers during defragmentation. This caused incompatibilities with system utilities that tracked files by file-ID number instead of by name (such as the batch and print symbionts).

Free space tended to become more and more fragmented as old fragmented files were copied to contiguous free space over and over again. Eventually, unless the disk had substantial amounts of free space, insufficient contiguous space prevented full defragmentation.

Other problems included excessive use of system resources while running, poor (nonexistent) reports of what had been done, and very limited (or nonexistent) error recovery.

**1st GENERATION TOOLS**

| benefits | drawbacks |
|---|---|
| • defragmented files | • gave up easily |
| | • file locked |
| | • FID's not preserved |
| | • heavy system load |
| | • poor reporting |
| | • incompatible access |
| | • free space frag'd |
| | • err recovery limited |

*Figure 20*

## THE 2ND GENERATION

Next came what I categorize as the 2nd Generation utilities (Figure 21). Software in this category is characterized by its greater sophistication. Many of the currently marketed disk structuring software belongs to this classification.

The benefits of the 2nd Generation tools are manifold. They accomplish the goal of defragmenting files, but they also address the need to consolidate fragmented free space. This is done by packing files together to create a (usually one) large area of free space. In addition, most utilities solve the problem of preserving file-ID numbers. To be easier for the already harried system manager to use, they are equipped to be scheduled and run in BATCH when users are likely to be less active. Lastly, they provide better reports about their activities for audit purposes.

Some drawbacks still remain, however. Most importantly, these utilities lock the files they are working on for at least a short period during processing. This is a disadvantage because it can cause an unsuspecting user to receive a file access conflict error that the user would not otherwise expect. This can be a big problem for late night users who attempt to work during the time scheduled, by the system manager, for running a disk structuring operation in batch. And no system manager needs this sort of frustration cropping up on his desk in the morning.

A second unsolved problem for 2nd Generation utilities is that scheduling the execution of such a utility in the BATCH stream is somewhat simplistic, and the utility may not be able to handle situations that are incompatible with its operation without an operator's assistance. These utilities also generally impose a heavy load on the system while running. This defeats, to some extent, the whole purpose of enhancing system performance through optimizing the disk. And so the cure is almost as painful as the affliction. Error recovery in some still leaves something to be desired, depending on the thoroughness and testing of the particular implementation.

**2nd GENERATION TOOLS**

| benefits | drawbacks |
|---|---|
| • defragmented files | • file locked |
| ... also free space | • incompatible access |
| • more intelligent | • heavy system load |
| • FID's preserved | • err recovery limited |
| • can run in BATCH | |
| • operation reports | |

*Figure 21*

## THE 3RD GENERATION

The most recent arrival on the scene is disk structuring that has improved significantly enough to be considered the 3rd Generation tools. The hallmark of this software is its full *transparency*.

This type of utility has the advantage of scheduling flexible and appropriate times for disk structuring operations. The system manager should be able to say, "I want the disk to be restructured only between the hours of midnight and 2:00 AM on Saturday nights only."

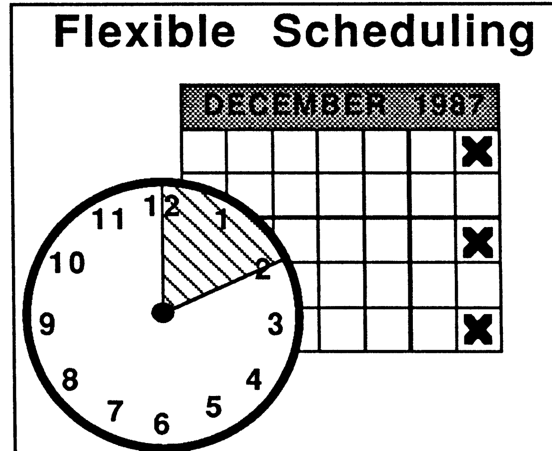



*Figure 22*

In addition, the system manager should be able to place constraints on the operating schedule to preclude activity during periods when the system is being used by others, even if that is known in advance. Thus, not only should it run during specified time intervals, it should also monitor system resources and scale-back or suspend operations if a preset amount of CPU or I/O is observed.

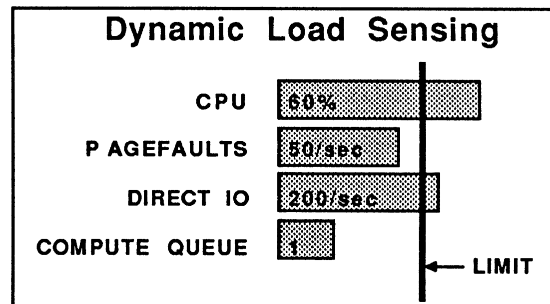



*Figure 23*

In the 2nd Generation tools, locking files for exclusive access was the common method of determining if a file was open by any other user or application on the system. This is not a sufficient technique, however, due to a seldom[10] but sometimes used file access option called "NOLOCK" access. With this style of VMS file processing (and suitable privileges), the system grants access to a file, despite any other users that have requested their own exclusive access to the file. Worse yet, with the NOLOCK option, no other access attempts are rejected. Thus, a restructuring utility that expects to determine if a file is open by requesting exclusive access to a file, does not properly detect the operation of applications that can use this option. The incompatibility of most 2nd Generation tools with NOLOCK files is a common oversight.

---

[10]INGRES file access, for example, is incompatible with this technique and can lead to database file corruption because the structuring tool cannot determine that INGRES has the file open.

The 3rd Generation tools provide a solution for this problem by using the VMS distributed lock manager to determine conclusively all types of file access, even instances of hidden NOLOCK file access.

Another significant problem to be solved was how to be sure that the restructuring tool could safely update a file's retrieval information after moving a file. This had to be done without allowing a user to come along and attempt a file access. 2nd Generation tools (see Figure 24) prevented this possibility by locking up the file with exclusive access. But again, this is a faulty technique if users implement NOLOCK access files: the tool would first satisfy itself that no user presently had access to the file in question, and then request exclusive access from the VMS and, when this was granted, the structuring tool would perform its processing, conclude, and then release its exclusive access.

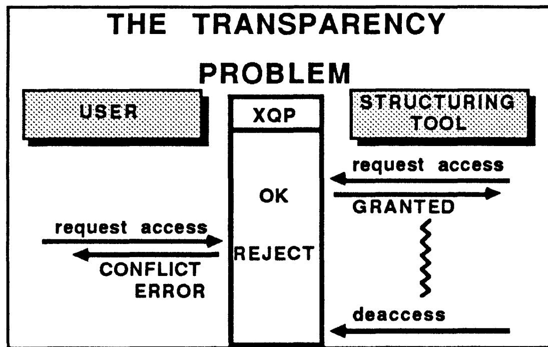



*Figure 24*

This mode of operation could cause file access conflicts, however, for users whose applications might attempt to access a file while the structuring tool had data in transition. This occurrence was statistically rare, so most users have not experienced or even realized the potential for a problem. The 3rd Generation tools offer an elegant fix for this problem, without any possibility of conflict with a user.

In the 3rd Generation tools, the VMS lock manager is used in conjunction with the XQP to create a file access notification AST. This technique uses existing VMS capabilities that have never before been used in this fashion. In this design, the structuring tool first ensures that a file to be restructured is not open anywhere on the local system (or on the VAXcluster, if appropriate). An exclusive mode access is then requested from the XQP and a *special* option is set-up via the distributed lock manager. A notification AST routine is defined to be called by the lock manager in case any user or XQP activity should attempt to access the specific file.

If a user *should* attempt access, the 3rd Generation structuring utility is notified immediately. In almost all cases, the utility can immediately relinquish its access to the file and the user is granted their access with complete transparency. Then, the structuring tool can abandon any intermediate processing that was interrupted and try again at a later time when the file is inactive.

In the rare case, when a user is attempting to access a file at the most critical phase of processing (when the structuring utility has initiated the update of the file's retrieval information), the utility simply continues processing until all critical processing is complete and then relinquishes its lock on the file. The user simply waits a few extra milliseconds to gain access to their file; *they receive no access conflict errors*. The benefit of this type of file locking is that users can transparently run their applications and never receive an unexpected error message due to the structuring tool. This improved technique is a major development in the evolution of the disk structuring tools.

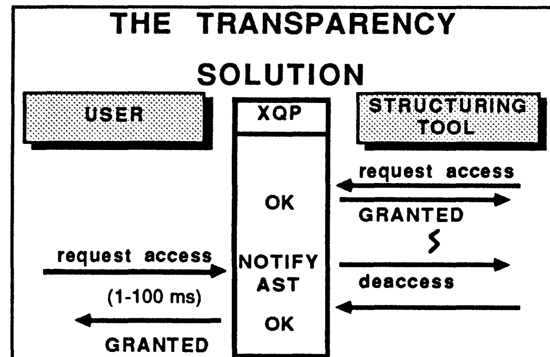



*Figure 25*

To summarize, the 3rd Generation tools offer all the benefits of 2nd Generation tools but are even more intelligent and sophisticated in their features. They support a greater breadth of *controls* to allow the system manager a great deal of flexibility in adapting the tool to his specific environment, and they are completely *transparent to users*.

Some of the areas *not* addressed by the 3rd Generation tool are the defragmentation of files that are open during processing. Also, no current structuring tools address the need to internally restructure RMS indexed files. Both of these areas may be addressed by future software offerings.

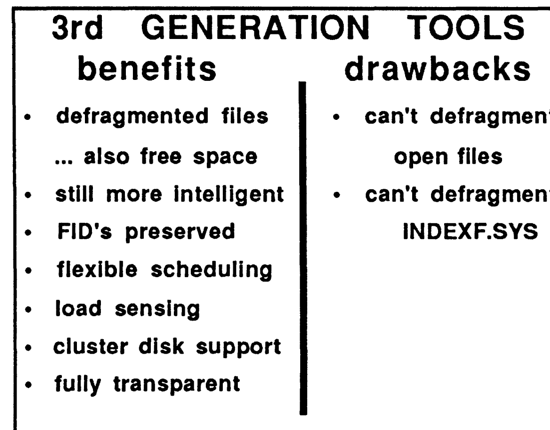



*Figure 26*

## SUMMARY

We began by talking about five basic factors that affect disk performance: file placement, file fragmentation, free space fragmentation, inherent disk characteristics and the tuning of the cache system. In understanding the way VMS organizes the disk, you can come closer to diagnosing where your system is experiencing bottlenecks and make the most appropriate adjustments.

Third-party disk structuring utilities have evolved to help with these management tasks, all the way from methods of using standard system utilities solely to having sophisticated utilities that take into consideration the need for full user transparency and system load management.

The author, Wef Fleischman, can be contacted at:
UIS/Software Techniques, Inc. - 6600 Katella Avenue, Cypress, CA 90630
714/895-1633

# COPING WITH FULL DISKS

Malcolm Dunn
UIS/Software Techniques, Inc.
DECUS Anaheim, Fall 1987

### ABSTRACT

The management of disk (online) storage can affect the performance of any VAX/VMS system, while offline storage presents its own set of organization and security problems. This article discusses the problems of full disks on the typical VMS system, and some of the possible solutions to cope with the situation. An attempt is also made to clarify the different methods for insuring data integrity: i.e., the differences between Backup procedures, tape management and file archiving, and the applications of each. The following text is a transcript of a DECUS presentation given at the Anaheim, Fall 1987 symposium.

## INTRODUCTION

Before my intra-company transfer to the United States, I worked with the DP division of the company, which has been in the DP business for over 20 years. We had a traditional timeshare bureau service with a communications network connecting users to DEC 10 and Xerox Sigma 9 mainframes. So we were concerned about data management for our clients and internal users. We supported our user community running a great variety of programs under several different operating systems, as shown in Figure 1.

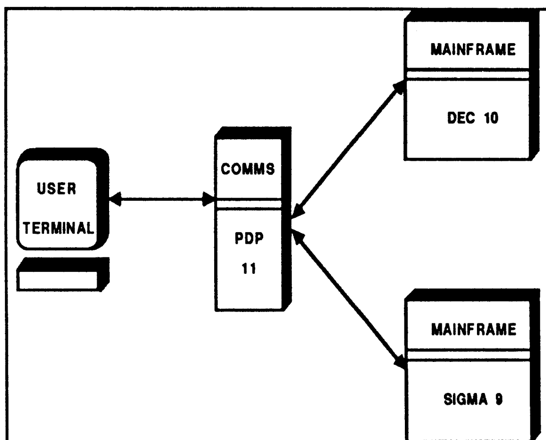



*Figure 1*

We faced the same problem that seems to be universal across virtually the whole computing community-- the headache of diminishing disk space. Every time we added more disk drives to our hardware an unwritten law would mysteriously come into play: the extra space was rapidly absorbed to the point where we were back where we started. This even seemed to happen on a system that gave no indication to the users what disk space was available to them. We wanted to avoid unnecessary disk hardware purchases. But we couldn't simply ignore the problem of full disks, and hope it would go away because experience showed that it would not. Users were understandably upset when they ran totally out of disk space.

One of the most effective solutions implemented was a file archive management system that ran on each of our computers. These systems gave us a fairly tidy way of migrating data between on and offline storage. The archiving software was tied to using tape as the offline storage medium.

When the hardware on-site started becoming a little long in the tooth, to say the least, we had to consider our next major hardware purchase. The decision was made to go for DEC VAX, and we purchased our first 11/780. This was great-- all these wonderful tools and facilities that our other systems did not have; a program developer's dream compared to them.

However, we soon realized that we were back to our old problem. How should we cope with ever increasing disk usage on the VAX? There appeared to be no convenient software available for purchase that would adequately sort this out. Would we have to throw more disk hardware into the system?

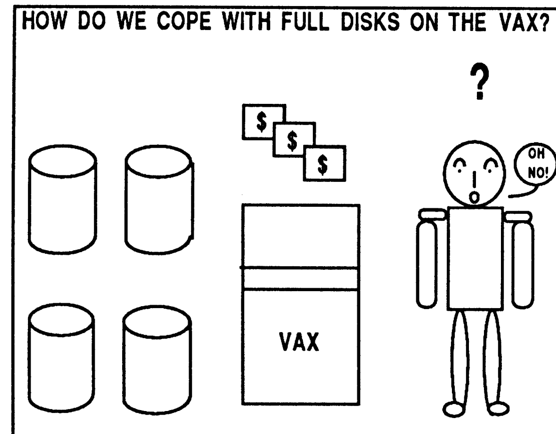



*Figure 2*

VMS' idea of multiple versions of files was terrific, but made it easy for users' disk usage to grow at an unexpectedly fast rate. Imposing strict disk quotas helped us keep users in check for a while. However, we had to find out what to do when those users genuinely needed to keep files that were presently inactive but were required at a later date.

We looked at the various ways we could control disk space on our VAX: through BACKUP, through a tape management system, and file archiving, as shown by Figure 3.

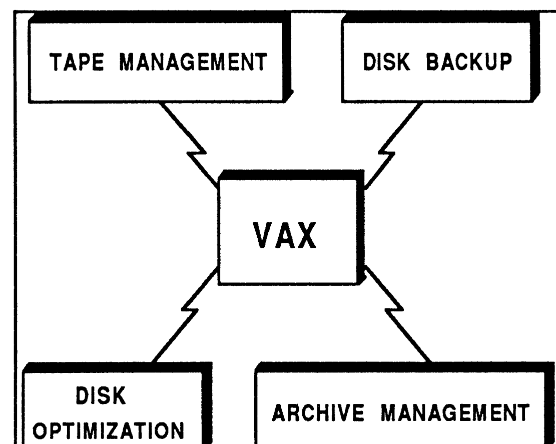



*Figure 3*

These were the choices we faced. I'll discuss what these choices really are, and what we actually found we needed for our users. Let's take each one in turn and explore what they will do for you.

## SYSTEM BACKUP

The regular system backup, that you carry out with the VMS BACKUP utility, comes as part of your VMS license, and it is therefore worth exploring what data management can be achieved with this utility.

First of all, when you use the BACKUP utility for regular file saves, you usually make your file saves on a disk-by-disk basis. You are also likely to save either all the files on each disk (the full backup), or only those files that have been modified since a given time, such as the last full save (the incremental backup).

The system manager usually coordinates BACKUP activity. He decides what to backup and how to arrange the backup schedules and tape cycles. The end users are not really involved, at least not until they lose data and come running to their system manager or operations department for help.

The backup tape cycles are just that-- cycles. You don't want to keep them indefinitely or you could end up with vast numbers of tapes over a period of years. So you perhaps have daily incremental cycles whose life span might be a week; weekly backups, each kept for 4 weeks; and monthly backups might be kept for 6 months or a year. A tape management utility helps you track both the tapes available for backup, and track the data that has been saved to tape.

However, although you have the data on tape (and even a tape management utility to help locate files), you may still have a major headache-- the problem of responding right away to user restores.

You know the situation: a user loses a file for whatever reason. He asks for help, but he may be unsure exactly when the file was last in good shape. He probably doesn't know what backup cycle the file is on because that is not his concern. So he pesters the operations department to restore his file, providing only vague information. The operators then typically have a manual task of looking up various listings for the file, checking which tape to mount, and finally running a job to restore the file. And if you site is security conscious, as most are these days, you'll probably want a manager's signature to authorize the restore.

Finally, BACKUP's main strength is in dealing with disaster situations such as a head crash. To wind-in one tape after another from an image save set is really not too difficult. It may be time consuming, but this is simpler than restoring individual files tucked away in the middle of a tape. BACKUP is better suited to recover from total disaster, such as the loss of a disk, than from specific accidents, such as a user deleting his own file or his own directory.

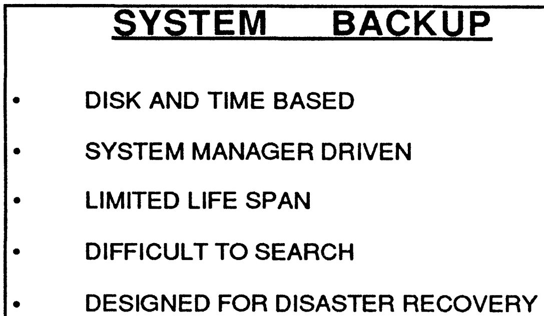



*Figure 4*

To summarize BACKUP, as shown in Figure 4, regular save operations are carried out on a disk-by-disk basis, with files selected for saves by data and time criteria. The system manager is responsible for performing BACKUP procedures. Information saved to a BACKUP tape usually has a limited life span, due to the nature of tape cycles. BACKUP tapes are difficult to search, unless you have a good tape management utility. And BACKUP is designed for disaster recovery mainly, rather than to create an organized "archive" for important data that must be kept.

So what else can you use?

## TAPE MANAGEMENT

Let's turn our attention to tape management systems. This type of system is not so much concerned with the software to read and write tapes as maintaining the tapes themselves. So a tape management system will have tapes as the basic units of interest. A tape management system is really an outer shell around the tape read and write software to control the environment. Let's now examine some of the features, benefits and drawbacks of these systems, as shown in Figure 5.

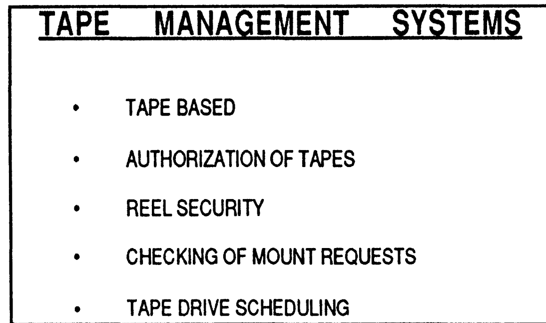



*Figure 5*

One of the features to look for in a tape management utility is tape authorization. You want simple procedures that let you put tapes into the system for use by whoever you wish.

A decent tape management system is concerned about tape security. When Joe Blow asks for tape XYZ123, you don't want to just mount the tape and hope that the tape truly does belong to Joe. The alternative to this is to manually look-up a file to see whether a request is legitimate. Your management software should actually make these checks, and blow old Joe away if he's trying to break security, without your involvement. That's real (and reel) security.

Once Joe gets past the first security check, the tape management utility also needs to ensure that the correct tape is mounted. Anybody could make a mistake and select the wrong tape from the shelf. So, these systems handle tape label checking once the tape is on the drive.

Another good feature you may find useful is scheduling of tape drives. A system that knows what devices are available, what devices are already in use, which tapes are already mounted and what drives can handle the density you require makes your life a lot easier, and your time more productive, especially where you have heavy tape usage.

A good tape management utility can help you automate a lot of the work done by BACKUP, and can give you a better method of taking files offline. However, what if you have several hundred users? Or have ten very demanding users? Or have users who cannot give you sufficient information about files so that you can use a tape management utility and restore the lost data? In any of those situations, which covers almost every type of site, you'll need something in addition to the tape management utility.

## FILE ARCHIVING

An archive system has some features in common with BACKUP and tape management systems, but archiving specifically addresses important issues. These issues are summarized in Figure 6.

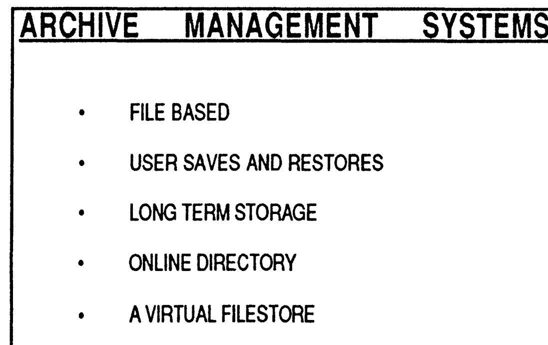



*Figure 6*

An archive management system focuses the interest on the user. It is a system that allows files to migrate between disk and an offline medium, such as tape. So it is user and file based, rather than disk based.

The end user, not the system manager, can be given responsiblity for deciding which files to save (or take offline) and which to restore (or put online again). He does so on whichever criteria he wishes. However, a good archiving utility allows the system manager to control exactly how much or little access is given to different users.

Providing safe and compact long term storage is the heart of the purpose of data archiving systems. Because such a system can save specific files as requested, rather than all files on a disk, it is more practical for selecting data that requires long term storage. Critical information-- statistics on chemical experiments, contractor's building plans, safety records, company information-- can be kept in one compact, easily accessed location.

A good archive management system allows you to see what is located on the offline media. You don't want the same frustrations of ploughing through listings in order to check what files you have and which tape they happen to be on.

In all, a decent archive management system can be summed up as being a virtual file store. BACKUP and tape management systems provide important features, but they are not adequate to meet the needs of larger sites, sites that depend on vital data, or sites with heavy user activity that require offline file management. This is why we selected archiving as the best solution to crowded disks: it gave us the means to clear-up the excessive online disk storage that we could not afford to erase. And gave our system managers more time for their work by automating this management.

But let's go a little further into the situation in England.

## DESIGN GOALS

As we wheeled in our first VAX, we knew that we needed an archive management system. The problem was how to design a solid, effective utility. A product that would really meet the needs we anticipated. So, when we got down to the actual design goals, we took into careful consideration the point of view of the end users, the system manager and the operators, as shown in Figure 7. We did not want anything which worked great for one group but at the expense of the others.

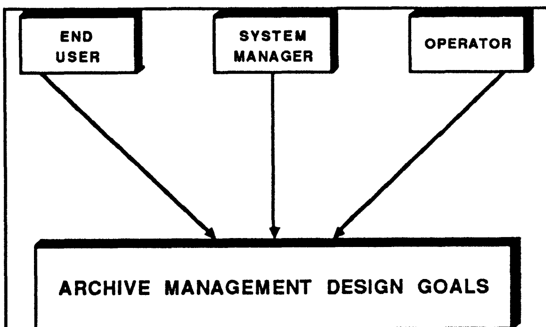



*Figure 7*

First, we evaluated the needs of the end users, and came up with the list of needs shown in Figure 8.

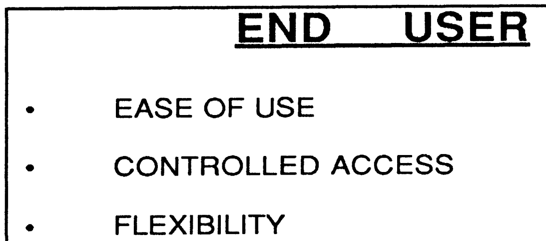



*Figure 8*

For the end users, we knew we had to create something that was easy to use. It's great to have bells and whistles for the guys who are able to make use of them, but there was no way we wanted to have our users complaining that they would *like* to co-operate but it was just too complicated to be worth the effort.

It was also our intention to give the users enough control to create and access offline directory structures that would suite their needs. But we had to give users assurance of sufficient controls that they would feel confident that, once offline, their data was just as safe and intact as if it were still online.

We also wanted a flexible utility. Haven't you experienced the situation where you put a lot of effort into some software project and within days of you handing over your precious baby, people come back to you complaining "I don't want it if it can't do such-and-such!"

The needs of the operator, shown in Figure 9, proved to be similar to those of the users.

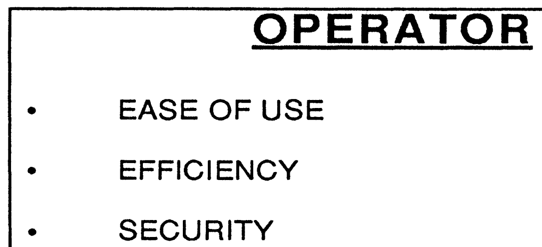



*Figure 9*

From the operator's point of view, we didn't want to build a system that put him at the mercy of the user. The system had to be easy and friendly and clear for operators as well as users.

Another important goal was to avoid inflicting upon operators yet another system that required them to run around looking up listings and looking for tapes every hour of the day. We wanted a system which was efficient in time and effort, letting the operator's schedule their day to work with the archiving system, not against it.

And security was again a big issue. The system needed to be solid enough to avoid mistakes as much as possible. A system that lets an operator mount the wrong tape and overwrite archived data is NOT designed to win friends and influence users.

But what could we put in such a system for the system manager?

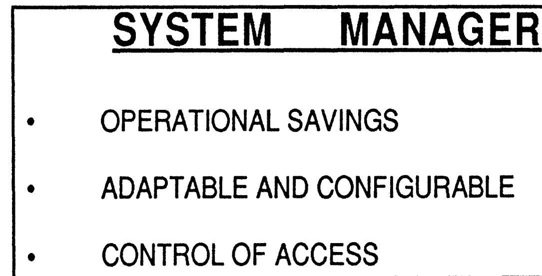



*Figure 10*

As Figure 10 shows, the system manager had different needs from the users and operators. All system managers are, or should be concerned with cutting costs, so the product had to produce demonstrable operational savings. At our site, the system manger wanted this tool to help him avoid unnecessary hardware purchase or labor intensive procedures.

We were not quite sure exactly what sort of operational parameters we would want to run under. This was our first VAX, so many details about actual usage were unknown to us. Avoiding assumptions helped us avoid making decisions about the archival software that could restrict us unnecessarily. So we decided we should build into it as many choices about the archival environment as we were ever likely to want.

And our own system manager, true to his kind, was concerned with security issues, at different levels. For a start, he didn't want people

getting hold of data which they should not be allowed to access; but also he didn't necessarily want to let everybody use the software. He anticipated that he might need to restrict usage to certain authorized people. So we had to build in that ability, too.

## IMPLEMENTING THE GOALS

After evaluating the goals, we came up with the main archiving program. As shown in Figure 11, the archiving program allowed a 2-way flow between that program and the user at his terminal.

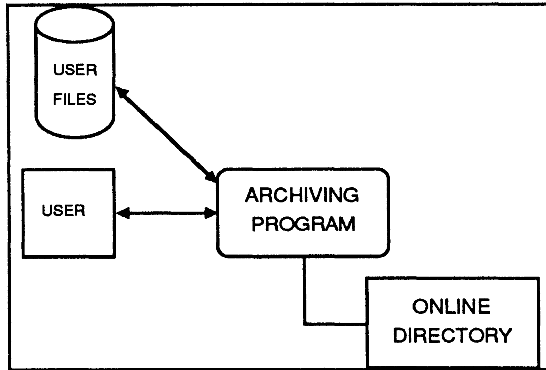



*Figure 11*

This flow allows the user to send requests to the program, which can then send back information. The program then accesses user files. For instance, if the user wants to archive a file, the program checks that the file exists and that the user has access to it. There's no point in delaying those checks.

The next major component in our diagram is the central online directory. The archive program reads and writes the master directory file so that it contains up-to-date information about all files which have been archived or are in the process of being archived. User archive directory commands cause the program to read the directory and pass information to the user.

What happens when files are to be transferred from disk to some offline medium, or vice-versa? We handled this situation by creating a separate VMS process to perform the data transfer, as shown in Figure 12.

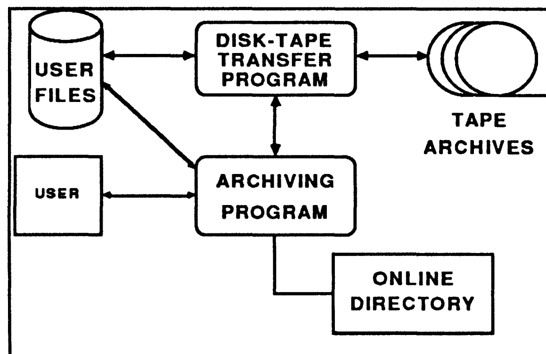



*Figure 12*

The archiving program hands the transfer program a list of commands to be carried out. One or more requests can be processed for files to be saved offline (transferred from disk to tape or some other medium), or restored to online media. The commands include all the details of the disk file specification, the archive file specification and the tape to be used. The online directory keeps the information about which tape holds which file, and this is used for any restore request.

Finally, we built a request queue system for our operators. We didn't want the operators having to run around the instant anybody entered a request for a file to be saved or restored. So requests usually enter a queue, as shown in Figure 13, which is simply a file managed by the archiving system rather than a VMS queue.

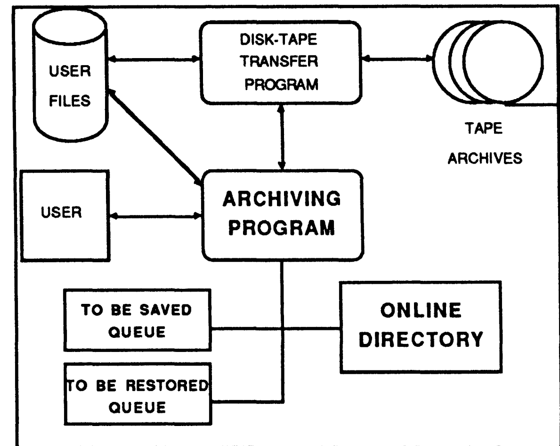



*Figure 13*

Separate queues are maintained for files to be saved and files to be restored. When the operator wishes to process the queues, the queue entries are processed one by one and the tapes read or written as required. We felt that the queue system was convenient for holding requests together from any number of users until they were to be processed at some agreed time.

## RESULTS ACHIEVED

The acid test was: did our design turn into the features that our various categories of users wanted, to make this whole thing work? I'll outline a few of the features that we implemented so you can judge for yourself whether we achieved those design goals.

Again, let's evaluate the results for the end user, as shown in Figure 14.

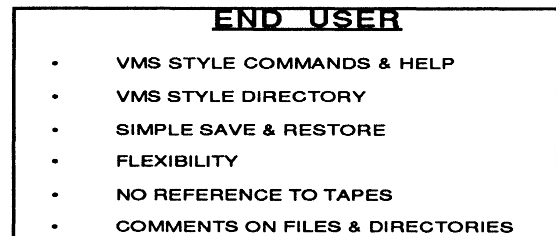



*Figure 14*

Any self-respecting designer or programmer on VMS is going to want to use the VMS parsing, command syntax and help features. It's one of the aims you tend to have-- to make your software look as if it came straight from DEC. The benefit then to the end user is that the software is easier to use.

Our central directory for the archiving software lets users duplicate standard VMS directory structures. This means that users can save and maintain whole directory tree structures onto archive storage. That also means that the online directory information is displayed in a form very similar to that produced by the standard VMS directory command.

You may recall that simplicity was a goal. We achieved it by allowing users to enter save or restore requests. They can use all sorts of fancy qualifiers as well, but a basic save or restore request applies sensible defaults in much the same way that VMS copy would.

We also felt that there was really no need for users ever to be aware of which files were on which tape. There was no need for any concern about the physical media. The operators could manage that. So the end users are not given the unnecessary information about tape names.

A common problem with long-term storage is that users forget the directory for a particular file, and can forget a whole directory structure.

The names of files and directories can give you a good idea, but we decided to allow comments that can be held in the online directory. Users can put comments on individual files or on the archive directories and subdirectories. For instance, some of our departments were concerned with software development. They needed to freeze a software release and archive it. A comment on the top level directory of the software tree structure was useful as a reminder of exactly which revision level had been archived.

To make life easy for the operators, we had to fit in with their style of working, which was mainly working to schedules, and so we provided the abilities shown in Figure 15.

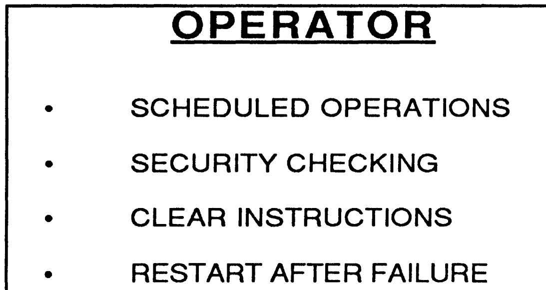



*Figure 15*

We didn't want to introduce unnecessary ad-hoc work for them to do. So our queue-based request system allows them to schedule archiving activity. We agreed that regular archive save and restore requests would be handled after 6 PM, making the normal service effectively an overnight one. (Other agreed times can also be added, if wished, such as midday.)

It's not that we don't trust our operators, but, like other sites, we cannot afford the risk of accidentally trashing a tape that is in use for long term archiving. So we made sure that our software makes careful checks on every tape that is mounted. The tape has to be recognized as a valid archive tape and have the correct label before it is read or written.

Whenever our operators perform the archival save or restore operations, the program knows exactly which tape to use. For saves it requests the current archival tape and appends to it, continuing where it left off the previous run. The next tape in sequence is requested when a tape is full. For restore operations, the software knows which tape each file is held on. In all cases, the operators just need to follow the instructions and, as we have seen, if they make a mistake, the system protects them from the unpardonable sin of data trashing.

Tape, of course, is not known as the world's most reliable medium. You can get tape read or write errors when archiving or restoring files; or the archiving job may be interrupted accidentally with a CONTROL Y key command, and so on. The software needed to be able to pick up where it left off. This means, for instance, putting unprocessed requests back into the queue to be dealt with next time. We certainly didn't want operators to manually re-insert requests.

How did all this fit in for our system manager?

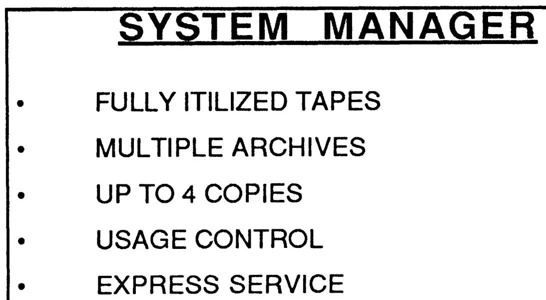



*Figure 16*

Figure 16 shows the abilities we put in the product for the system manager.

Using the full capacity of a tape allows us to cut our magnetic tape bills, and reduced the system manager's problems of managing tape storage. In a study of our tape usage on our timeshare services, we estimated that the average amount of tape in use on a user tape was less than 10%. That turns into a lot of tape wastage. Our archiving always appends to tapes until the end of tape marker so we use far fewer tapes than we would with user tape cycles.

Our business was also expanding. This meant that the requirements for archiving would be many and varied. We needed a flexible system that could cope. An "archive" is a whole group of tapes that, together, forms an offline archival set . By allowing multiple archives, we could accommodate different needs: for instance, we have set-up a general archive for long term file storage, and one that automatically sets the files to expire after 6 months. Hospitals in England have to keep archival records for a fixed number of years even if they are not likely to be required. A computerized archival system with automatic expiration of files can satisfy such a law.

A further refinement of archive configuration lets us specify that an archive should have from 1 to 4 tape copies. (We typically make 2 copies, but with growing importance of data, we saw the need to be flexible and create more copies. For example, sites may want 3 copies of every file archived: one in the computer room as the first choice for file restores, another in the basement in case the first cannot be read, and a third offsite in a secure vault.) The archiving system should let you define the number of copies when you first set up the archive, and then sort everything out from there.

Realizing that we may need multiple archives, one for general use by anybody, another for some special purpose restricted to a certain group of users, we allowed an archive to be configured with different authorizations. Users could simply start archiving files, or the system manager or other privileged user could explicitly authorize each user before he could begin to use the archive facility.

We had a slightly uneasy feeling about one aspect of our request queue system: agreeing with the users that we would process the save and restore requests at a certain time each day would probably be fine most of the time, but there is always the guy jumping up and down telling you that he absolutely MUST have his file restored by yesterday. So we allow an express service or a fast queue system. This means that whenever somebody puts a restore request in the fast queue, the operators repeatedly see messages telling them there is urgent work to be done. But you have to make sure your users don't abuse this privilege, or your operators may be overworked.

## DESIGN INTERNALS

We've gone over why you might select an archiving system to help take data offline, and both the goals and appearance of such a system. Now, I'd like to focus on a few of the practical design aspects of our archiving system, or the internals as identified in Figure 17.

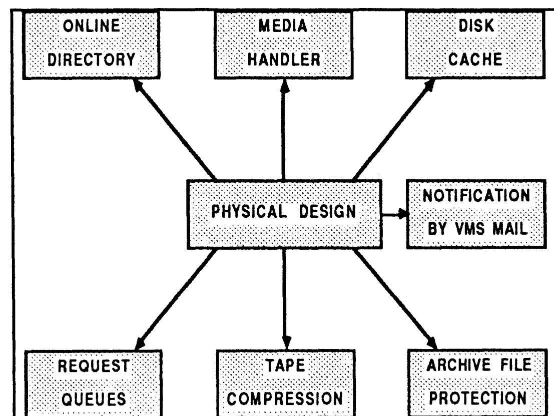



*Figure 17*

The topics shown above emphasize the ways in which the software should be designed to be self-maintaining, modular and efficient while serving the needs of the user, operator and system manager groups. As I go through each distinct area, I'll bring out some of the reasons why we designed the software the way we did and mention changes we may be making in the future.

Each separate archive needs its own online directory in a central file for all the users of that archive. If individual directories for every user resided in the user's VMS area, a user could damage his archive directory or even deleted it, destroying access to his archive files. A centralized directory also makes it easier to secure data from non-privileged users. For the directory structure, we used an indexed sequential file whose keys are based on the directory and file specification. This provides an entry for every file archived or put in the archive save queue, and keeps plenty of information about the file, such as name, size, dates, etc. (It's important to keep this data in a compact form or you defeat the object of saving disk space.)

The request queue system uses an RMS sequential file that contains the details of users' save or restore requests. This method suits both the users and the operators. Separate queues are used for saves, restores, fast saves, and fast restores. Each different archive has its own set of queues. When a restore queue is processed, all the requests are sorted into tape order. The online directory entries shown which tapes hold which files. By sorting restore entries, the system doesn't need to ask for the same tape twice.

We also provided a modular media handler-- which is the process that handles all the data transfer between disk and tape for save or restore operations. This is run as a separate subprocess for several reasons. First, there is no need for our users to have that data transfer code brought in when they run normal archiving operations. That code is only needed by the operators.

Second, by making the code modular, we can configure an archive to use different media handlers. Our normal handler is a process that transfers files between disk and tape. However, we also created a disk handler to transfer from disk to disk, with the archival disk acting as a pseudo-tape. (The main benefit of this was for debugging purposes in the development phase.) There was another interesting fact to take into account: a high proportion of file restores take place in the first few days after the file has been saved. This meant that it would be a good idea to allow the option of a disk cache. We allow our system manager to define an area on any mounted disk of any size that he chooses. He can also define the maximum size of any individual file that may be saved into this disk cache. Whenever a user enters an archive save request, the system checks whether a cache is enabled, whether there is enough room in it, and whether the file to be saved is at or below the maximum size allowed. If these parameters are met, the file is saved immediately and no queue entry is made. If not, a regular queue entry is made. Eventually, files in the cache migrate to the normal archival tapes, either because their life span in the cache has reached some pre-defined limit, such as a week, or because the cache is getting full.

A question that may have occurred to you is that after we write files to tape, what happens if users wish to delete these files? Well, a deletion of an archived file causes the directory entry to disappear. That means that there will be files on tape that are no longer active. Tape compression allows us to take archival tapes which are no longer very full, say less than 60%, and compress them. The active files are restored from the tape into the disk cache and then re-saved, using the current tapes. Then , of course, the original tape can be re-used as needed.

The file protection scheme we chose at first was fairly simple: all protection, including any ACL, was held with the data on tape so it could be faithfully restored; we also had access protection in the online directory entry for every file to indicate who could restore the file. This protection specified group and world user access. Owner and system were assumed to have access. Later, we decided that we could do better. The full system, owner, group and world protection fields are now entered directly into the directory. We also keep ACLs there, but it is possible to configure an archive so that the ACLs are only kept on tape, just in case we end up with excessive space taken up in the online directory.

Finally, we decided that we should have a notification procedure. If archival saves or restores were not happening immediately, via a disk cache, then our users needed some way of knowing when their request had been processed and what the result was. So we send notification of errors through VMS mail-- for example, a file that is to be saved may no longer exist on disk at the time of the disk-to-tape transfer. We also allow the user to specify if he wants a mail message even if the save or restore is successful. Mail seemed the obvious mechanism because VMS already does the work of alerting the user to new messages, and VMS Mail lets him read them.

So, in summary, as shown in Figure 18, we have ended up with a system that has evolved to a state where all our main file archiving needs are being met. Our tape library is greatly reduced in size as a result, and our operational procedures are simpler and more efficient.

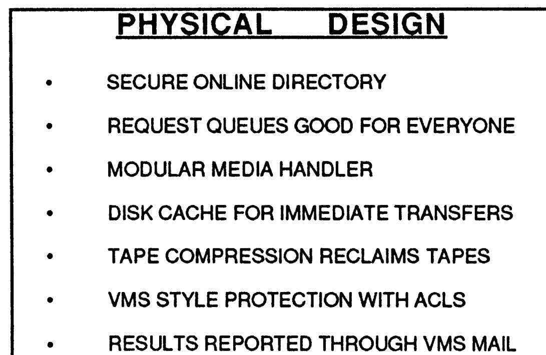



*Figure 18*

## THE FUTURE

As far as our archiving software goes, we do not intend to put the brakes on our development. We've developed an interface to All-In-1 for document archiving, and we already handle a variety of media such as standard half inch tape and TK50 and removable disks, as shown in Figure 19. However, we have also put together a prototype for optical disk - the write-once-read-many mass storage medium fits nicely into the concept of file archiving.

It doesn't matter whether the archiving medium is fast or slow, is mounted ready for use or needs to be manually placed on a drive, such as tape-- these are just variations on the basic theme. We will probably be looking into an interface to the 'juke box' packaging of optical disks as well. And as other storage technologies emerge we can develop interfaces for them too and plug them in, so who knows what the future may hold?

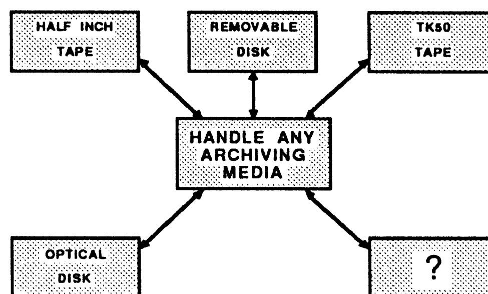



*Figure 19*

## SUMMARY

We have defined different solutions to coping with full disks, evaluating BACKUP procedures, tape management and archiving to see what each offers. Archiving systems, as was demonstrated, meet the needs of larger sites, or sites with heavy user activity. The design goals of a good archiving system were discussed. With the resources now available, there is no longer any need to allow the situation to develop where those bulging disk drives and huge volumes of tapes threaten to engulf system manager, operators and user with an unmanageable amount of data.

The author, Malcolm Dunn, can be contacted at:
UIS/Software Techniques, Inc. - 6600 Katella Avenue, Cypress, CA 90630
(714) 895-1633

Proceedings of the Digital Equipment Computer Users Society, Marlboro, MA 01752-1850