LBL-5315
UC-32
TID-4500-R64

BERKELEY WORKSHOP ON DISTRIBUTED DATA MANAGEMENT AND COMPUTER NETWORKS

Papers Presented at a Workshop Sponsored by

Lawrence Berkeley Laboratory
University of California

and

United States Energy Research and
Development Administration
Washington, D.C.

May, 1976

Dr. Donald M. Austin, Workshop Chairman

Mr. Dennis E. Hall, Program Chairman

Dr. David Richards, Program Advisor

Prepared for the U.S. Energy Research and
Development Administration under Contract W-7405-eng-48

DISCLAIMER

THE BERKELEY WORKSHOP ON DISTRIBUTED DATA MANAGEMENT AND COMPUTER NETWORKS

Donald M. Austin
Computer Science and Applied Mathematics Department
Lawrence Berkeley Laboratory
University of California

INTRODUCTION

The Berkeley Workshop on Distributed Data Management and Computer Networks was held May 25-26, 1976 at the Lawrence Berkeley Laboratory and the Marriott Inn in Berkeley. The workshop was organized by members of the Computer Science and Applied Mathematics Department of LBL under the sponsorship of the Energy Research and Development Administration (ERDA). Workshop participants included representatives from ERDA, the national research laboratories, universities, research institutes, and industry. There were approximately 150 registered participants, 44 of whom were speakers.

These proceedings contain short position or idea papers presented by the invited speakers. In some cases only abstracts are available because the presented papers are to be or have been published elsewhere. It is interesting to note that many of the abstracts and several complete papers were delivered via network mail. The last paper submitted for these proceedings was copied on a high-quality terminal at LBL the day after the "deadline"; I was even able to correct a few typographical errors before copying the paper. This practice is becoming quite common for some journals and newsletters, and is proving to be a viable medium for such tasks.

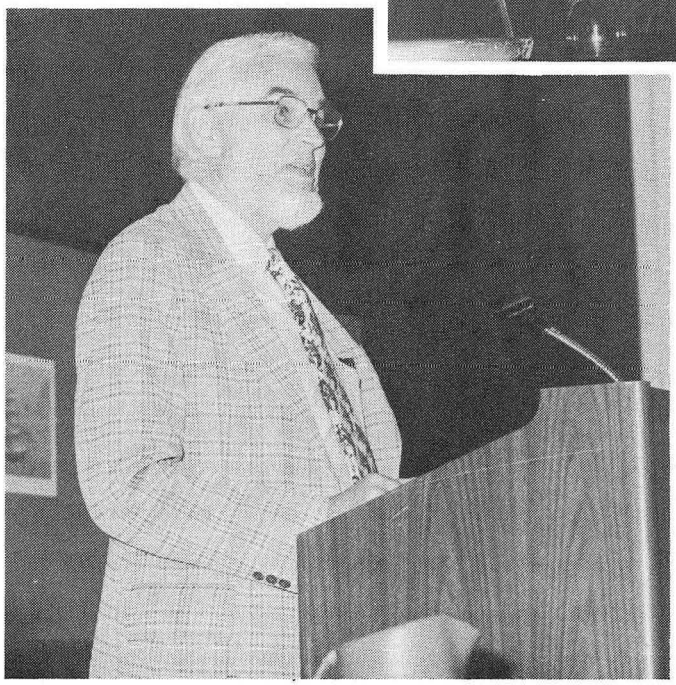
The interest and enthusiasm shown in these two areas of computer science appear to justify making the Berkeley Workshop an annual event. In response to many suggestions on the organization and format of the workshop, we expect to send out notification by late fall or early winter for the spring workshop. Next year's workshop will change somewhat to provide less parallelism in subject matter and more in level of presentation. Tutorial or survey sessions will be scheduled in parallel with more technical research results sessions. In addition the schedule will expand to three days, to allow more time for discussion, between-session conversation, demonstrations, and exploration of Bay Area phenomena.

I wish to offer sincere and enthusiastic thanks to all the speakers, many of whom prepared excellent presentations and papers on very short notice. I also wish to thank all those folks who did an excellent job in making arrangements, tracking down details, and generally keeping the workshop running smoothly:

The Coordinators: Jean Lynch, Lee Handeland, Suzanne Kranz
Agenda Preparation: Bruce Burkhart
Program Arrangement: Dennis Hall, Dave Richards
Wine Tasting: Carl Quong (and Inglenook Vineyards)
Terminal Arrangement: Harvard Holmes, Virginia Sventek
(Marvin Catchpole of Tektronix, Inc. provided several terminals)
Computer Center Tour: Bob Harvey, Bob Fink

Finally, for offering financial support and intellectual encouragement, I wish to thank Dr. Milton Rose of ERDA, Dr. Andrew Sessler, Director of LBL, Dr. Robert Birge, Associate Director of the Physics, Computer Science and Mathematics Division, and Carl Quong, Head of the Computer Science and Applied Math. Dept.


Marriott. INN
BERKELEY MARINA



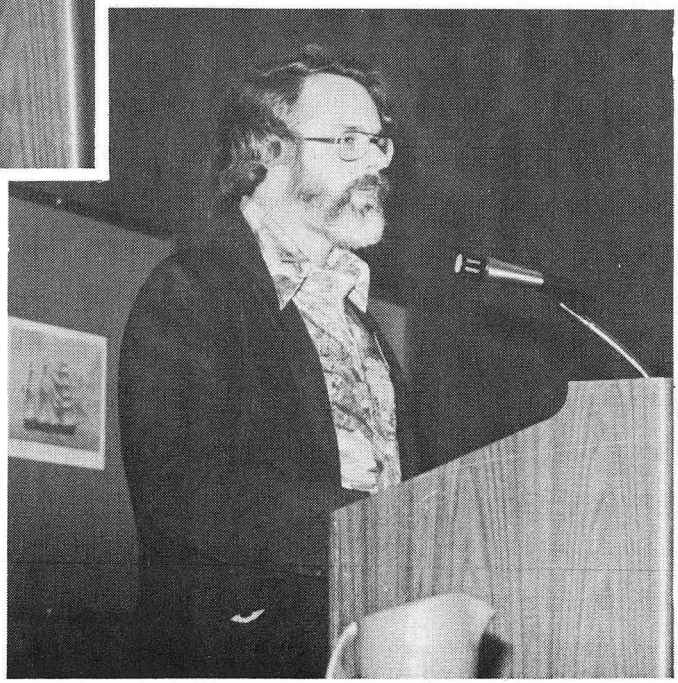
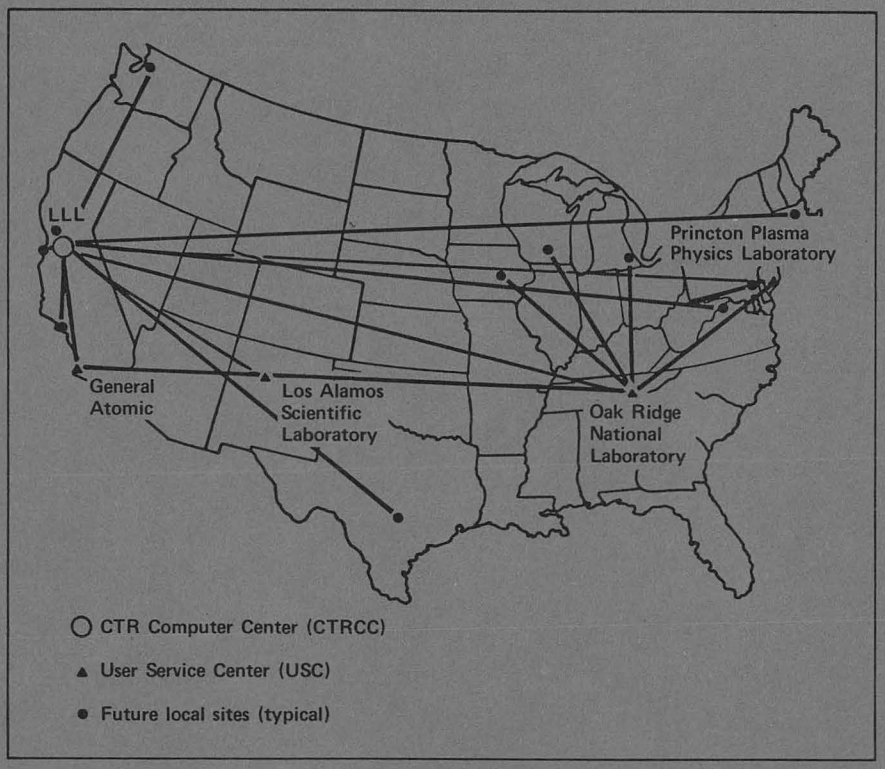


Table of Contents

I. AN OVERVIEW OF EXISTING DISTRIBUTED NETWORKS



Implementation and Future Plans for Existing Networks

EDUCOM

Gene F. Franklin
Associate Provost for Computing
Stanford University

In 1972-73, Educom sponsored a series of working seminars on national networking. These seminars are reported in the book, Networks for Research and Education, edited by Greenberger, Aronofsky, McKenney and Massy. In that report a facilitating network is proposed to stimulate the sharing of computer resources among universities. A direct outcome of that study was the establishment of the Planning Council of Educom with representation from 20 universities who have agreed to conduct an intensive study of computer resources at their respective campuses, and to investigate the best means of sharing those resources. The result has been the establishment of a preliminary network, referred to as the EDUNET, utilizing the communications facilities of Telenet and constituting the first phase in the fullfilment of the plans initiated in the original study.

In parallel with the prototype EDUNET, Educom is conducting a simulation and gaming study of a network of university and college computers to explore the important parameters in the decision-making processes respecting resource sharing. This talk will give a brief progress report of the several phases of the EDUNET, and present some personal projections of the next steps to be taken.

The CTR Computer Network*

John Killeen

Lawrence Livermore Laboratory, Livermore, California

ABSTRACT

The National CTR Computer Center has been established at Lawrence Livermore Laboratory to meet the major computing needs of the U.S. magnetic confinement fusion program.

The initial configuration of the Computer Center consists of a Control Data Corporation 7600 computer with 500 K words of large core memory and 64 K words of small core memory as well as disk storage. The computer network consists of User Service Centers (USC's) located at the CTR projects at Los Alamos, Oak Ridge, Princeton, General Atomic, and Livermore, which are connected to the Computer Center via leased 50 kilobit/sec lines. Each USC consists of a Digital Equipment Co. PDP 10 computer, disk packs, printers, and other peripheral equipment. The USC's serve as local computers for small jobs and experimental data processing, and as remote job entry terminals to the 7600 for large numerical calculations.

In the future, the CTR computer network will be expanded to include smaller User Service Centers at universities engaged in CTR research. In addition, the computing capability of the Center will be increased by acquiring the most advanced computers available in order to meet the needs of the fusion program.

*Work performed under the auspices of the U.S.E.R.D.A., contract No. W-7405-Eng-48.

THE CTR COMPUTER NETWORK

Orderly development of fusion power has required the evolution of a comprehensive developmental plan. These analyses have permitted detailed projections of the needs for large-scale computational support, without which the research and engineering studies cannot proceed. A 1973 *ad hoc* study for the Atomic Energy Commission categorized these needs by general type (seven categories) and estimated the time and equipment requirements for each, predicting a rapid expansion in demand that would quickly outstrip computer availability. This study urged the AEC to enlarge its CTR computer capabilities both locally - at individual CTR research sites - and through creation of a national CTR computer center.

In 1973, the AEC adopted this plan. LLL was selected as the site for the national CTR Computer Center (CTRCC). Requests for quotation went out in June 1974, and ERDA subsequently approved capital purchases including immediate acquisition of a CDC 7600 and supporting equipment for the national center, equipment for the individual CTR research sites, and the lease of high-speed transmission lines. Meanwhile, about October 1974, the national center went online to CTR users via voice-grade telephone lines, using a CDC 6600 computer and a remote job entry terminal (RJET) designed and built for Princeton. Last October, the CDC

7600 became operational, replacing the CDC 6600. Other equipment is being phased in as ready.

Figure 1 shows the present status of the data communications network that links the remote sites with the national center. The first priority is activation of the leased 50,000-bit/s transmission lines to four major CTR sites: Princeton, Oak Ridge, LASL, and General Atomic. Plans eventually call for extending service to other laboratories and universities across the nation that are involved in ERDA-supported CTR research projects. At present there are 20 such support centers anticipated, some to be connected directly to the national center and others through an intermediate site such as Oak Ridge (see Fig. 2). Plans also provide for high-speed interconnecting transmission links among the major research sites: from General Atomic to LASL, from LASL to Oak Ridge, and from Oak Ridge to Princeton. These will provide users at least two independent pathways for transmitting data and priority messages.

The concept is to provide different levels of local computer capability at the various remote locations according to research priorities and anticipated computational demand. At the national center, available to the entire community, is the high-speed CDC 7600 central computer with 64,000 words of small semiconductor memory, 500,000 words of large-core memory, and disk storage. Additional equipment at the

national center includes a PDP-11/50 central communications control processor, a PDP-11/50 network control station, and a CDC 6400 file management computer that stores files and maintains indices for information retrieval. Short-term storage is accomplished on rotating disk units with instantaneous access. Long-term file storage, now handled on conventional magnetic tape, will be switched over, in 1977, to a mass storage device whose capacity is about 500 billion bits.

At the next level are user service centers (USC's): PDP-10 computer systems with direct high-speed access to the national center through PDP-11/40 remote communications control processors. The general arrangement is shown in Fig. 3. Six USC's are now operational (Princeton, Oak Ridge, LASL, General Atomic, a USC for the CTRCC and a hard-wired USC at LLL for the mirror confinement program). PDP-10's were chosen for this application partly for their ease of upgrading by the addition of field-installable memory and disk units. (Their memories have already been upgraded to 192,000 words.) Local capabilities include text editing, compilations, and programs that do not require the speed and capacity of the CDC 7600.

Smaller problems are handled locally. Large-scale problems go to the national center and are returned to the USC for final processing, e.g. display and printout. The advantages of this arrangement include local availability, cost-effective

equipment utilization, and an interactive computing network among the various user service centers.

For the offsite users (those without USC's), the ultimate intent - depending on research priorities, anticipated demand, and the availability of capital funds - is to provide either mini-USC's or RJET's. The RJET comprises a card reader, printer, and the equipment needed for multiple-terminal access to the national center. The mini-USC provides all the above plus a 64,000-word memory. Meanwhile, we will maintain access to the national center, for these users, via the voice-grade telephone lines. Demand is already such as to warrant expanding the present 16-line dial-up capability to 32 lines. A second, similar dial-up capability might be added in the East, accessing the national center through a local USC.

Funds have been budgeted in FY 1977 to construct a computer building at LLL to house the national center's equipment and personnel. This building will accommodate permanent staff members and some 25 to 30 visiting research scientists from participating CTR laboratories. The first increment of the structure, containing some 3700 m^2, will cost approximately $5 million. In FY 1979, another 3700 m^2 of space will be provided, at a cost of about $2 million, to accommodate the expected growth in computing machinery and personnel.

Based on the foregoing, DCTR anticipates a steady increase in demand and capabilities of the CTR Computer Center involving

major expansions of equipment and function. The acquisition of new, larger central computers requires not only a substantial increase in ancillary equipment (such as file managers, storage devices, and network controllers) but also, as a prelude to specification writing, investigation into the compatibility of prospective next-generation machines with the existing system. This task is slated for FY 1977. Memory expansion at the national center and the user service centers is another priority project involving both hardware and software development.

As regards network development, the first task is to complete the remaining 50,000-bit/s transmission links. Eventually a space-satellite link may be established between LLL and the East, especially for very-high-bandwidth transmissions.

INITIAL CONFIGURATION OF THE CTR COMPUTER NETWORK

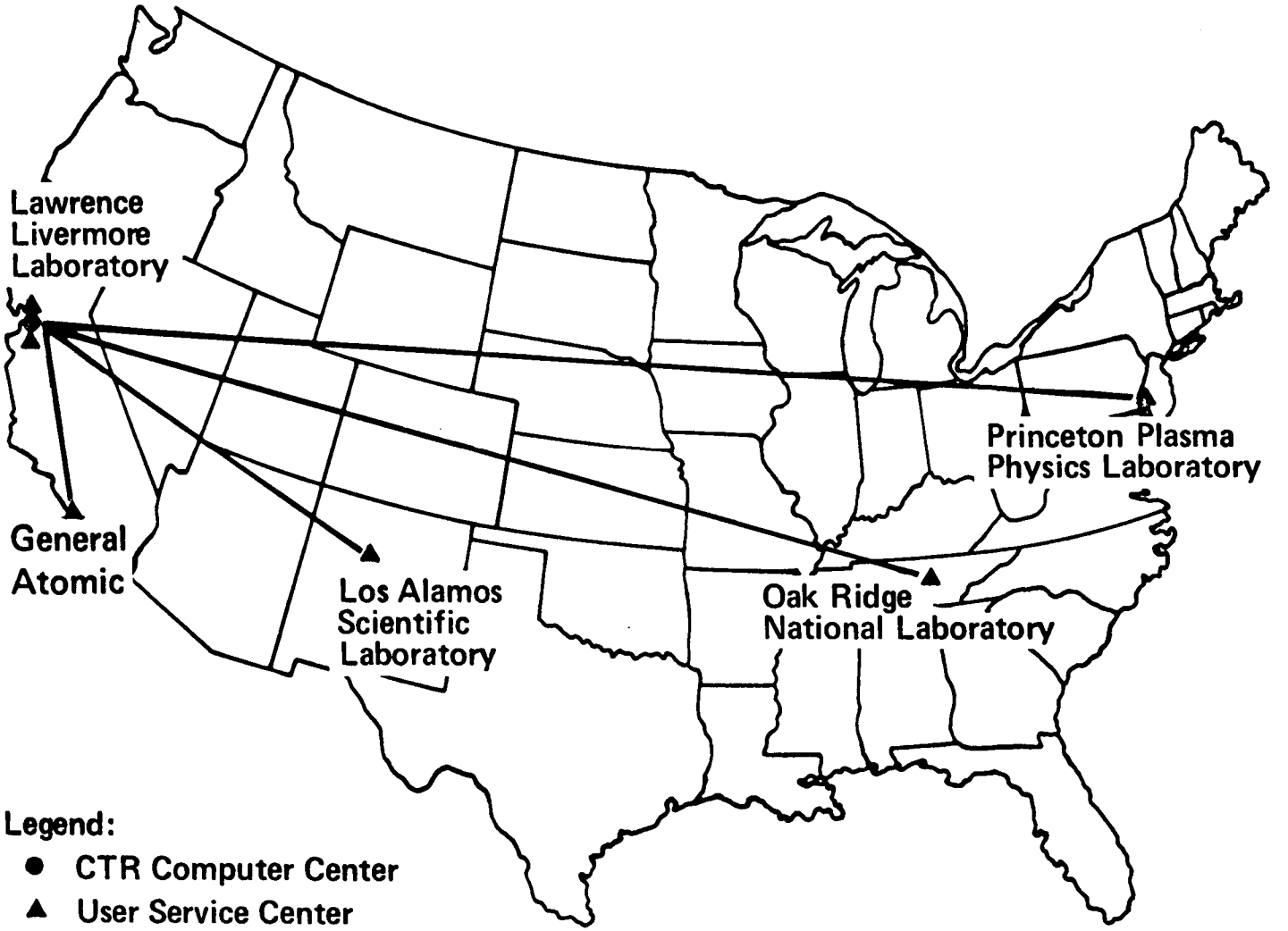


FIGURE 1

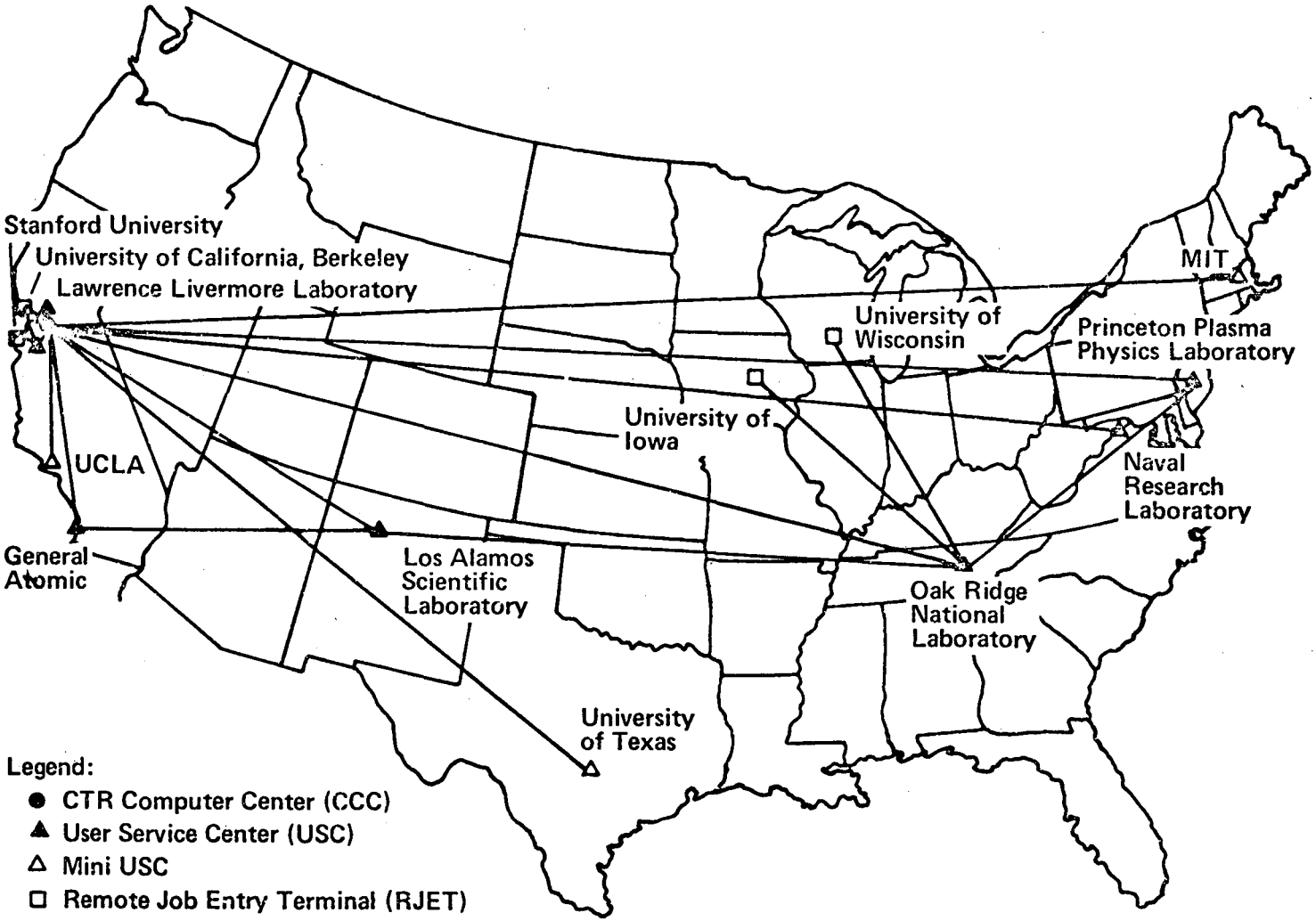
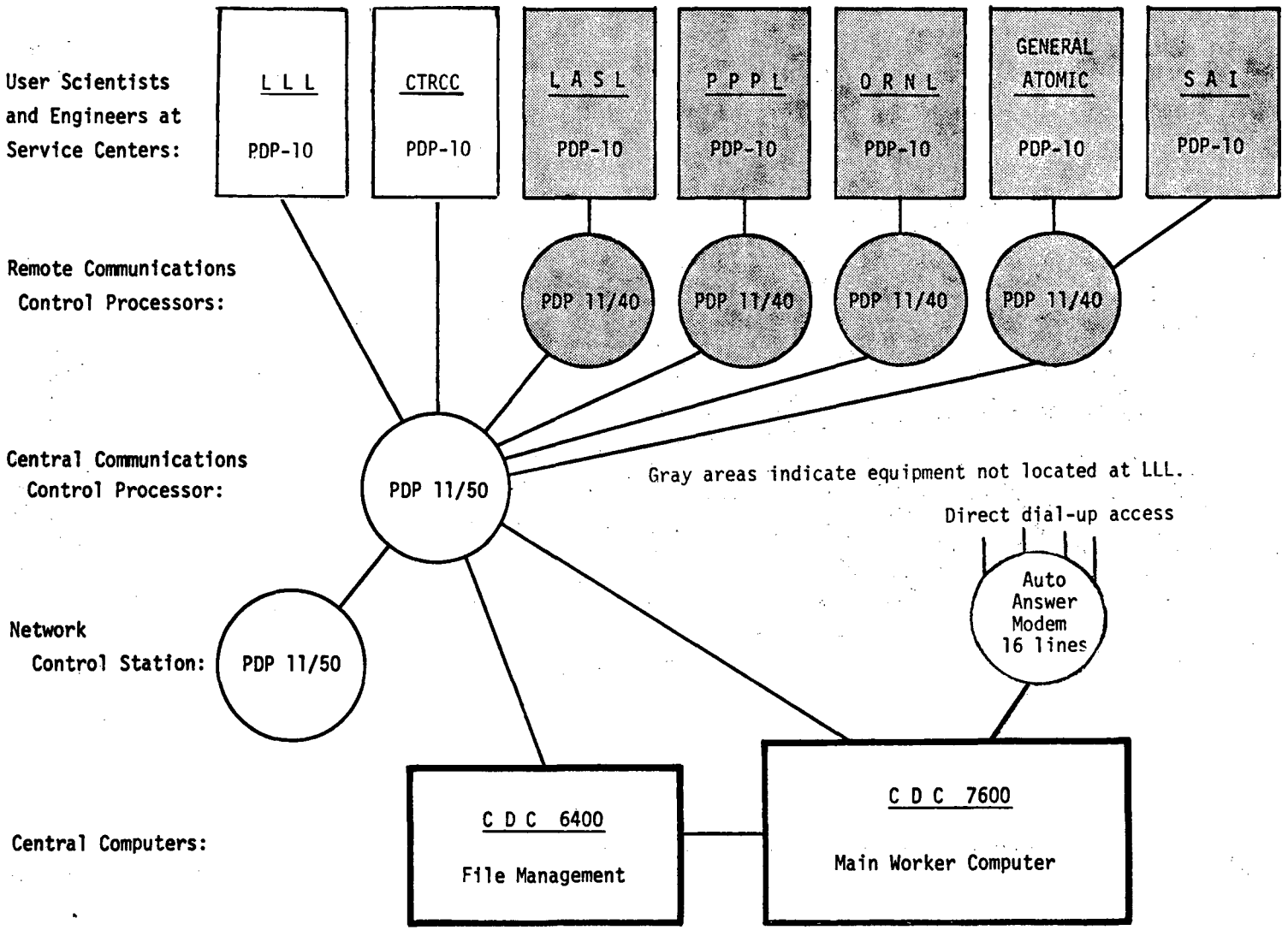

Stanford University
University of California, Berkeley
Lawrence Livermore Laboratory
UCLA
General Atomic
Los Alamos Scientific Laboratory
University of Texas
University of Iowa
University of Wisconsin
MIT
Princeton Plasma Physics Laboratory
Naval Research Laboratory
Oak Ridge National Laboratory
Legend:
CTR Computer Center (CCC)
User Service Center (USC)
Mini USC
Remote Job Entry Terminal (RJET)

FIGURE 2

LAWRENCE LIVERMORE LABORATORY

FIGURE 3

NATIONAL CTR COMPUTER CENTER



TYMNET

LaRoy Tymes
TYMSHARE, Inc.

Abstract

The evolution of Tymnet is described. The network was first designed to connect with the XDS 940 timesharing system. This emphasized intimate character by character interaction with low speed full duplex terminals. Nodes were small and primitive to reduce expense. Complex functions were centralized in a swapable user program in a timesharing environment.

The network grew rapidly in function and size while cost of mini-machines dropped, causing the initial design to be inappropriate. The growing pains and breakdown of the initial design are discussed. A new network, Tymnet II, is evolving to fit the new technology to the new requirements. Capacity, reliability, versatility, and response time will all be greatly improved while retaining compatability with Tymnet I.

The Future of the ARPA Network

Steve Walker

Advanced Research Projects Agency

The ARPA network is alive and well under the care and feeding of the Defense Communications Agency (DCA). The transfer of operational responsibility for the network took place on the first of July, 1975, with a transitionary period completed on the 31st of December, 1975. The network is now approximately 60 nodes in size, 26 of which are TIPs. It encompasses four cross country 50 kilobit links. There are nearly 100 computers on the network.

DCA has established policies for new applicants for ARPANET service. Essentially, the network is an operational Defense Department facility, and as such, DOD departments and agencies can be connected to the network by application to DCA. Contractors, working on defense related activities, may be sponsored by a DOD department or agency for access to the network in relation to their contract responsibilities. Non-Defense Department agencies may apply for access to the ARPANET to DCA and will be considered on a case by case basis. There are a number of non-Defense Department activities that were already using the network in research related activities prior to the transition of management to DCA. These organizations will be allowed to remain on the ARPA network under a grandfather clause. Non-government use of the ARPA network will be prohibited except as sponsored by Defense Department agencies or departments.

DCA is responsible for operation and maintenance of the backbone portions of the network including the network nodes. Subscriber costs

are assessed on a node basis, dividing the total cost of operating the entire network by the number of nodes and charging on a per month basis. Node equipment (IMPs and TIPs) are purchased by the subscriber and remain his property, though while attached to the network are under the operational control of DCA. Thus a new user has the option of purchasing an IMP or TIP or negotiating with the owner of a local IMP or TIP for access to his node on the network. There are eleven major sponsors of nodes on the network, ranging from ARPA, as the largest sponsor, through each of the three military services and on to the Energy Research and Development Administration, the National Aeronautics and Space Administration, the National Bureau of Standards and others. There is a sponsors' group chaired by DCA. It meets periodically to discuss the issues confronting the ARPA network.

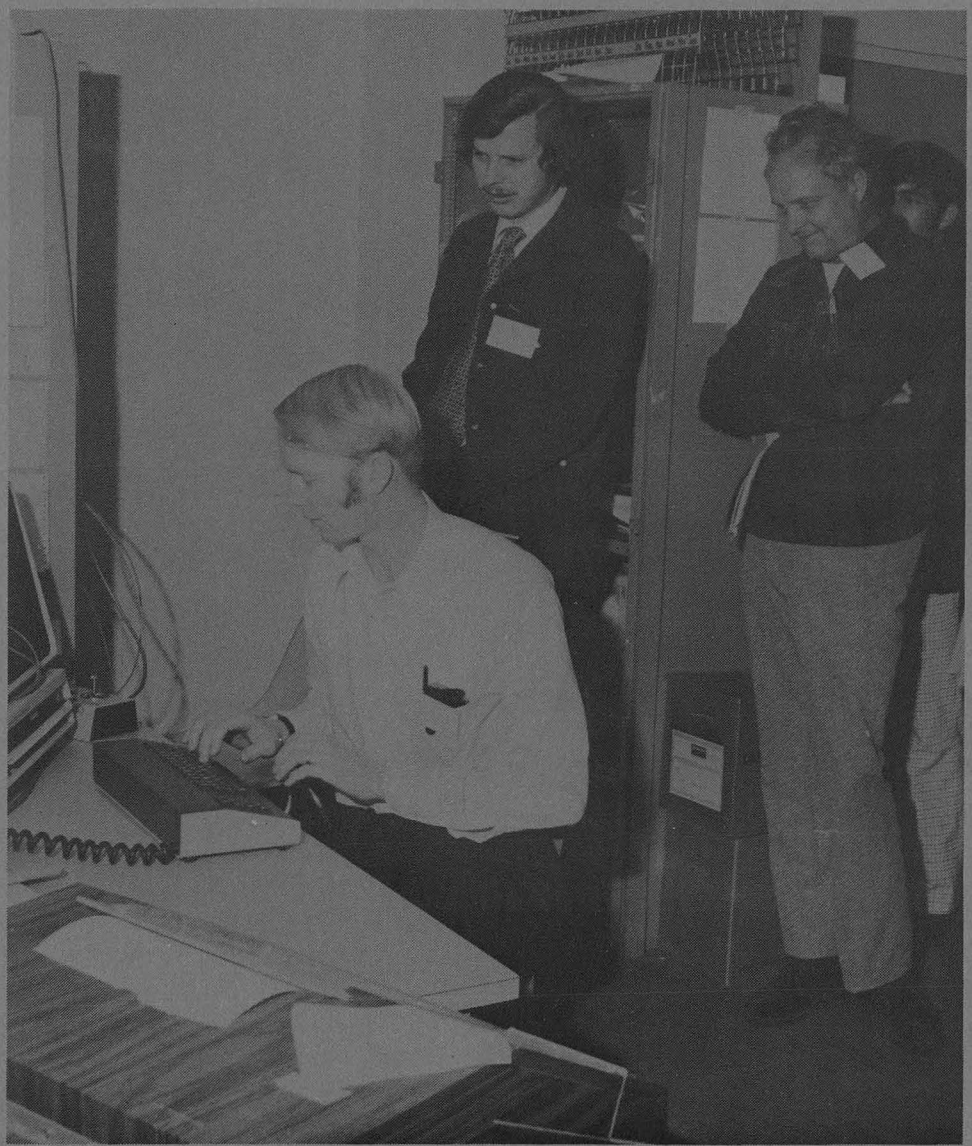
The Memorandum of Agreement transferring control of the ARPA network to DCA signed on the 5th of March, 1975, called for the continuation of the ARPA network for at least three years or until equivalent service could be provided. DCA is now in the procurement cycle for the development of a packet switched, common user defense network called AUTODIN II. As soon as AUTODIN II can provide service equivalent to the present ARPA network, the network will be phased out. Until such time it is expected to continue to operate essentially as it has over the past several years. In January of 1974, traffic on the ARPA network approached four and a half million packets per day. In January of 1975, it had grown to approximately six million packets per day. In April of 1976, the traffic exceeded eight million packets per day. In this same time period the

number of nodes has increased by approximately one-third, from 45 to 60. This reflects the maturing of the uses of the network and its growth as an operational facility.

I wish that I could convey these thoughts to you in person at the conference, but in my absence I will have to use the ARPA network message service. I want to leave you with the very strong impression that the ARPA network is alive, healthy, growing, and is as effective an existence proof in its operational role at DCA as it was during its experimental development role at ARPA.

(Ed. note: This paper was received over the network mail service, and in Steve's absence, was read to the Workshop by me.)

II-A. NETWORK GRAPHICS PROTOCOL



The Design of the ARPA Network Graphics Protocol

By

James C. Michener[1]

Abstract

The Network Graphics Group was founded on the desires to provide the graphics power of some network host computers to non-local users, and to employ the network environment in the same way that some sites employed their local terminal system -- for teleconferencing and shared, cooperative work surfaces. By successively lowering its goals to exclude first multiple users, then video data, then three dimensional data, and finally picture/subpicture structuring, the group was able to agree upon the set of graphics constructs which currently forms the Network Graphics Protocol. This protocol is sufficient to support moderately interactive graphics applications on a variety of graphics devices. Although the protocol can reduce (re-) transmission of graphics data, further savings could be realized through additional constructs.

The similarities between network graphics development and graphics standards development -- basically, the recognition and acceptance of common practices by a diversity of participants -- indicates the advisability of close cooperation between persons involved in Data Base standards development and network development.

---

[1]Author's address: Intermetrics, Inc., 701 Concord Avenue, Cambridge, Massachusetts 02138. The work reported here was performed while the author was at the Massachusetts Institute of Technology.

Introduction

This paper presents issues related to the development of the ARPA Network Graphics Protocol. The definition of the current protocol can be found in [1] and will not be discussed here in detail. The issues presented here appear to have relevance to the development of any distributed system in which interfaces must be defined between a variety of interchangeable subsystems, in particular, to a distributed data base system.

Goals

The ARPA Network Graphics Group was founded on the hope that sophisticated graphics systems on the ARPA Network could be made available to an audience wider than just the users of the host computers supporting them. It was intended that sites with display

equipment with a variety of capabilities would be able to make use of any properly written application program. Gaining access even to a simple application program like one for preparing plots of data would be beneficial. Also, the exploratory use of programs for computer aided instruction and graphical debugging was deemed desirable, for instance, prior to the local implementation of a similar system.

Another goal was that of achieving dynamic pictures on modest display equipment by utilizing the power of the sophisticated display processors on the network, like the Evans and Sutherland, Inc., LDS-1 at MIT. The idea was to transmit a picture structure to the powerful display device once, and then dynamically alter part of the structure (like a transformation matrix) over the network. After each change, the device could produce a digital copy of the transformed and clipped image which could then be shipped back to the user's display device.

Experiments

During the early stages of the protocol development, it was considered desirable to gain experience with the ARPA Network. To this end, various pairs of persons at different hosts volunteered to perform

experiments in their areas of interest in graphics. Among the experiments were the following:

1. A character recognizer at the System Development Corporation was interfaced to the LDS-1 at MIT. Multi-stroke tablet data (representing several printed characters) were sent to SDC, where the data were processed. The characters' codes, positions and sizes were returned to MIT for display.

2. An aircraft simulation was done using several computers at Harvard, connected over the Network. Values of "pilot" controls were read and processed and the visible display was updated about five times per second.

3. The OnLine System at UCSB was made accessible via the network and an interface to it was implemented at RAND.

Protocol Design

The protocol design proceeded by two methods -- discussions at meetings and publications of ARPA Network Requests For Comments. It was decided fairly early not to consider teleconferencing or video applications because of lack of relevant expertise among the participants. After some argument, three-dimensional graphics was also dropped.

Early protocol discussions were based on an approach of distinguishing strictly ordered "levels" of capability in a graphics device and requirements in an application program. It was intended that, during initialization of any graphics protocol connection, the level of requirements of the application would be compared with the level of capabilities of the display device. If it were determined that the device could not support the application, the initialization would fail. The lowest level was defined to include very simple graphics primitives -- similar to those available on a storage tube terminal. Higher levels were initially envisioned to include picture structure, transformations, etc.

The "level" approach was discarded when definition of the higher levels was attempted -- in particular, identification of what input mechanisms were associated with which levels of output. The level number was expanded into a detailed description of terminal capabilities and features which the terminal side of a network connection ("using end") must send to the application program side of the connection ("serving end") (as part of the protocol). It is up to the serving end of the connection to determine whether the device has sufficient features to support the application.

During the development of the level approach to the protocol it was hoped that a device/application level-mismatch could be corrected by a standard "level-matching" program which could simulate all the features of the higher level using only the features of the lower level. This program would pretend to the (real) application that it was a powerful display and pretend to the (real) display that it was modest application. When the level approach gave way to the descriptive approach, the concept of the level matching program was replaced by the concept of a library of subroutines in the application computer to be called to simulate features required by the application but unavailable at a display device. A benefit of the descriptive approach is that, based on a device description, a flexibly written application can modify its own behavior. A simple example is for the application to choose between color, blinking, or intensity (according to availability) for drawing attention to part of a picture.

It was decided that the using end should describe its capabilities instead of the serving end describing its requirements, because the using end is sometimes associated with very little computing power

and little can be done to correct a mismatch. An example of this situation is an intelligent terminal at a TIP (Terminal Interface Processor).

At first, a great deal of effort was spent on describing and attempting to choose among various ways of transmitting different sorts of graphical information. However, the problem of deciding on a protocol was clarified considerably when our task was seen to be the design of a general-purpose graphics programming system. The reason this represented a breakthrough is that most graphics people know what a graphics programming system is, but few know what a "protocol" is. The design thus became one of deciding 1) what features the system should provide, and 2) what lines of communication within the system should pass through the network.

In fairly short order, then, it was decided to base the system on a segmented (non-hierarchical) picture structure in which individual segments could be created, blanked, unblanked, appended to and deleted. Further, the fact that the proposed graphics programming system was general-purpose in nature operated to decrease the importance which people attached to internal formats employed to transmit graphical data, thus permitting agreement to be reached more readily.

It is interesting to note that during the development of the Network, it became clear that the TELNET protocol could support simple graphics applications for storage tubes, independent of the graphics protocol (which was then still being designed). Applications developed for locally situated storage tube display terminals (or a functionally equivalent refreshed terminal) work perfectly well on an equivalent terminal logged in via a TELNET connection. This achieved, in part, one of the goals of the graphics group, that of utilizing graphics applications remotely, without using a "graphics protocol" at all. Of course, this remote utilization is limited to terminals equivalent to those at the application's site. Use of the graphics protocol would lift this restriction and would also permit access to applications requiring incremental erasure.

Conclusions

The early goals of the Network Graphics Group have, in part, been met. Some have been met indirectly by the TELNET protocol. Others are met by the current graphics protocol which can support moderately interactive applications which do not have real-time

constraints. Other early goals still require development; one, tele-conferencing, is the subject of a session at this workshop.

The various experiments, and the early meetings of the Network Graphics Group, involving a large number of people, exposed the diversity of graphics problems and the multiplicity of possible alternative solutions. After the early attempt to classify graphics systems into levels, the group decided that a detailed description of terminal capabilities would provide the framework necessary for coordinating the two halves of a protocol. The design of the protocol took a firm direction when it came to be viewed in familiar terms, i.e., as a graphics programming system. The fact that the system was distributed over the network was seen to be secondary in importance to the design of the system as a whole.

It is hoped that this history has provided some insight into the problems of designing a network oriented system. The approaches taken by the Network Graphics Group seem applicable to the design of distributed data bases. In particular, the presence of the Network should not dominate the overall design of the

data base. Also, if a rigorous description of data base systems can be developed, it will aid in building systems that have to deal with more than one data base.

References

[1] Sproull, R., and Thomas, E. A Network Graphics Protocol. SIGGRAPH-ACM Computer Graphics 8,3 (Fall 1974), 27-51.

A SIMPLE IMPLEMENTATION STRATEGY FOR THE ARPA NETWORK GRAPHICS PROTOCOL

Harvard Holmes

Lawrence Berkeley Laboratory
University of California
Berkeley, California 94720

June 1976

The Network Graphics Protocol[1] (NGP) is a welcome addition to existing device-independent graphics standards. At the Lawrence Berkeley Laboratory, we are developing some simple software to enable the NGP to be used within the framework of our existing device-independent graphics system. Our software is reviewed here with the intent of encouraging more implementations of the NGP.

The existing system is shown in Figure 1. The most important characteristic of this system is that the device-independent interface is at a very low level -- almost the lowest possible level. The device independent interface does not incorporate any structuring facilities for the picture; it simply accepts commands to display lines and characters and converts them to the appropriate display codes for the particular devices. In practice, this system supplies the needs of a batch processing environment very well. Several packages of convenience routines or utilities have been interfaced to the standard calling sequences which comprise the device independent interface.

The device independent interface is designed to be used with Fortran and is written in Fortran. Only four functions are provided by this system: (1) clear the screen or advance to the next frame, (2) display a sequence of points or lines, (3) display a string of text, and (4) send or empty the internal buffers to complete a plot.

A minimum of parameters are used for each of these calls and a common block is used to communicate less frequently changed parameters.

Since the primary hardcopy device (a microfilm recorder) and the primary interactive terminal were both square, we provided a device coordinate system of 0.1 inclusive in X and Y. Non-square devices, e.g., the Tektronix storage terminals, provide a square area as the prime plotting area, with the user allowed to plot outside the prime area at his own risk. Users, of course, want to put the same picture on different shaped plotting areas without wasting any of the screen. In general, this is not possible.

An existing subsystem, Figure 2, writes an intermediate file containing the parameters for each call to the plotting routines. Another program reads this file and reconstructs the calls, which are then passed to the standard device drivers for interpretation. Suitable commands are provided for interactive operation of this program.

The NGP system is comprised of two parts (see Figure 3): a subroutine package to generate the NGP in the server host, and a program in the user host to interpret the NGP. The server host facility is easy to provide since every function provided by the standard calling sequences has an obvious counter-part in the NGP. A small amount of additional logic is required to automatically open and close segments as desired to make the operation completely transparent.

The user host system is more difficult since it must provide structuring facilities which are not present in the device independent system.

A data structure is therefore interposed between the interpretor and these standard calls, as illustrated in Figure 4. Since this structure is not being used by a hardware display processor but is only interpreted by software, double buffering is not required. Also, since memory space is at a premium, a simple memory compaction scheme is used for garbage collection. New segments are always added at the end of the list and segments are deleted by moving the following segments up.

The flags are used when an "end batch of updates" is received. They allow easy processing of incremental additions to an existing picture. That is, when only additions have been made to a picture, the flags are used to avoid erasing and redrawing the portions already there.

Although the system is not yet finished, only one major difficulty is anticipated. The difficulty is that at present the operating system does not swap processes with an active ARPANET connection. This means that large application programs would clog up main memory.

As an interim solution, a separate process will be used for the ARPANET communctions which communicates with the application program via shared files. Another alternative being considered is to multiplex the data over the TELNET connection.

In conclusion, our experiences have convinced us that (1) the design of the NGP is well done, (2) the details of the protocol such as the specific encodings used, really don't matter that much, and (3) a minimum NGP is really quite easy to implement.

REFERENCE

1. Sproull, R., and Thomas, E., "A Network Graphics Protocol," SIGGRAPH - ACM Computer Graphics 8,3 (Fall 1974), p 27-51.

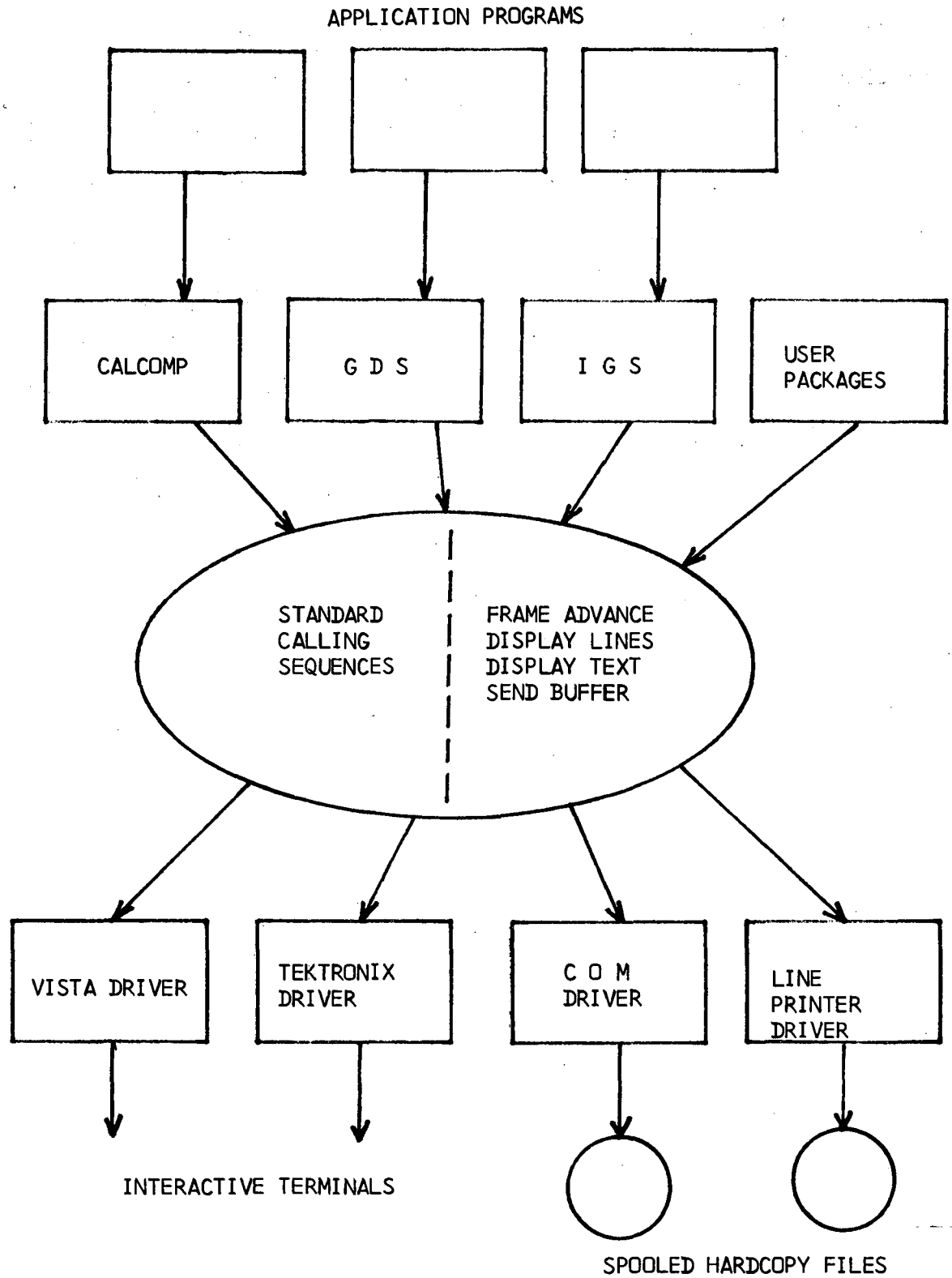


FIGURE 1. EXISTING SYSTEM

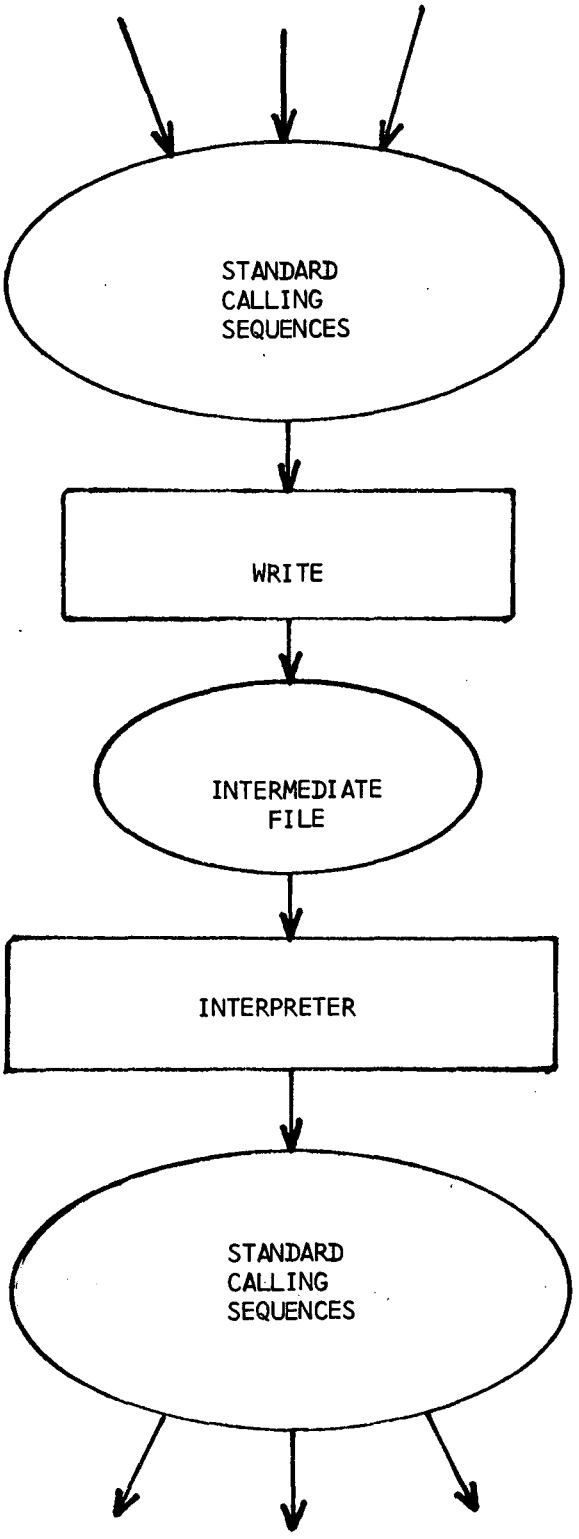


FIGURE 2. THE "REVIEW" SUBSYSTEM

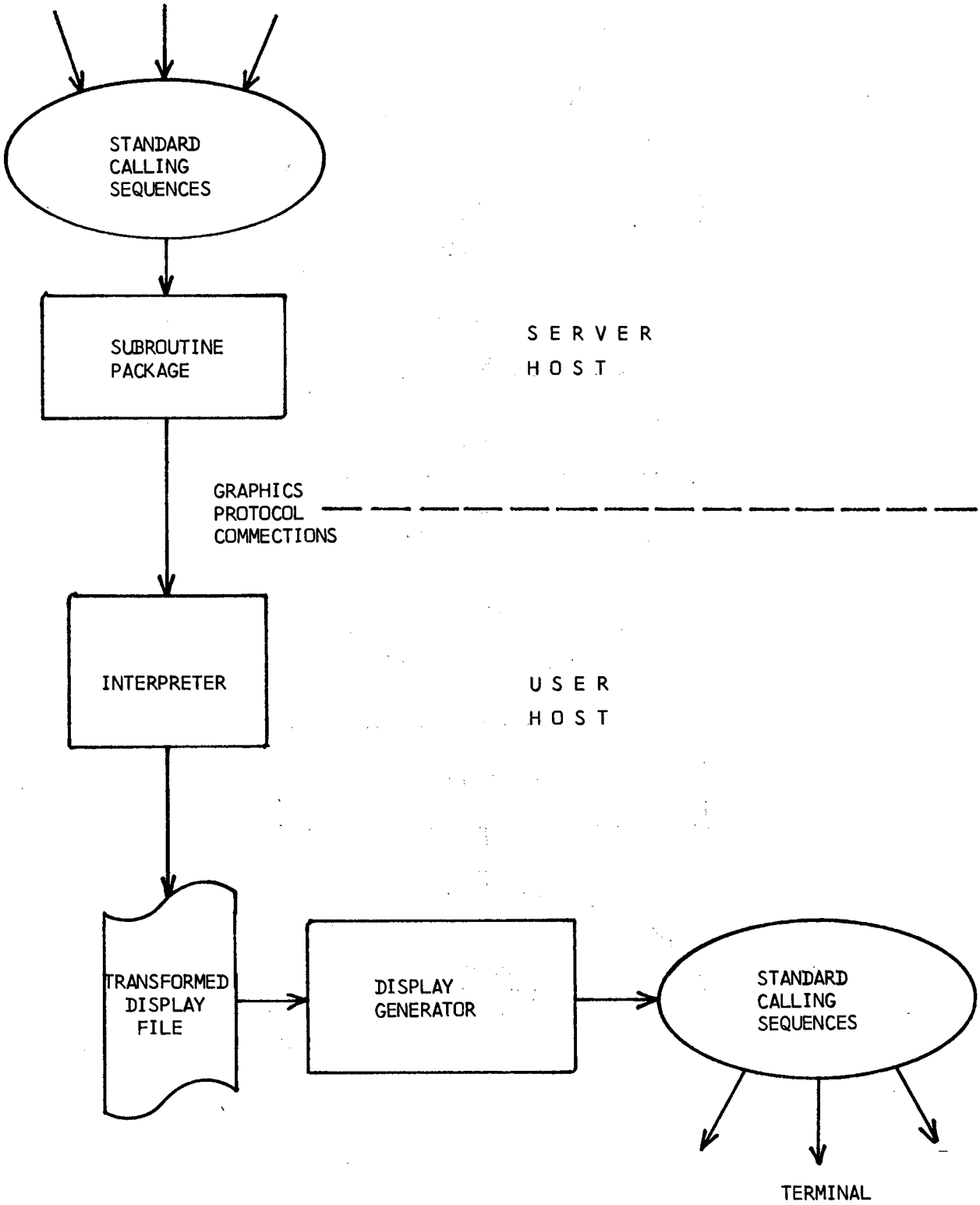


FIGURE 3. THE N G P SUBSYSTEMS

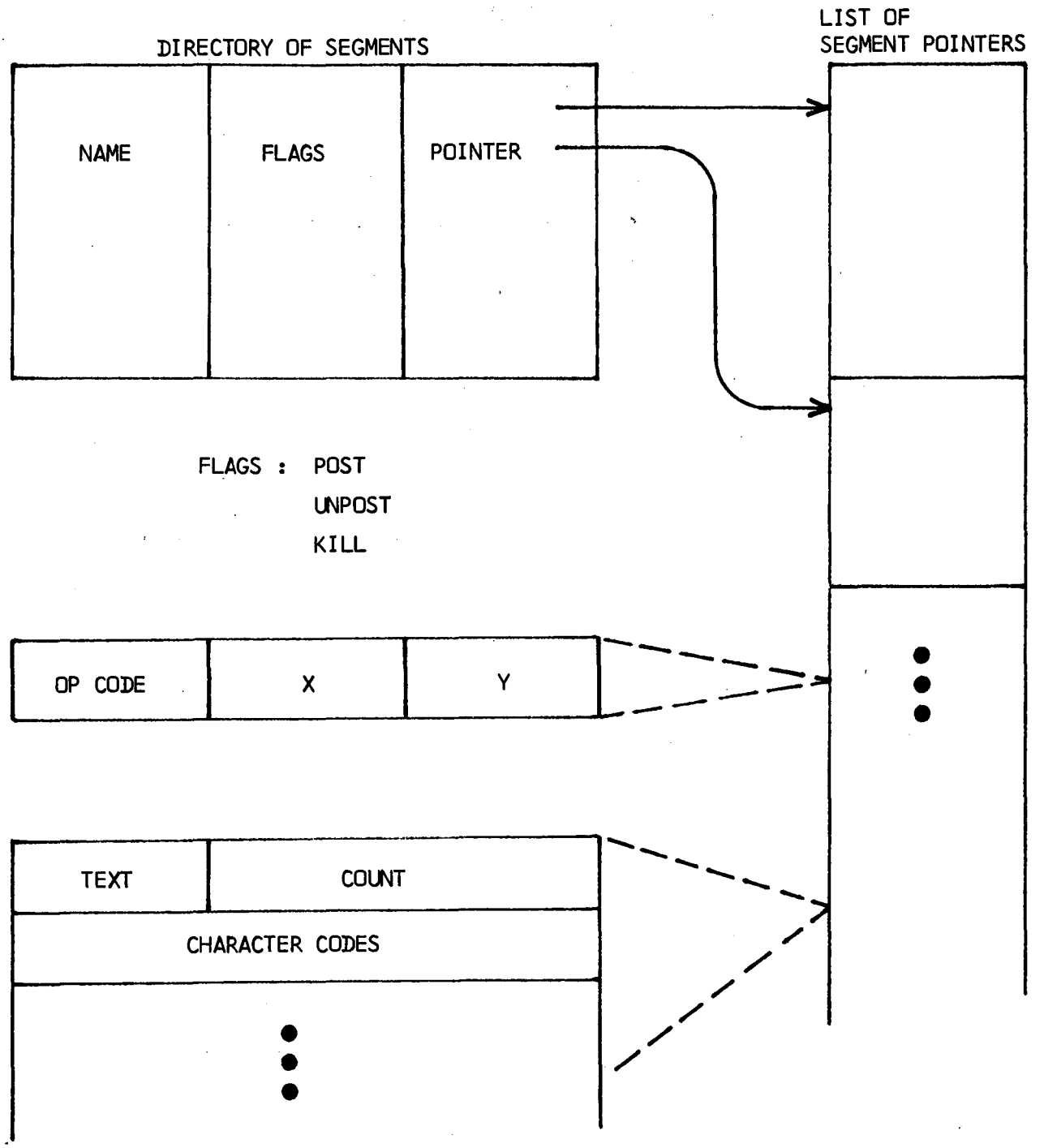


FIGURE 4. TRANSFORMED DISPLAY FILE DATA STRUCTURE

Network Graphics Isn't Networking

Robert F. Sproull
Xerox Palo Alto Research Center

The design philosophy behind the ARPA Network Graphics Protocol (NGP) (1) is that a general purpose graphics system (or "graphics package") should serve as the model for network graphics. This is a good philosophy. However, problems in the presentation of the design faults in some design details, and lack of applications in the ARPA community have impeded implementation progress. The major problems are:

- The NGP design document attempts to explain a philosophy of graphics system design and to describe a particular NGP design built around that philosophy. However, the document does not communicate the philosophy adequately, and gives many readers the impression of a mass of complex detail in the protocol itself.
- The greatest misunderstanding concerns the extent to which portions of the protocol are optional. There are about a dozen mandatory protocol commands that will produce a very good implementation. If the description of the protocol had been confined to this set alone (perhaps with the others in an appendix), the design might have been better understood.
- Some of the members of the NGP committee had too much experience with high performance displays and not enough with device independent graphics systems. Many of the optional commands arose from a counter-productive attempt to handle too many frills of current display devices. Pressure to include these features came from sophisticated programmers of sophisticated

equipment who thought an NGP could achieve high performance on high performance displays.

- As the NGP design was undertaken, and certainly after it was completed, there was no pressing application for network graphics on the ARPA network. The effect was disastrous: neither the NGP committee nor any other group knowledgeable in graphics system design worked out an implementation and thereby showed others how simple an implementation could be. Moreover, users of such a system would have emphasized design simplicity rather than raw, complex power or ability to cope with all sophisticated displays.

The blueprint for a successful NGP design is, in my view, very simple:

1. Designing an NGP has almost nothing to do with networking. However, it is essential to avoid viewing the NGP as driving a "terminal", and thereby confusing it with terminal protocols and encumbering it with the conventions that most operating systems apply to such terminals. Unfortunately, many aspects of graphics hardware give the terminal illusion (interactive use, portable devices that can be connected over telephone lines, etc.), which is hard to suppress. Instead, an NGP should separate graphics information from terminal information, and use a separate protocol and a separate "connection" over the network.
2. A design for an NGP is best derived by observing the cooperation of the two important parts of a (non-network) device independent graphics system: the "device independent" part and the "device dependent" part. The communication between these two parts can

be made simple and can be the prototype for introducing networking: the network provides the communication between the device independent part, which resides on the "host" computer, and the device dependent part, which resides on the computer that drives the graphics hardware directly. To gain the insights offered by such a software interface, the designer of an NGP should first have designed, built, and extensively used a good device independent graphics system capable of driving a reasonably wide variety of devices. (There are few such "good" systems. Almost no designs are adequately parsimonious in the functions of the package. See (2) for a reasonably good design, see (3) for how much can be accomplished with a few functions.)

The ARPA NGP design observes these dicta, but their power and worth were largely vitiated by poor exposition in the NGP document.

References

(1) R. F. Sproull and E. L. Thomas, "A Network Graphics Protocol," Computer Graphics, Fall, 1974, Vol. 8, No. 3, p. 27.

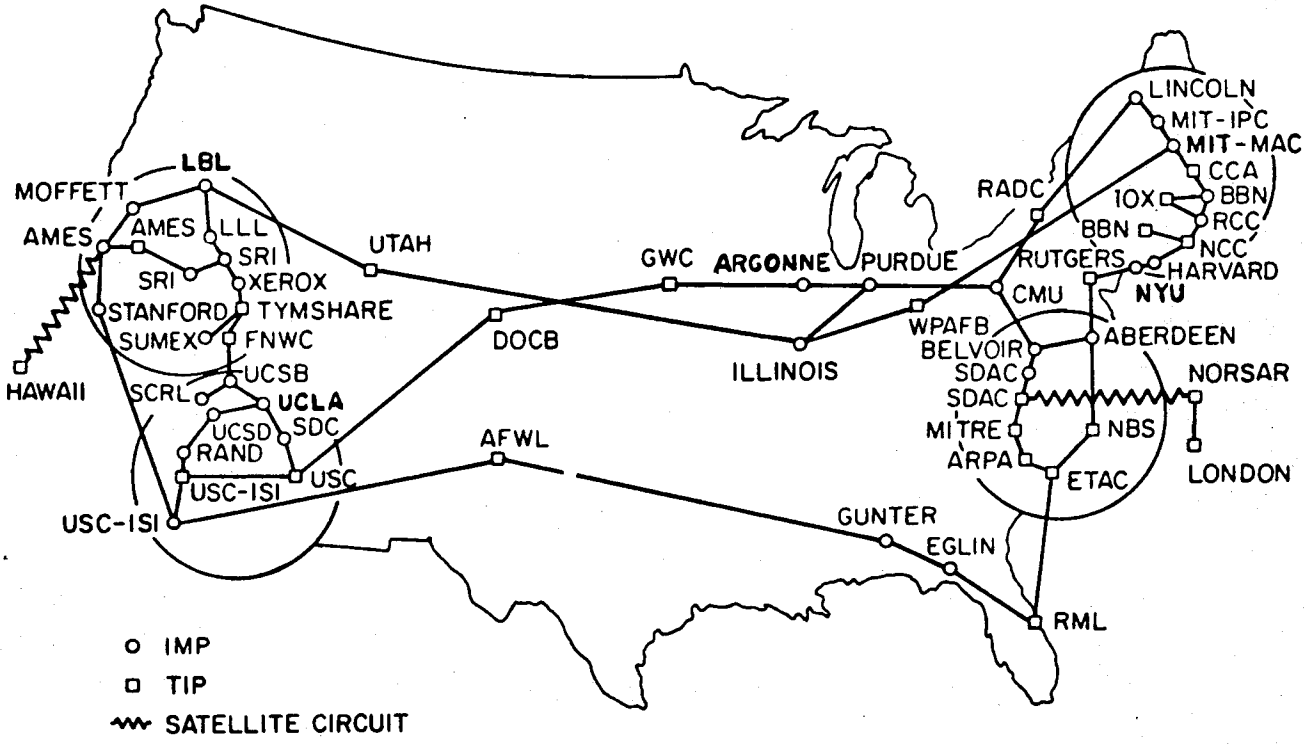
(2) A. C. Goodrich, "Integrated Graphics System User's Guide," Computer Center Memo 299, Computing Center, University of Michigan, Ann Arbor, Michigan 48105.

(3) W. M. Newman and R. F. Sproull, "An Approach to Graphics System Design," IEEE Proceedings, April 1974, Vol. 62, No. 4, p. 471.

II-B. MINI-HOST FRONT ENDS TO THE ARPANET

ARPA NETWORK, GEOGRAPHIC MAP

JUNE 1975



FRONT-ENDING AT ARGONNE NATIONAL LABORATORY

by

L. W. Amiot*

Argonne National Laboratory
Argonne, Illinois

INTRODUCTION

The cost, complexity, and productivity of large computing centers have changed drastically during the past decade. Large scientific computer complexes, such as the Central Computing Facility (CCF) at Argonne National Laboratory (ANL), provide unique resources which are not generally available either to the private sector or to smaller governmental agencies and contractors. The ability to share such resources on a laboratory, regional, national, or international basis has thus become a factor in the design and operation of these systems.

Historically, we have seen the total investment in hardware and software for large central computing systems increase while the average cost per computation has decreased. The utilization of these computing resources is thus of a critical nature in order to maintain an efficiency in their use. This provides a motivation for off-loading the network control from the central host system to a communication front-end system, and serves to relieve the load on the main processor and increase the amount of memory and CPU cycles available for processing of user jobs. This is the approach taken at the Central Computing Facility by the Applied Mathematics Division of Argonne National Laboratory and is the subject of this paper.

HISTORICAL BACKGROUND

The main processors in the present configuration of the ANL/CCF consist of an IBM S/370/195 which handles the batch computing and an IBM S/360/75 which provides a TSO, time sharing capability. The facility provides for the Laboratory wide remote batch capabilities through an ANL designed network (RADS) of remote card

---

*Work performed under the auspices of the U.S. Energy Research and Development Administration.

reader-line printer stations. As the RADS system reached its design limit of fifteen stations and the need for communication with off site, industry compatible remote batch stations developed, the support for standard, IBM HASP workstations was added to the system.

Together the RADS system and IBM HASP workstations have provided the remote batch processing capability for the system. Previous to the design and installation of the communications front-end system, support for the HASP workstations was handled by IBM 2701 and 3705 communication controllers. The low speed terminal oriented communication support for the time sharing system was and still is handled by standard IBM controllers.

FRONT-END SYSTEM

Hardware Configuration

The initial goal of the front-end system was to provide the communications support for the HASP workstations which are connected to the CCF. The front-end is a multiprocessor system of three interconnected Varian Data Machine, V73 computers (see Fig. 1). The CPU's are interconnected on computer-to-computer DMA channels and share a common memory. The memories are dual ported, 660 n.s. cycle time modules and thus support a high level of performance. The system peripherals include a moving head disk drive on each CPU and I/O devices necessary for program development and local file I/O. The system supports programmed I/O, cycle steal or DMA operations through the CPU, and direct memory access through priority memory access channels.

Communication lines are controlled by the CPU's through communication multiplexors (DCM's) and associated line adaptors. The system supports asynchronous, synchronous, and binary synchronous communications. Each MUX supports sixteen line adaptors which are character buffered and once initialized, transfer data to memory independent of the CPU on DMA channels. Support of the binary synchronous lines are under hardware control with control characters being recognized and interpreted by the adaptors as well as the checking and generating of cyclic redundancy characters. Each sixteen line MUX is capable of transferring data at an aggregate data rate of 60K bytes per second.

Communication between the front-end and the S/370/195 is maintained by a Varian channel interface unit (IFCU) attached to a selector subchannel of an IBM byte multiplexor. The IFCU logically appears to the IBM system as an IBM 2803 tape control unit with sixteen tape drives.

Data can then be transferred between the front-end and IBM systems in the burst mode from any of the sixteen logical units. IBM channel control is simulated by the IFCU hardware and handles such functions as initial channel selection and command decoding.

Operating System Software

The software in the front-end runs under the Varian VORTEX disk based, real-time operating system. VORTEX is a modular operating system which provides for the controlling, scheduling, and monitoring of tasks in a real-time multiprogramming environment. The system supports both user written foreground tasks and background processing such as assembly and Fortran compilation.

Intercomputer communication is handled by drivers which support the computer-to-computer interface. Control of the communication lines is handled through the Varian VORTEX Telecommunications Access Method (VTAM). VTAM is an integral part of VORTEX and provides a macro level of programming to the applications program with lines and terminals being configured through a Network Definition Language.

SUPPORTED NETWORK FACILITIES

HASP Workstation Support

The model 195 runs the IBM Attached Support Processor system (ASP) under OS/MVT. The system supports standard remote batch HASP workstations via binary synchronous communications lines using an ASP dynamic support program (DSP) called RJP (see Fig. 2). This DSP uses an access method which allows control of communications lines through IBM 270X and 370X units. In order to allow communication with the front-end in a burst transfer mode, RJP was modified to allow read and write macros to selected RJP lines to be trapped and rerouted to the front-end.

The task of emulating the IBM 370X in the front-end was written as a foregound task (EMU) running under VORTEX. The EMU program:

- Opens and closes communication lines under command from RJP.
- Performs a store and forward operation for data between the communications lines and the model 195.

- Performs a cyclic redundancy check on data received on the lines and causes retransmission if necessary.

- Performs statistics and error recording functions.

- Configures physical and logical line relationships.

The protocol used between RJP and the HASP workstations is the HASP multileaving protocol. The protocol supports the following data streams for each workstation:

- console input stream
- console output stream
- seven reader streams
- seven printer streams
- seven punch streams
- seven file streams

All of these data streams are supported by RJP except the file streams; thus the front-end in addition to providing the communications support for standard HASP workstations also provides the capability to other front-end tasks to transfer data to the model 195 by having that task emulate a HASP workstation.

ARPANET Support

A number of ERDA laboratories and ERDA contractors (ANL, BNL, LBL, MIT, NYU, and UCLA) have formulated a research activity that has as its common goal a study of the usefulness of high capacity computing networks in resource sharing. High capacity computing networks can provide the necessary link to remote computing institutions and their corresponding diversity of resources. Indeed, networks have been extremely successful, for example, in providing time-sharing resources. Nevertheless, their effectiveness in transferring large amounts of information typical of scientific and engineering computations has not been adequately demonstrated. It is towards this goal that a cooperative set of experiments is planned over a connection to the ARPANET.

The ARPANET is an international network which interconnects host computers of diverse types. The network was developed under control of the Advanced Research Projects Agency (ARPA) and funded by the Department of Defense. The network is a packet switching network with the communications subnetwork consisting of a series of nodes interconnected by 50K baud links. Each node is implemented with an Interface Message Processor (IMP) computer. It is the function of the IMP to interface the host computers to the communications

subnetwork and perform the store-and-forward function of that subnetwork.

In deciding to provide a connection between the CCF and the ARPANET, a strategy was developed to decide what remote resources were required from the ARPANET, what local ANL facilities should be made available to the ARPANET, and how to best implement the strategy. The following were determined to be the required functions the resultant system should provide:

- terminal access to remote hosts on ARPANET for local ANL users,
- a remote batch entry capability to the ANL model 195 for remote ARPANET users,
- a file transfer capability between the ARPANET and the ANL model 195 so that data could be stored or retrieved as OS datasets,
- terminal access to the ANL TSO system for remote ARPANET users, and
- a general means to interconnect the ARPANET with other ANL networks as the need arises.

Implementation of the connection required a hardware interface to the IMP and a set of processes for the following network standard protocols:

- User File Transfer Protocol (FTP)
- User Telnet
- Server Telnet
- Network Control Program (NCP)

It was decided to implement the processes in the front-end system and provide a special hardware interface between the front-end and the IMP rather than to implement the protocols directly in the model 195. Indeed, this was found not only more beneficial than the alternative of adding the processing load to ASP but the front-end also provides a more convenient interface to other networks to which connection was planned.

Other Network Connections

The front-end provides a convenient facility for interfacing other networks to the CCF and to the ARPANET. The ANL Intra-Laboratory Network is under design and has as its goal the development of an ANL wide packet

transmission network. The network will allow information to be transferred between nodes on the network and the CCF. Nodes will be implemented using microprocessor technology to provide line handling, communication capabilities and to provide a convenient interface to user systems. A connection between the Intra-Laboratory network and both the CCF and ARPANET was determined to be desirable.

One of the thrusts of our research program is ANL-MATHLAB which addresses the use of minicomputers for algorithm and software development with emphasis on numerical mathematics. MATHNET is a project which explores alternative approaches to network organization and to virtual memory systems as it relates to the MATHLAB project. A design goal of the system is to support not only terminal access but also provide the capability to transfer data files to remote sites. The ANL front-end system has the potential for allowing remote ARPANET terminals to access MATHLAB and to similarly allow files to be stored and retrieved to ARPANET from MATHLAB.

The ANL/CCF also supports a TSO system on a S/360/75. Terminal access to the system is being supported on slow speed asynchronous lines through standard IBM communications controllers. Terminal access from remote sites is via long distance, terrestrial phone lines. Because of the increased interests in resource sharing at the Laboratory, the need to economically and efficiently provide remote terminal access has emerged as an important need. A connection between TSO and the ARPANET is thus proposed.

INFORMATION TRANSFER PATHS

Front-end Terminal Access

Presently, terminals can access the front-end system through two systems (see Fig. 3). First, local terminals can make a dial-up connection through ports on the front-end which support asynchronous modems at 110, 300, or 1200 baud. Connection is made to a terminal handler task running in the front-end. Through the use of key-words, the terminal handler will:

- establish a connection with a requested command process;
- break a connection with the command process previously requested;

- handle delete character and delete line characters;
- transmit input and output data between the terminal and the command process to which a connection has been established.

One of the available command processes is User Telnet. If the following entry was made from the terminal,

↑TELNET

the terminal handler would establish a connection to the User Telnet task running in the front-end. User Telnet is an ARPANET protocol which provides terminal access to remote host computers which have available corresponding Server Telnet implementations.

In a similar manner terminal connection can be made to the file transfer protocol process task. The FTP task allows data files to be transferred between remote ARPANET hosts and ANL facilities supported by the task. The FTP protocol, however, requires input parameters to describe the transfer. They are supplied by the terminal which communicates with User FTP and initiates the transfer.

Other command processes which have been proposed but not yet implemented include a TSO process. This would allow terminals to be handled and multiplexed on a single line connected to a complementary protocol handler running under TSO/TCAM in the model 75. Terminals could thus communicate with and appear to TSO as if they had made a dial-up connection directly to the model 75.

Command processes are also planned for providing connection to the MATHLAB and Intra-Laboratory Networks. The MATHLAB process will multiplex the data onto a single communication line in a manner identical with the TSO process. Since the front-end will be a node on the Intra-Laboratory Network, connection between the process and the network will be internal to the front-end through inter-task communication.

The second method of terminal access to the front-end is provided by a Server Telnet Protocol implementation. This allows remote ARPANET sites with User Telnet implementations to have terminal access to the ANL terminal handler. These remote terminals are thus provided with the full capability of requesting a connection to any of the ANL front-end command

processes. A remote ARPANET terminal on say Multics at MIT could establish a connection to ANL and request a connection to the MATHLAB process. That terminal would then appear to MATHLAB as if it were a local terminal connected directly to MATHLAB.

File Transfer Facility

One of the file transfer paths in the ANL front-end system is handled by the User FTP process. The transfer takes place between a remote ARPANET host and a selected local facility supported by the ANL User FTP process. File transfers are initiated and controlled by terminal access to User FTP.

The set of possible file transfer paths is shown in Fig. 4. Once all of the presently planned processes have been implemented, files will be retrievable from the ARPANET to:

- a line printer peripheral of the front-end,
- the terminal that initiated the transfer,
- the batch input stream of the model 195, or
- a model 195 stream that will store the file as an OS dataset.

Similarly, the front-end provides the capability to store files to the ARPANET from the following local sources:

- a card reader peripheral of the front-end,
- the print and punch output stream of the model 195, or
- a model 195 stream that retrieves files written as OS datasets.

The method of providing for file transfer into the CCF model 195 is through the use of a HASP workstation package (see Fig. 5). The package runs as a series of VORTEX tasks in the front-end system. These tasks communicate with ASP through the EMU emulator using the HASP multileaving protocol.

The workstation tasks were modified from a standard implementation in that the input and output is not to reader, punch, and printer peripherals but rather to other front-end processes. This allows data transfer between the workstation and other networks supported in the front-end. At the present time the workstation is supported as a resource for User FTP. Not only can files be transferred between ARPANET and the model 195,

but console inquiries can be initiated from remote terminals as well. These terminal requests are treated by the workstation process as if they were standard HASP workstation console requests.

Other modifications to the standard workstation concept and to RJP, the HASP workstation support in ASP, will allow files to be written and read on the model 195 as OS datasets. This extends the workstation support beyond the normal batch input and printer or punch output support characteristic of a standard workstation implementation.

One of the important requirements of the Intra-Laboratory Network (ILN) is access to the CCF. The front-end is being developed as a node on the ILN and will provide the CCF access function. The ILN will have a process which will communicate with the modified HASP workstation tasks in a manner similar to the workstation/FTP interface. All of the reader, printer, punch, console, and file transfer streams defined by the workstation are placed in an available pool and are dynamically requested by and allocated to the ILN and FTP processes as they are needed to initiate transfers. Data transfers can thus be easily initiated between the CCF and ILN.

CONCLUSIONS

I have presented the front-end as a multi-function system. The system acts as a communication front-end, as a network-to-network interface, and as a network node providing network control functions. The system processes include HASP workstation and IBM 370X emulators, small network interface modules, and larger network protocol and network control modules such as those used to communicate with ARPANET.

The system is presently operational with a connection to the CCF and ARPANET in production use. The design is flexible in nature and other ANL network interfaces are being developed. It is envisioned that connection to other national or international networks will be made in the future. The design of the system is such that the front-end is evolving as a large switch mechanism with inter-task communication between processes supporting communication and network interfaces. The system appears to provide a common interface in which network connections can easily be made.

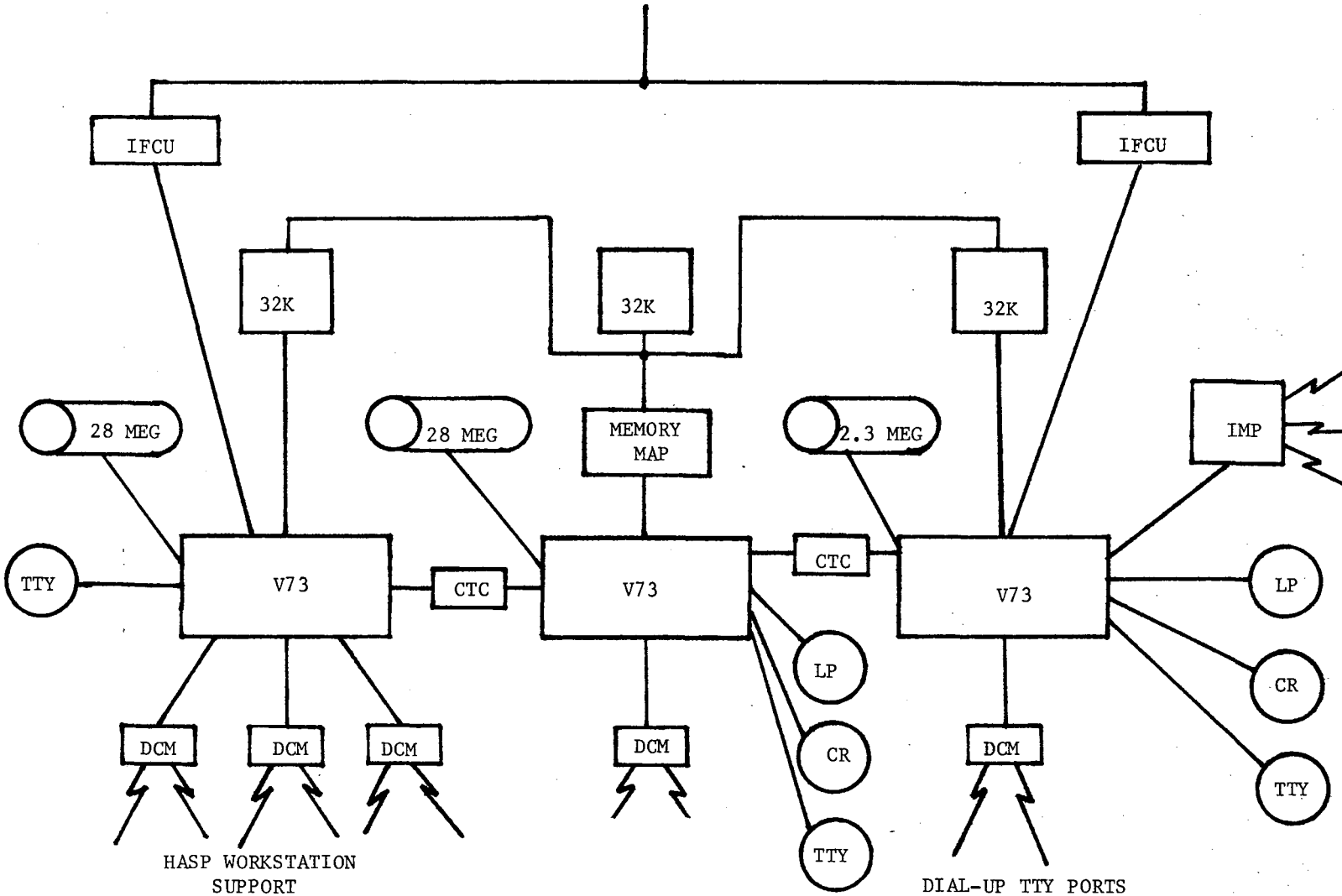
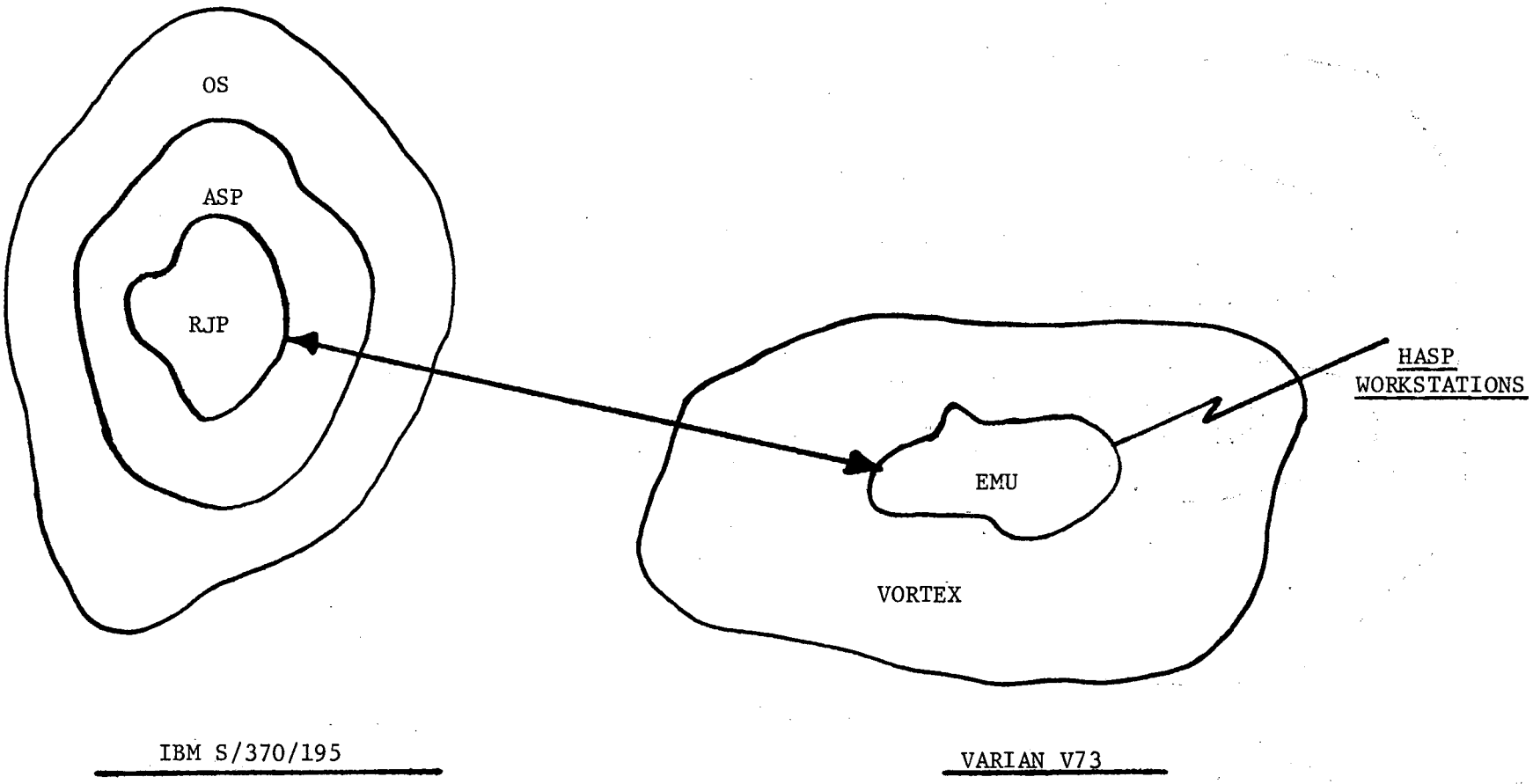
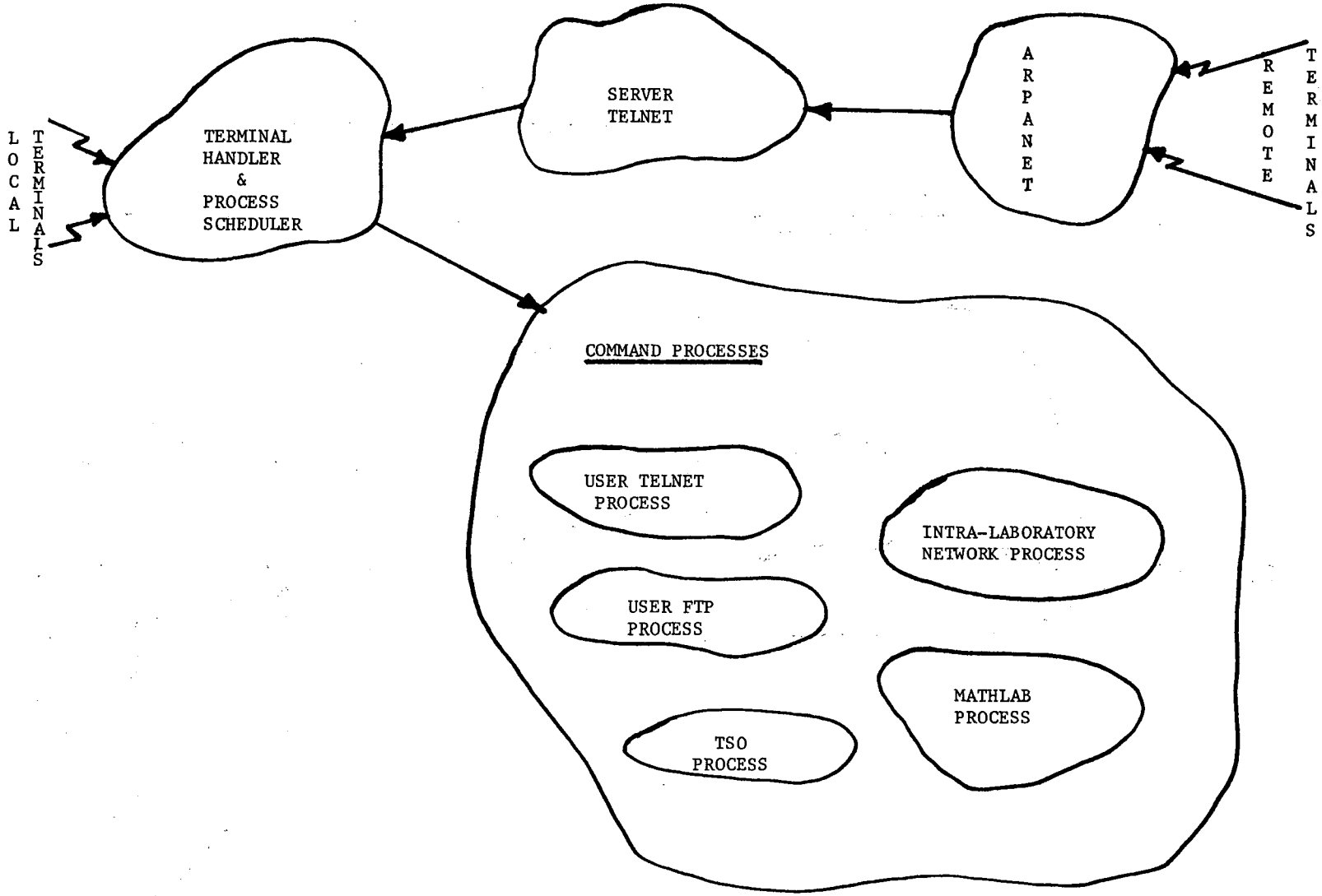

S/370/195 SELECTOR SUBCHANNEL
IFCU
IFCU
32K
32K
32K
28 MEG
28 MEG
MEMORY MAP
2.3 MEG
IMP
ARPANET
TTY
V73
CTC
V73
CTC
V73
LP
CR
TTY
LP
CR
TTY
DCM
DCM
DCM
DCM
DCM
HASP WORKSTATION SUPPORT
DIAL-UP TTY PORTS
ANL FRONT END

Figure 1



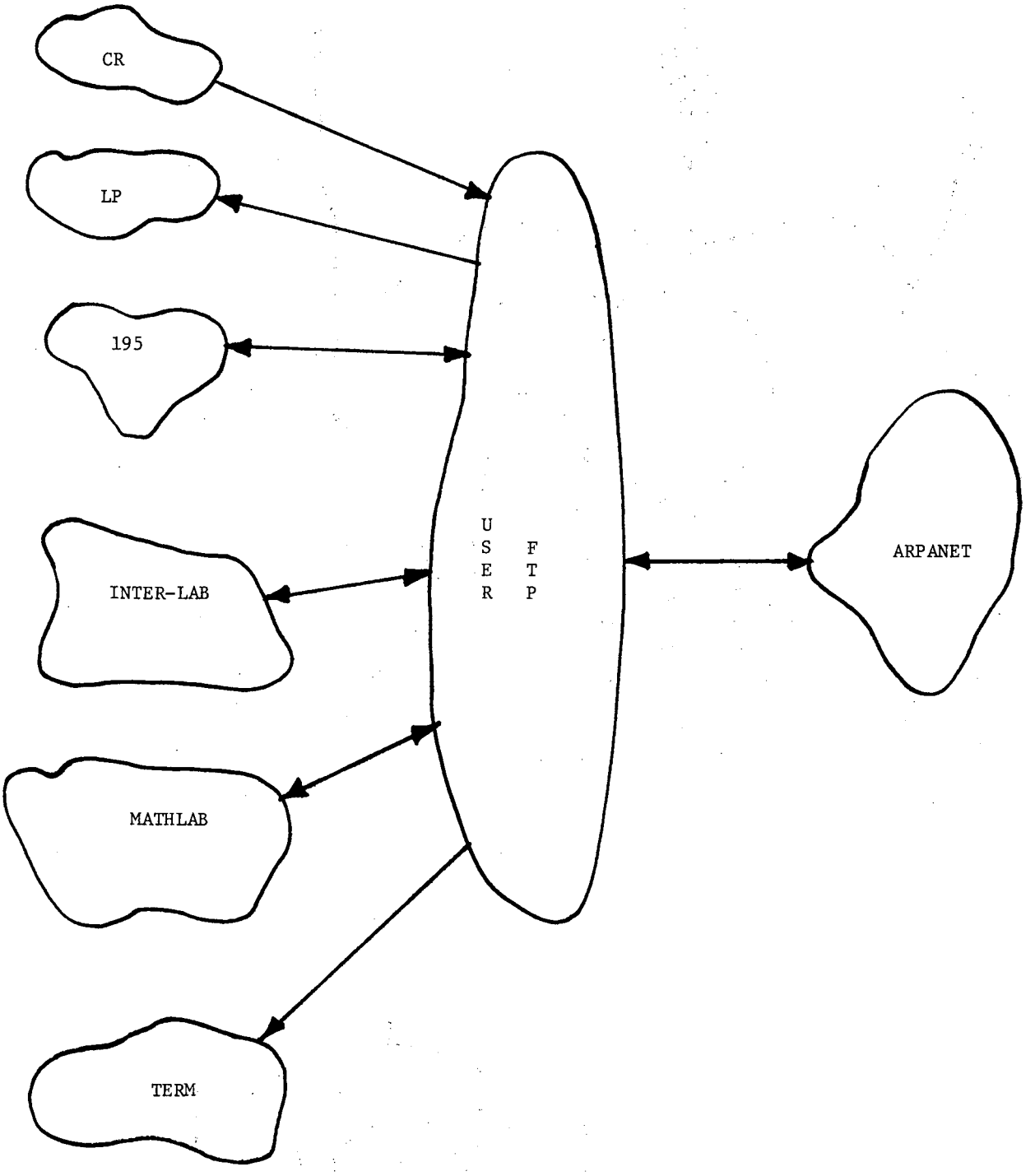
FILE TRANSFER PROCESSES

Figure 2



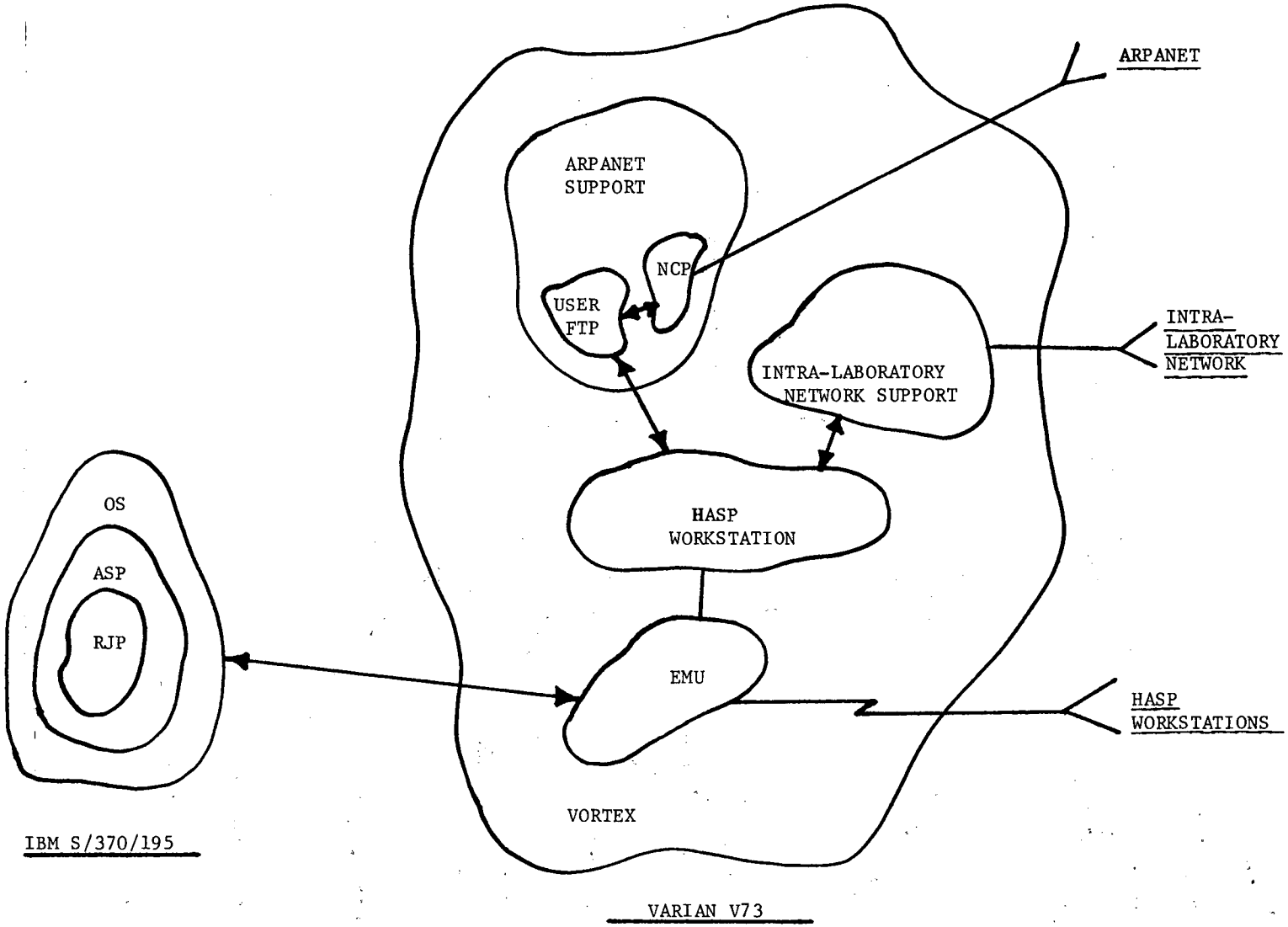
TERMINAL TO PROCESS COMMUNICATION

Figure 3



FILE TRANSFER PATHS

Figure 4



FILE TRANSFER PROCESSES

Figure 5

AN ARPANET FRONT-END FOR LARGE COMPUTERS

Ed Franceschini

Courant Institute of Mathematical Sciences
New York University

ABSTRACT

The ELF system of David Retz was written to provide access to the ARPA network for communities of users which did not have a HOST computer. As such it incorporates support for many of the standard network protocols in a PDP-11 minicomputer. ELF thus provides a good foundation for an ARPANET front-end system. This paper describes certain extensions to the ELF system which provide time sharing and file transfer access to an arbitrary HOST computer. The emphasis is on the external specifications which define the interfacing requirements to be met by the prospective HOST system, rather than on internal design. For illustration, a current implementation using a CDC-6600 as HOST will be described.

This work was performed under contract AT (11-1) 3077, Energy Research and Development Administration.

1. SCOPE OF THE DISCUSSION

This paper describes the external specifications of the first of a two stage ARPANET front-end system currently under development at the Courant Institute of Mathematical Sciences. It is a preliminary document intended to provide material for discussion to groups which may be considering using this or a similar method for interfacing their HOSTS to the ARPA network.

No attempt will be made at this time to consider issues such as motivation for the front-end approach, internal program design, performance evaluation, or operational procedures. At a later date a more comprehensive document covering these topics will be prepared.

2. BACKGROUND

It is assumed that the reader is familiar with the ARPA network design, protocols, and access methods.

For background knowledge of the ELF system of David Retz, on which the NYU front-end system is based the reader is referred to various unpublished documents prepared by the Speech Communications Research Laboratory of Santa Barbara, California. For those not anxious to get deeply enmeshed, "An Overview of the ELF Operating System", by David Retz, is perhaps best.

However, I will mention a few things about the ELF system which will help with the matter at hand. The ELF system is written in assembly language and runs on the PDP-11. It supports various HOST TO IMP interfaces including the ANTS interface, the A Consultant interface for Very Distant Host Connection, and the DEC interface.

The system is hierarchical in design. Its lowest level, called the KERNEL is responsible for the management of hardware resources such as the processor, storage and peripherals. The second level includes the EXEC which provides terminal support, and the NCP which supports communications between processes within the ELF and processes in HOSTS at other network sites. Both the EXEC and NCP provide a set of system calls to the next level in the hierarchy which consists of the so-called user programs. The user program of interest here is TELNET, which provides conversational access to the network for ELF users, that is, it implements the User Telnet protocol.

3. THE NYU ELF EXTENSIONS

The purpose of the NYU effort is to provide access from the network to the time sharing system and file system of an arbitrary HOST computer system. (The notion "arbitrary " means that no HOST system dependencies should appear in the ELF). The ARPANET protocols needed to achieve these goals are Server Telnet and Server File Transfer. Thus, generally speaking, the NYU ELF extensions consist of programs to support these two protocols. Operationally, the programs run as detached user jobs which have access to the terminal multiplexor through the EXEC and to the network through the NCP.

3.1 HOST TO FRONT-END COMMUNICATIONS

Hardware communications between the two computers is achieved by interconnecting several ports of the respective asynchronous telecommunication device multiplexors. In effect each computer will see the other as several interactive terminals. As a future enhancement of this system we contemplate a direct channel to channel coupling, but that is beyond the scope of this discussion.

The ELF Server Telnet process and the HOST time sharing system use one physical connection for each network connection (that is, for each user), while the ELF Server File Transfer process and the HOST file system use one physical connection to multiplex all concurrent file transfer sessions.

For the sake of clarity the time sharing and file transfer functions will be considered separately.

3.2 TIME SHARING FUNCTION

Of the several ports allocated to the time sharing function, one is reserved for control messages between the ELF Server Telnet and the HOST time sharing system. Through this medium the following control functions are implemented:

1. ELF Control of output flow from HOST time sharing system.
2. ELF notification to HOST time sharing system of explicit CLOSE from user.
3. HOST time sharing system notification to ELF of normal user logout.
4. HOST time sharing system notification to ELF of auto-logout due to user inactivity.

Appendix A shows the details of these transactions.

The implementation of these functions may be easier in some HOSTS if some characters are reserved for control of flow. For example, TELENET controls flow from its HOSTS by the X-ON and X-OFF characters in the data stream. The other controls are analogous to disconnect events associated with real teletypes and can be implemented by manipulation of the modem control functions of Data Set Ready and Data Terminal Ready. We chose the control port method in order to maintain full transparency with respect to the data streams and to avoid modifying the ELF KERNEL modem drivers.

3.3 FILE TRANSFER FUNCTION

All communications between the ELF file transfer process (FTPSVR) and the HOST file system process take place over a single full-duplex teletype line (although not necessarily at teletype speed). The protocol used has been adapted from Michael A. Padlipsky's "Host-Front End Protocol" and is outlined in Appendix B.

The teletype line is driven from ELF by a new kernel driver (KDFE). This new module is a version of the standard ELF keyboard

driver (KDKB), modified to eliminate the special handling of certain control characters.

The H-FE protocol has just five message types. All message types have interpretations in both directions, although some fields have no meaning or relevance in one direction or other. In the diagrams of messages given in the appendix, each field has its constituent number of bytes marked above it.

The changes and restrictions applied to the H-FE protocol for our use are as follows:-

Field	Use
SOCKET	is always 3, indicating that only an FTP connection is valid
CONNECTION TYPE	is always 1 for duplex
PAD	is one byte long and used to indicate the type of message (see below)

Three message types are used (in the PAD field). The three possible values (in octal) are:-

10 - FTP command or reply
11 - FTP ASCII data
12 - FTP image (binary) data

The FTPSVR in ELF decodes each incoming FTP command and decides whether to process it itself, or to pass it on to the HOST. This choice is defined in the table given in Appendix C. (FTP commands not shown in this table are not implemented in the first stage of the project).

In order to facilitate communication and synchronization between ELF and the HOST, we have introduced two new "pseudo-FTP" commands called OPEN and CLOS.

Upon establishing an FTP connection, FTPSVR sends the HOST an OPEN. The HOST, if alive, replies with a 300-type greetings message. If the HOST does not reply in a certain time, FTPSVR closes the connection.

Whenever a connection is closed, FTPSVR sends the HOST a CLOS command. The CLOS never needs a reply; it terminates ELF to HOST communication associated with the particular FTP connection.

All FTP replies sent out by the HOST are passed on to the network by FTPSVR. However, they are all decoded on the way, so that ELF can keep track of the state of the connection. In particular, this enables FTPSVR to know when a data transfer is in progress. Then, if data is received when FTPSVR thinks that no transfer is in progress, it is discarded. While a transfer is in progress, all data is passed blindly through ELF.

4. AN IMPLEMENTATION USING A CDC-6600 AS HOST

The CDC-6600 at NYU is currently running the CDC KRONOS Operating system and a change to NOS is planned for the near future. Unfortunately with respect to transportability, CDC makes available three operating systems and many users have either modified these or built their own. Our operating system is close to standard except in the one area impacted by the ARPANET connection. Figure 1 shows that the ELF PDP-11 is connected to the CDC-6600 through a telecommunications front-end consisting of a Honeywell H516 minicomputer. Figure 2 is a slightly idealized illustration of the interconnection of processes in the three computers. TELEX is the name of the KRONOS time sharing subsystem and ITD is its handler for H516 communications. EXPORT is the name of the NYU remote batch subsystem and RIO is its handler for H516 communications.

The Server Telnet commands to the HOST are handled completely in the H516 so that no change has been made to TELEX or ITD.

The HOST-FRONT END protocol used for the file transfer function is implemented in the H516 but all file transfer protocol commands and data are passed on to EXPORT, which provides the interface to the file system.

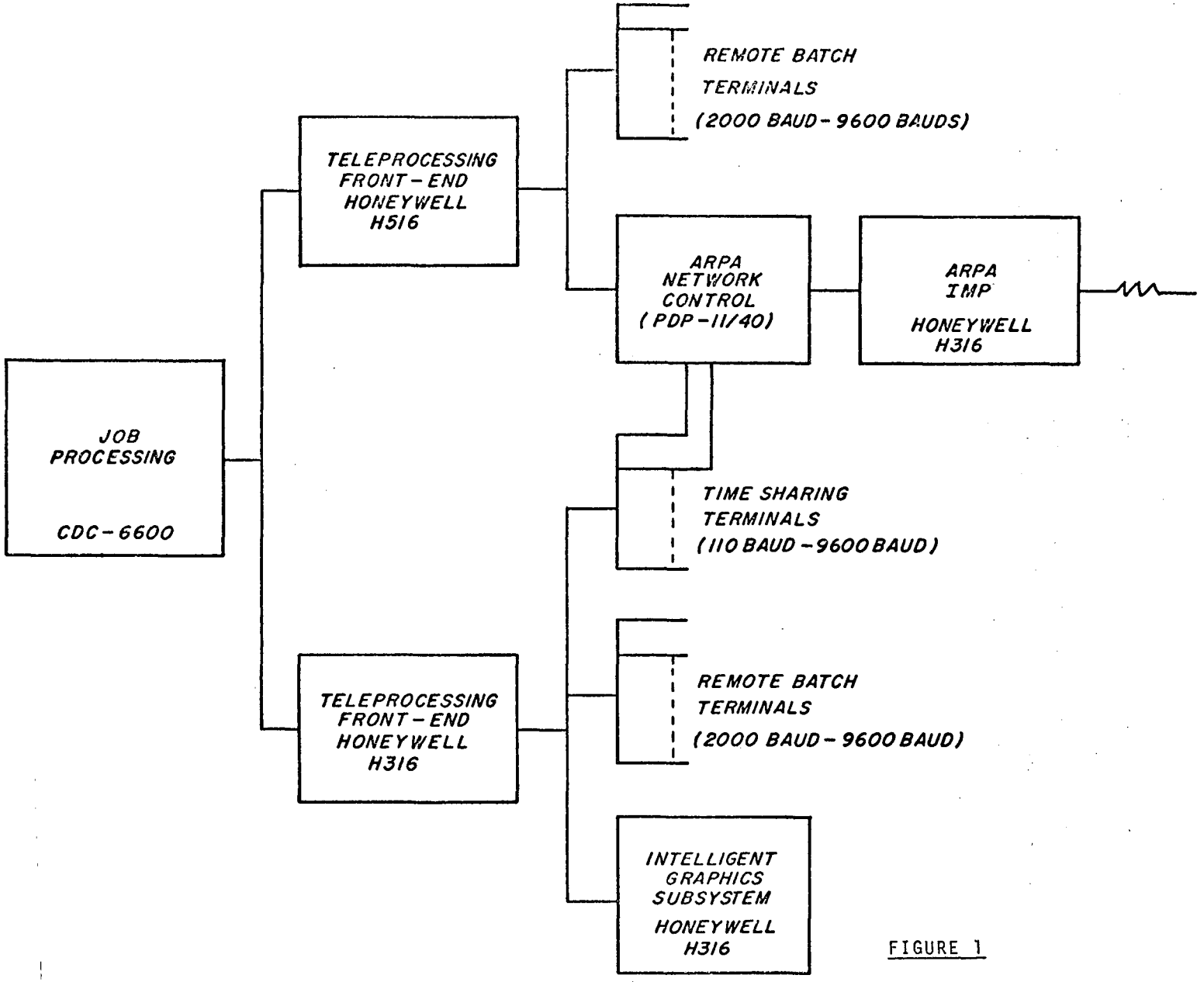

JOB PROCESSING
CDC-6600
TELEPROCESSING FRONT-END HONEYWELL H516
TELEPROCESSING FRONT-END HONEYWELL H316
REMOTE BATCH TERMINALS (2000 BAUD-9600 BAUDS)
ARPA NETWORK CONTROL (PDP-11/40)
ARPA IMP HONEYWELL H316
TIME SHARING TERMINALS (110 BAUD-9600 BAUD)
REMOTE BATCH TERMINALS (2000 BAUD-9600 BAUD)
INTELLIGENT GRAPHICS SUBSYSTEM HONEYWELL H316

FIGURE 1

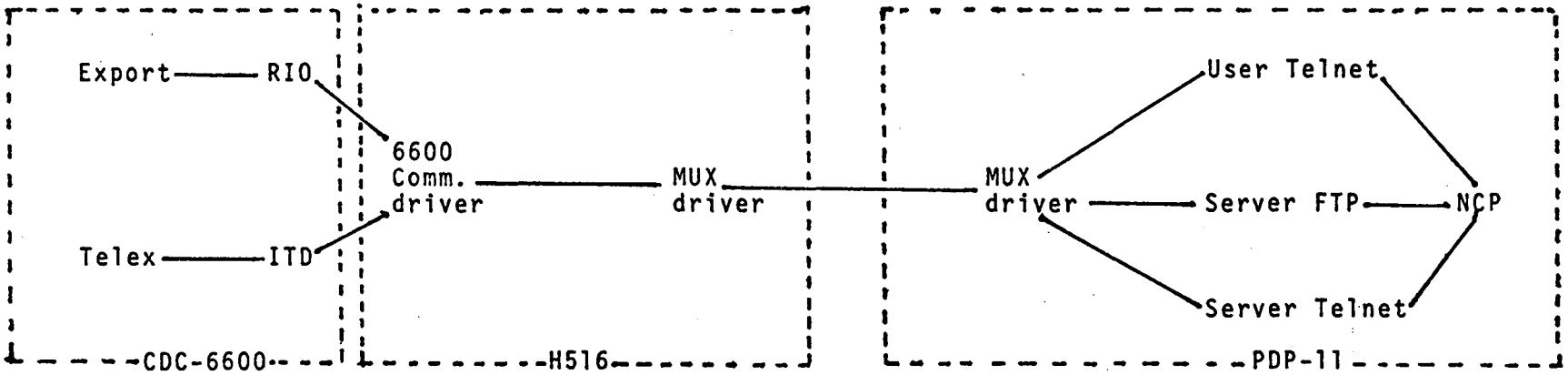
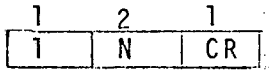


FIGURE 2

APPENDIX A

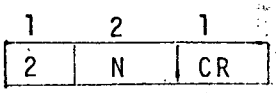
ELF Server Telnet to HOST Time Sharing System Control Messages.

SUSPEND OUTPUT



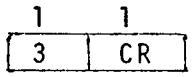
N = port number on which output is to be suspended

RESUME OUTPUT

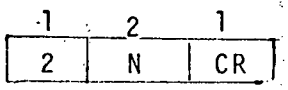


N = port number on which output is to be resumed

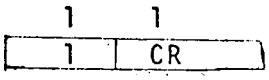
REQUEST FREE PORT



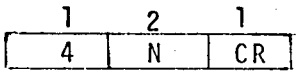
The HOST is being asked whether one of its (preallocated) ARPA ports is free. (It should be noted that if a LOGOUT has occured but no CLOSE, a port is available but ELF is not aware of it.) The HOST responds either with the number, N of an available port:



or, if no ports are free:

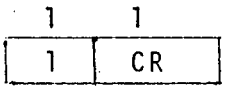


CONNECTION CLOSED



Informs the HOST that the ARPA connection associated with port N has been closed. (The HOST should treat this as if a teletype had disconnected)

In all the above cases where specific responses have not been indicated the HOST replies with the general acknowledgement:



All messages are in ASCII. Port number is a two digit octal number represented in ASCII.

APPENDIX B

SUMMARY OF MICHAEL A. PADLIPSKY'S HOST-FRONT END PROTOCOL

BEGIN

1	2	2	4	1	1
"1"	INDEX	HOST	SOCKET	TRANS. TYPE	CONN TYPE

INDEX = index of connection to which message pertains. Indexes will be assigned in sequence from 1 upwards. Index Ø refers to the global connection between the Host and the Front-End. The index field is the same in all messages.

HOST = ARPANET host number of connection.

SOCKET = 1 for a TELNET connection
3 for an FTP connection
5 for an RJE connection
77 for a site-specific connection.

TRANSLATION
TYPE = 1 for binary data
2 for ASCII data

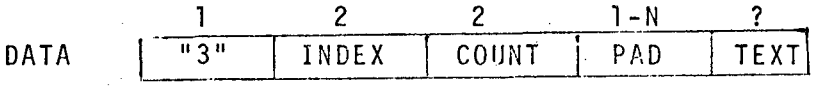
CONNECTION
TYPE = 1 for duplex
2 for simplex

NOTE: At the start of communication, the host and front-end exchange BEGIN Ø messages (in which all other fields are Ø).

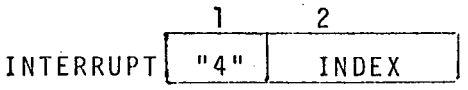
ACKNOWLEDGE

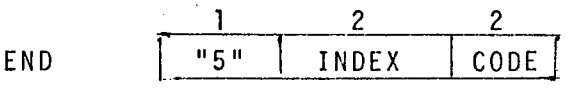
1	2	2
"2"	INDEX	CODE

CODE = Ø for success/acceptance of last message received for same INDEX
= 1 for failure/rejection
= 3 for invalid/inactive INDEX



COUNT = number of bits of data in the TEXT field.
PAD = fixed number (per-installation) of bytes used as padding to enable the text field to start on a word boundary.





CODE = 1 for general close case
= 2 foreign system has gone down
= 3 foreign imp has gone down
= 4 local imp has gone down

APPENDIX C

FTP Command Processing

FTP Command	HOW/WHERE PROCESSED
USER	HOST
PASS	Will not be implemented
BYE	HOST (ELF will respond to a BYE reply)
BYTE (8)	ELF
SOCK	ELF
TYPE (A,I)	ELF
STRU (F)	ELF
MODE (S)	ELF
RETR	HOST file system
STOR	HOST file system
NOOP	ELF
MLFL	Will not be implemented
MAIL	Will not be implemented
ABOR	HOST
OPEN	Sent from ELF to HOST
CLOS	Sent from ELF to HOST

An Alternative Front End Architecture

(Abstract)

Gary R. Grossman

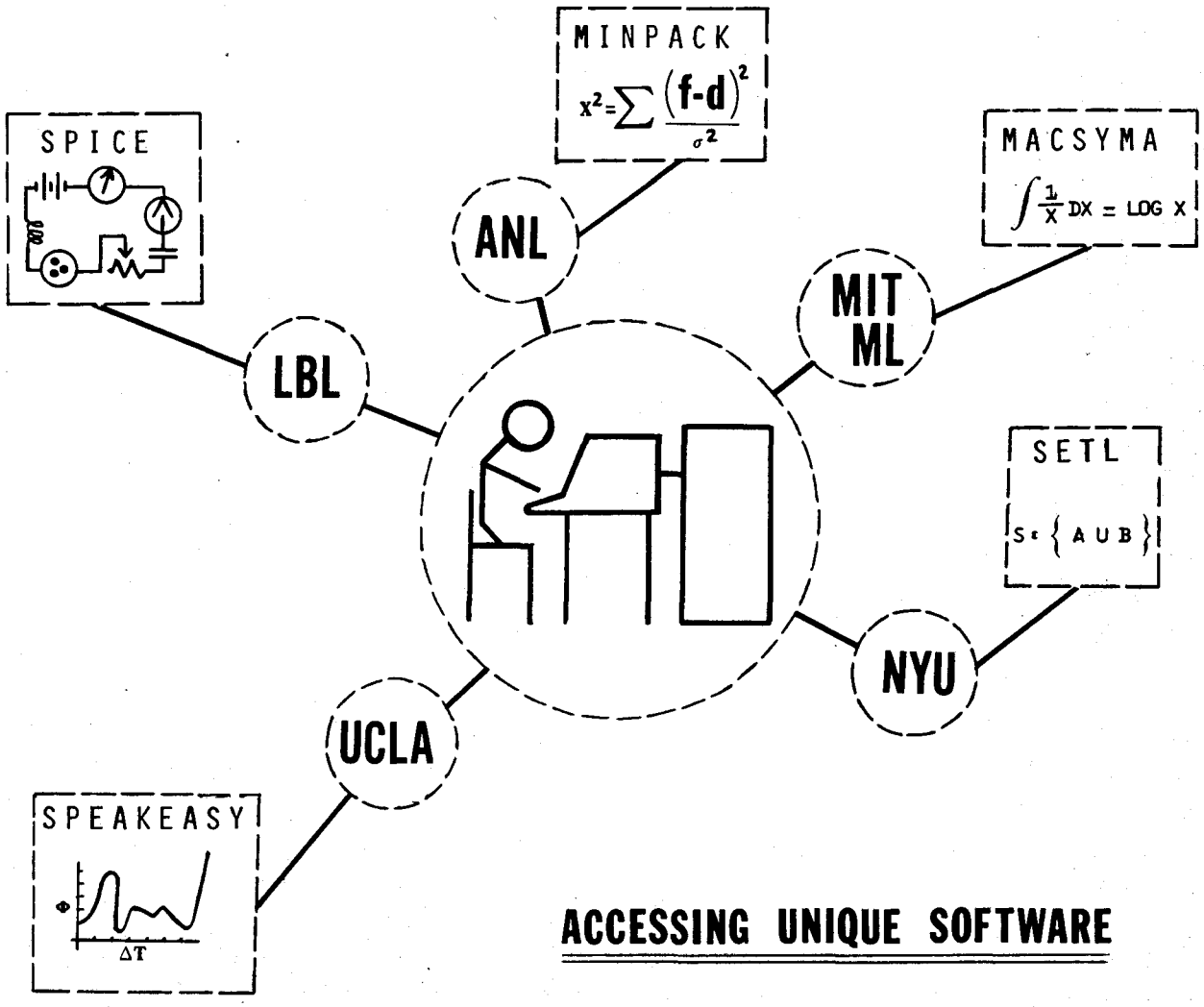
Center for Advanced Computation

University of Illinois

Urbana, Illinois

The primary design goals for a network front end are modularity and throughput. Those systems which have provided high throughput (ANTS I, ELF I) have done so at the expense of modularity. Conversely, those systems whose designs emphasize modularity (ANTS II, ELF II, UNIX) have proved disappointing in terms of throughput. This may be due to the "process oriented" nature of the designs involved and the high overhead thus incurred in inter-process communication. An architecture, called HUB, is presented which promises modularity without sacrificing throughput. In HUB, each software module is structured as a set of entry points which are assigned to classes of events affecting the module. The system schedules a "micro-task" for each event; execution of the micro-task is initiated by passing control to the appropriate entry point. The simplicity of the mechanism involves low CPU and storage overhead: measured inter-module communication times on a PDP 11/45 are on the order of 150 microseconds; the entire mechanism takes 750 words of PDP 11 code. An evaluation of the design is presented, with emphasis on software engineering implications. Finally, the effect of software architecture on protocol implementation is discussed.

II-C. VALIDATING SOFTWARE OF DISTRIBUTED SYSTEMS



ACCESSING UNIQUE SOFTWARE

JOB CONTROL IN A NETWORK COMPUTING ENVIRONMENT*

T. E. Gray
Computer Science Department
University of California
Los Angeles, California

ABSTRACT

Two strategies for the network job control problem are compared: a standardized JCL approach which attempts to make all computer systems look the same to the network user, and a universal job control language (UJCL) approach which attempts to provide a uniform and machine independent way of controlling diverse types of network computing resources using pattern matching techniques. The structure and features of such a UJCL are discussed.

INTRODUCTION

In the last few years a rather extensive literature has evolved on the subject of job control and command languages [1]. One of the major developments in computing technology which has accelerated interest in this subject is undoubtedly that of computer networking. Computer users who used to curse the design of the JCL at their local computer center now find themselves cursing the design of many different JCL's at installations all over the world. Most prior work on JCL has been oriented toward a local, homogeneous computing environment. The purpose of this paper is to propose a way of looking at job control which will allow the development of a machine independent JCL suitable for implementation in a network environment.

NETWORK JCL DESIGN GOALS

A JCL suitable for a network environment must be very flexible and powerful, but it must also be easy to use. The following specific goals would be important:

- ability to use current and projected computing capabilities in a network;
- ability to control the environment provided to the user (i.e. limitations on operations he is allowed to use, changes in syntax of input requests);
- flexible data structures and flow of control;
- procedures, both locally defined (within the job) and permanently stored;
- string manipulation, arithmetic and logical operators;
- a simple and uniform way of identifying all computing resources, and of specifying installation dependent job control information;
- machine independence, not predicated on any particular system architecture;
- minimal differences between batch and time sharing use;
- consistent use of constructs which exist in other (algorithmic) languages.

To a large extent, these goals will dictate the structure of the JCL, and will heavily influence the way in which the network computing system is viewed, that is, the nature of the abstract machine and virtual computing environment upon which the language is based.

TWO APPROACHES TO NETWORK JOB CONTROL

A job control language suitable for network use must somehow accommodate the wide variety of computing system architectures and user interfaces found in a heterogeneous computer network. The diversity of these user interfaces was strikingly illustrated to the author during recent work on networking of ERDA facilities. Compiling, linking and running a particular program in an IBM OS environment required approximately twenty JCL or TSO statements, with some containing several parameters; whereas to do the same thing at MULTICS required only four simple statements. IBM OS, however offers its users facilities, such an Generation Data Groups, which are not available from MULTICS, and it requires control of its resources at a lower level of abstraction. The IBM OS user must know about physical storage devices, cylinders, tracks, blocks, etc. These are generally invisible to the MULTICS user.

One strategy for dealing with the network job control problem is to define a standard JCL which attempts to make all the computer systems on a given network look the same to the user. There are two difficulties with this approach: First, it would be very difficult to achieve a consensus on exactly what kind of computing environment should become the standard. The high level view of resources, which makes space allocation parameters, physical devices, etc., invisible to the user is generally more convenient for the user. On the other hand, the lower level view offers the possibility of greater efficiency. The second difficulty with the standard JCL approach has to do with the technical problem of implementing an agreed upon standard set of functions on the disparate types of computing systems found in a heterogeneous network. This technical feasibility issue would strongly influence the shape of the standard computing environment. The usual result is that the standard set of functions becomes the "lowest common denominator", those functions which could be implemented most readily on the largest number of systems. Capabilities such as dynamic linking of high level language procedures could not be a part of the standard computing environment since relatively few systems support it.

There is an alternative approach to the network job control problem. Instead of defining a standard JCL which tries to make every system look the same to the network user, we may define a universal job control language (UJCL) which avoids the machine dependency and computer systems architecture problems cited above while providing a unified way of looking at and controlling diverse computing resources. The UJCL provides a uniform language structure, but not a uniform set of operations on programs, data sets. etc. The UJCL approach views job control as a problem of integrating sets of resource attributes and parameters into a description of a task. Thus, the UJCL contains primitives to manipulate resource attribute descriptions and task descriptions; the language does not know what the specific attributes of a particular resource are. The contents of resource attribute descriptions and task descriptions are machine dependent, but the UJCL itself does not ascribe meaning to the machine dependent attributes. Instead, UJCL procedures are written to associate programs with the resources needed to run them by using pattern matching techniques on the program requirements and resource attributes.

The UJCL structure and primitives must provide the flexibility and power needed to build a general routine for any of the usual machine dependent job control operations (e.g., EXEC, RENAME, COMPILE). In addition, it offers a framework upon which either a uniquely tailored or a standardized user interface can be created, while maintaining its machine independence. The term universal,

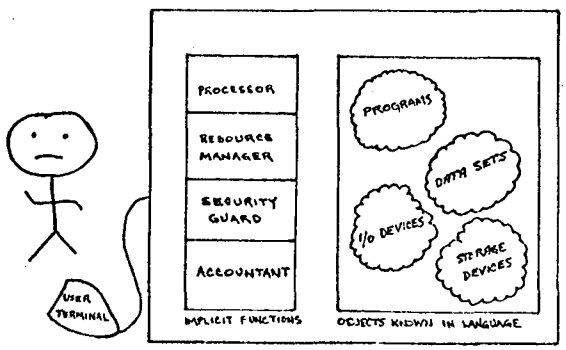
*This research was supported by the U.S. Energy Research and Development Administration, Contract No. E(04-3)-34, PA 214.

then, implies generality rather than standardization. The distinction between the standard JCL approach and the universal JCL approach can be clarified by considering the conceptual machines underlying them.

CONCEPTUAL JCL MACHINES

A language consists of three things: a set of objects, a set of operations defined on those objects, and a flow of control strategy. The conceptual, or abstract, machine for a language is a model of the computing environment upon which the language is based, and an abstract processor which performs the operations defined on the objects in the language.

Looking at job control as a formal language design problem is a relatively recent phenomenon [2] and this may explain why one does not often find explicit descriptions of the abstract machine which underlies a particular JCL design. Traditional approaches to JCL design use a model of the computing environment and conceptual machine which reflects the architecture of the particular operating system and hardware which the language is designed to control. The objects in the language generally relate directly to physical entities such as memory, I/O devices, channels, data sets, plus the program to be run. The conceptual machine upon which such a language is based allocates devices, binds logical and physical files together and runs the program. Thus, this type of JCL machine actually subsumes all of the computing resources (e.g. data sets, terminals, programs) and the processor becomes an implicit function used in the JCL machine in implementing the "execute program" primitive in the language. (This is also true of the resource manager, or allocater.) A model of such a JCL machine is illustrated in Figure 1. The difficulties in using this model as the basis for a network JCL are those inherent in the "standard JCL" approach mentioned previously: each different system architecture implies a JCL machine with a different set of objects on which job control operations are defined. If one were to try to present the network user with a uniform JCL interface, some standard set of computing resources - common to all implementations on the network - would have to be agreed upon. These resources would become the objects in the language upon which JCL operators were defined. In this model the processor is an implicit part of the JCL machine's "execute program" primitive. If used as a basis for a network JCL, the model must also account for multiple processors with widely varying attributes, and a method for specifying which type of processor is required by a particular program.



A Conventional JCL Machine
Figure 1

To precisely define the operational semantics of a language so heavily dependent on differing (and changing) architectures - as the standard JCL is - appears to be an impossibility. The UJCL machine described in the following section avoids this difficulty by moving the physical computing resources outside the boundaries of the abstract language machine.

THE UJCL MACHINE

Moving the physical computing resources outside of the UJCL machine boundary yields a conceptual model which is a suitable foundation for a machine independent network JCL. Instead of the objects in the language being actual computing resources, the principal UJCL objects are task descriptions and resource attribute sets which are obtained from a resource attribute catalog within the machine. In order to be independent of the various computer system architectures in a given network, the UJCL machine must not make many assumptions about what kind of resources exist, much less what attributes the resources have.

The resource attribute sets contain machine dependent characteristics, but they are simply abstract data structures as far as the UJCL machine is concerned. The UJCL primitives provide tools for doing pattern matching on the data structures. The person who writes UJCL procedures to associate program requirements with data set, terminal and processor attributes does need to understand something about the available physical computing resources, however.

Programs have attributes which specify the kind of run time environment that they require. In other words, the program attributes represent a "demand list" of names or attributes of other computing resources which must be made available to the program in order for it to run properly. These other resources include processors, data sets, terminals, other programs, etc. The program attributes ("demand list") will specify the characteristics of these other resources in varying levels of detail, as well as spelling out any necessary conditions on the use of the resources. For example, a typical demand list includes the program's logical file names, required data set organization and blocking attributes, device/channel separation requests, unresolved external references, and the type of processor required. The processor specification need not be a particular hardware type; it could be the name of a high level conceptual machine which was implemented somewhere on the network, such as the PL/I machine. Thus, the concept of program is not restricted to hardware executable load modules; it is perfectly consistent to consider a high level language source file as a program which, in its demand list, specifies the name of a processor for that language, in addition to file binding information, etc. A demand list can specify a class of resources rather than one specific resource.

The attributes of other resources would indicate the characteristics of that particular resource. For example, attributes of a terminal are its maximum length, number of lines per page, half or full duplex, echoing parameters, and so forth. The same idea applies to any other resources: graphics terminals, data sets, work space, devices, processors, scanners - and types that haven't been thought of yet. An additional option is for the attribute set of a particular resource to point to an attribute set for a generic class of computing resource.

The fact that processors are considered an allocatable resource, rather than as implicit functions in the JCL machine is significant. It illustrates the fact that the UJCL conceptual machine does not actually run programs; rather, it generates descriptions of the environment required by a program (task descriptions) which are

then shipped off to the resource manager for allocation. The resource manager requires that the string specifying the processor to be used map to a processor attribute set in the resource attribute catalog. The manager then transmits the task description to the allocator for that processor. The presumption is that the information in the task description will be meaningful to the local allocator. For example, the allocator for an IBM OS installation would recognize the keyword "SEP", to request that two data sets do not share the same device, but it would not recognize the keyword "absentee" which at MULTICS relates to running a program in background, rather than interactively.

Four kinds of objects exist within the UJCL machine:

1) the resource attribute catalog (accessed by the unique external resource name);
2) predefined system objects (including "standard" UJCL procedures, one of which is designated the user's process overseer, and data structures containing system defaults);
3) predefined user objects (global objects);
4) user objects defined locally (i.e. within the current job) such as task descriptions.

Just as all of the external resources available in the user's virtual computing environment bust be distinguishable, so all of the objects known to the UJCL machine (task descriptions, resource attribute sets, anything that would fall into one of the above four categories) must also be distinguishable. If all categories of object share the same name space, there can be no ambiguity, since any particular name maps to only one object. The external resource name space may coincide with the one for internal objects, since there are no operations defined on the resources themselves in the UJCL machine. Using disjoint name spaces for objects known within the UJCL machine offers some advantages from the point of view of protection of an object from unintended use. If the type of an object can be determined by its name structure, the user is less likely to mistakenly use the object, thinking that it is something else. This approach is taken at the expense of a feeling of coherence among the types in the language.

The resource attribute catalog does not contain allocation information (e.g. whether the resource is in use or currently available for allocation). Allocation information is maintained by the resource manager function in order to minimize synchronization problems. The catalog is hierarchically structured, having entries for generic resource types and sub-entries for each instance of that resource available to user. Only computing resources for which the user has previously been given administrative permission to use are included in his resource catalog. Each catalog entry contains the access privileges the user has for that resource. A function is defined which takes a resource name and returns its set of attributes. The UJCL user can obtain resource attributes interactively, and build a suitable task description from this information, or UJCL procedures can be created which will automatically "fish" for needed resource attributes and build an appropriate task tree for the application.

The set of predefined system objects includes UJCL procedures and data structures. Some of these will have been defined as installation standard procedures; others will have been previously defined and stored by the user. One of the procedures in this group will be a "process overseer" or "terminal handler" procedure which is invoked when the user's machine is activated at "login" time. The process overseer procedure serves as the standard command interpreter. The block structuring needed in the language to protect variable names also allows a user to create a subordinate command processor which could provide a tailored computing environment for a particular project. The standard process overseer would also be coded to allow the user to specify his own command processor at login time.

User defined objects include task descriptions, UJCL procedures, default data structures, etc. There is a UJCL primitive for permanently saving a user defined object. This would most frequently be used for storing UJCL procedures or the user's own default data structures.

The resource attribute catalog, predefined system objects and predefined user objects constitute the user profile. The user profile is the unique set of objects which become available to a particular user after successful "login" or "job start". The UJCL "login" primitive takes a string representing the user's identification and authentication and returns the appropriate profile. Provision is made for preventing accidental or malicious modification of the resource attribute catalog and predefined system objects, and for insuring that the latest versions are available to the user, even if updates are made during a job or interactive session.

Figure 2 depicts the relationship of the objects inside the UJCL machine with the computing resources in the surrounding virtual computing environment.

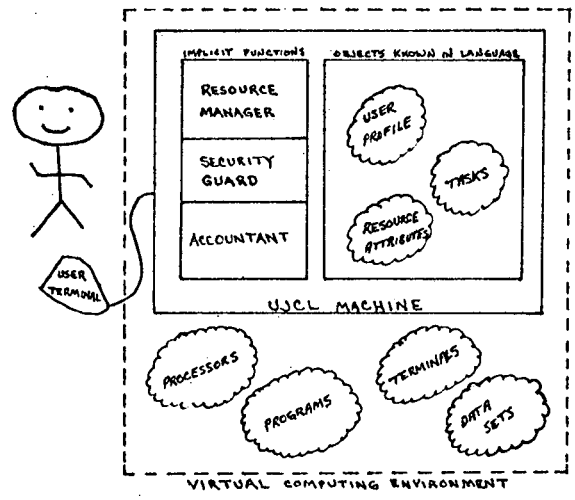


Figure 2

UJCL DATA STRUCTURES AND OPERATORS

In order to give the UJCL user sufficient flexibility, the objects in the language are trees. The class of objects is subdivided, in a manner similar to the Vienna Definition Language [3] abstract trees, into composite objects and elementary objects, or atoms. Atoms are varying length character strings, and the atom class is further divided into selectors, booleans and arithmetic types. A composite object is defined recursively as a set of pairs where the first element of the pair is a simple selector and the second element is either an atom or a composite object. A simple selector can be thought of as the label on a path between two adjacent, connected nodes of a general tree. A complete selector would be the entire pathname from the root of the tree to the desired sub-tree (it need not be a path to an atom). Each pair in a composite object is represented graphically as a sub-tree. The set of composite objects includes task descriptions and resource attribute sets. Objects are expandable both horizontally and vertically.

Selectors, arithmetics, and booleans are distinguished data types which are subsets of the string type. This minimizes conversion and I/O problems while still provid-

ing protection against user mistakes, since arithmetic functions are then only defined for strings containing numerals; boolean functions are only defined for strings containing "true" or "false"; and selection is only defined for selector strings. As might be expected from the use of general trees, an object name is not associated with any elementary data type (i.e. type free variables); however, explicit declarations are required to distinguish a new variable (object) inside a block from one of the same name in an outer block.

BUILDING A TASK

The task description tree is an object which contains all of the information required to run a particular program, plus a place to put the current status of the task (e.g. not started, running, suspended, normally terminated, aborted) and any information returned by the program. Information required to run the program includes: program id, parameters to be passed to the program, time or region or I/O request limits, logical file and corresponding data set names, file disposition information, file organization and blocking information, file/device separation requests, checkpoint requests, priority requests (or whether program is to be run in foreground or background). the type of information specified in the task description is principally a function of the program and the processor it is to be run on.

The sources of information for building a task are:

1) the program attributes (its demand list);
2) attributes of other resources (processor, data set, terminal);
3) data structures containing default information;
4) user supplied information (e.g. data set names, parameters).

The user can construct the task description without referring to the resource attribute catalog, if the necessary information is already known. The value of the resource attribute catalog is that it allows UJCL procedures to be written which will automatically build a suitable task description for a specified program using resource attribute sets, predefined defaults, and perhaps a few user supplied parameters or data set names.

For example, a file renaming procedure could be defined which, if both of the names given included the same installation id, would start a task which would invoke an operating system function at that installation to rename the file (the OS function is thought of as a program), but if the installation id part of the old and new file names differed, would build and start a task for a file transfer program. Similarly, a procedure could be written which, given the complete name of a program, processor, and data sets, would start whatever tasks were required to make the data sets local to the specified processor. The VCE model does not assume that a processor requires data sets to be local; therefore, distributed information retrieval applications are not precluded. In most cases, the data sets and processor must be closely coupled (i.e. have the same installation id). UJCL procedures could also be written to attempt to move a program to another installation, although human intervention is almost always required to get a program running at a new installation.

Once the task description is built, it can be shipped off to the resource manager for allocation by the UJCL START task operation. Once allocation is completed and the task is running, the SUSPEND, RESUME and ABORT primitives can be used on it from the terminal, or conditionally in a UJCL procedure. UJCL command processing continues while the task is running, consequently, multiple tasks can be started and run in parallel. In addition to WHILE...DO...OD and IF...THEN...ELSE...FI constructs, the UJCL flow of control structure includes a WAIT operator which will suspend UJCL command processing until the argument of the WAIT operator becomes true. In this way, a UJCL procedure can stop until a task has completed, interrogate the values returned by the program or processor, and branch accordingly.

CONCLUSIONS

The purpose of this paper is to explore the implications of taking actual computing resources outside the boundaries of the conceptual job control machine, and building instead a UJCL abstract machine around resource attributes and task descriptions. This is viewed as a good way of making the language independent of the diverse system architectures found in a heterogeneous computer network. The language structure outlined here appears to be powerful enough to handle network capabilities and flexible enough to be basis for future work directed toward establishing a standard job control language.

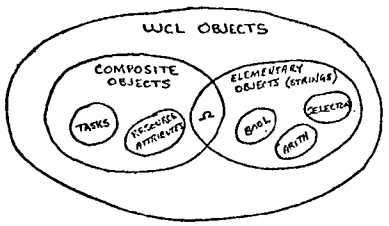
Only a brief sketch of the features which might be included in such a UJCL has been given here. Work on the formal definition of a completely specified UJCL based on these ideas is continuing.

APPENDIX 1: OPERATORS AND DATA TYPES

OPERATOR: DOMAIN → RANGE

```
START,SUSPEND,RESUME,ABORT     f:  TASK → TASK
COPY:  SELECTOR x OBJECT x OBJECT → OBJECT
SELECT:  SELECTOR x OBJECT → OBJECT
ATOM:  SELECTOR x OBJECT → OBJECT
SUBSTR:  STRING x ARITH x ARITH → STRING
CONCAT:  STRING x STRING → STRING
INDEX:  STRING x STRING → ARITH
LENGTH:  STRING → ARITH
ADD,SUB,MULT,DIV               f:  ARITH x ARITH → ARITH
AND,OR                         f:  BOOL x BOOL → BOOL
GT,LT,EQ,NE,GE,LE              f:  STRING x STRING → BOOL
INPUT:  { } → STRING
OUTPUT:  STRING → { }
FISH:  SELECTOR → RESOURCE ATTRIBUTE
SAVE:  OBJECT x SELECTOR → { }
EVAL:  STRING → OBJECT
LOGOFF:  { } → { }
```

Note: ATOM determines whether or not selected object is an atom.
EVAL interprets a string as a series of UJCL commands.
SAVE preserves a user defined object in the user profile.



REFERENCES

1. Enslow, P.H., "Operating System Command Languages - A Brief History of their Study," Proceedings of the IFIP Working Conference on Command Languages, Lund, Sweden, July 1974.
2. Barron, D.W. and I.R. Jackson, "The Evaluation of Job Control Languages," Software-Practice and Experience, Vol. 2, 143-164, (1972).
3. Wegner, Peter, "The Vienna Definition Language," Computing Surveys, Vol. 4, No. 1, March 1972.
4. Carper, C., T. Gray and C. Lai, "A Universal Job Control Language," UCLA-ENG-7544, July 1975.

MATHEMATICAL SOFTWARE IN THE NETWORK ENVIRONMENT

by

J. C. T. Pool *

Applied Mathematics Division
Argonne National Laboratory
Argonne, Illinois

INTRODUCTION

Resource sharing via computer networks is recognized as a possible method for dramatically reducing future software development costs by eliminating duplication of effort and by focusing scarce and geographically scattered expertise on areas of greatest need. Mathematical software provides a good test of this possibility since its methodology is better developed than, for example, that of systems software. Proposed collaborative ventures will, for example:

a. explore the role of computer networking in the development and evaluation of mathematical software;

b. establish the technical and operational problems of installing, maintaining, and accessing mathematical software packages and libraries remotely; and

c. examine computer system characteristics that impede software portability.

The objective of this presentation is to stimulate discussion of existing related experience and plans and, thus, to gain insight necessary to better define the mathematical software network experiments.

MATHEMATICAL SOFTWARE

The scope of "mathematical software" must first be clarified. Although programs for solving mathematical problems were developed simultaneously with the programmable computer, the term "mathematical software" was introduced by John Rice in 1969 to denote "the set

---

* Work performed under the auspices of the U. S. Energy Research and Development Administration.

of algorithms in the area of mathematics" (1). We will focus, however, on numerical computation rather than symbolic computation. Mathematical software currently is interpreted primarily to include only subroutines, the building blocks of application programs. With continued progress in understanding the creation of mathematical software, we anticipate the expansion of the field to include application programs and problem-oriented programming systems.

Creation of mathematical software involves three distinct steps (2). First, theoretical numerical methods are developed. Although crude computer programs are often written to verify gross properties of the methods, emphasis is upon error analysis and theoretical proofs of, for example, convergence. Second, one or more theoretical numerical methods are combined into a computational algorithm designed to solve a particular problem. This is a subtle process requiring an understanding not only of the theoretical properties of the methods, but also the interactions between the methods and the computer environment, e.g., the possibilities for underflow, overflow, and roundoff errors. For example, the quadratic formula is a theoretical method for finding the roots of a quadratic equation; however, reformulations of the basic formula are required to obtain a computational algorithm (3). Traditionally, numerical analysis has emphasized theoretical methods; however, the interplay between algorithms and computer architecture plays an increasingly important role in numerical analysis. Although computational algorithms are typically presented in an algebraic computer language (e.g., FORTRAN or ALGOL) and published in a refereed journal, the emphasis is usually upon the computational behaviour of the algorithm. The third, and final stage, is the incorporation of the computational algorithm into (numerical) mathematical software, an operational computer program and its documentation. A computational algorithm can be realized as mathematical

---

(1) Rice, J. R., Editor, Mathematical Software, Academic Press, New York (1971)

(2) Cody, W. J., "The Construction of Numerical Subroutine Libraries", SIAM Review 16, 36-46 (1974).

(3) Forsythe, G. E., "Pitfalls in Computation, or Why a Math Book Isn't Enough", American Mathematical Monthly 77, 931-956 (1970)

software in many distinct implementations with varying attributes.

ATTRIBUTES OF MATHEMATICAL SOFTWARE

Forsythe's comments on algorithms (4) apply equally well to software:

> "Most amateur algorithm writers ... seem to think that an algorithm is ready for publication at the point where a professional should realize that the hard and tedious work is just beginning."

During the development of EISPACK and FUNPACK, five attributes of mathematical software were enumerated as criteria of quality (5):

1. reliability - the ability of a subroutine to perform a well-defined calculation accurately and efficiently;

2. robustness - the ability of a subroutine to detect and gracefully recover from abnormal situations without involuntarily terminating the computer run and providing appropriate default numerical results and precise diagnostic information when necessary;

3. structure - the logical flow is easily traced;

4. usability - the interface with the user and the interface with other software allows the user to select the subroutine appropriate for his problem and apply it to solving his problem; and

5. validity - the existence of evidence that the subroutine has performed well in particular computer environments and availability of testing aids necessary to demonstrate that the present installation is performing well.

---

(4) Forsythe, G. E., "Algorithms for Scientific Computation," Communications ACM 9, 255-256 (1966)

(5) Smith, B. T., et al, "The NATS Approach to Quality Software," pages 393-405 in Evans, D. J., Editor, Software for Numerical Mathematics, Academic Press, New York (1974)

Implicit application of these criteria to subroutines during the late 1960's resulted in the recognition of mathematical software as a field of research. Their explicit application to "systematized collections" of subroutines was a significant conceptual advance.

SYSTEMATIZED COLLECTIONS

Examination of the subroutine library in your Central Computing Facility will reveal few isolated subroutines. These libraries typically contain several subroutines overlapping in intent and purpose, although they seldom contain sufficient subroutines to completely cover a particular problem area. When choices between similar subroutines are required, the user seldom finds documentation to guide his selection process. When similar subroutines exist, their calling sequences often vary and, thus, make substitutions difficult. These characteristics are independent of the quality of the individual subroutines. They relate to a subset of the library as an entity and the interactions among the elements of this subset.

The realization that a collection of subroutines should possess the same attributes as the individual subroutines was a major turning point for the mathematical software activities at Argonne. This observation, in retrospect, an obvious conclusion, was the birth of the concept of a systematized collection, a set of subroutines which collectively solve a defined spectrum of problems and which collectively and individually reflect the previously defined attributes of quality software.

The reliability and robustness of a systematized collection result from the reliability and robustness of the individual elements and the structure of the collection. If, for example, elements of the collection are used successively to solve a problem, then additional error detection facilities are required.

Structure is, thus, not only the most apparent attribute of a collection, it is also instrumental in achieving the others. It, for example, requires that calling sequences and parameters be uniform and that similar computational algorithms in different elements be implemented uniformly.

Usability again refers to the interface of the package to the user and with other software. Good structure insures proper interface among the elements of the package. However, comprehensive documentation is

required at the beginner, intermediate, and expert user level to insure usability. A structured document, such as the EISPACK Guide (6), must:

a. delineate the problem set addressed by the collection;

b. from a description of his problem, direct the user to the appropriate elements of the collection;

c. if alternative elements exist, enumerate the advantages and disadvantages of each approach; and

d. provide performance information applicable to combinations of elements of the collection and, thus, not contained in the documentation of individual elements.

The user's guide, thus, provides the user with a decision tree (7) together with any additional information necessary for using the collection to solve his problem. A "poly-algorithm" approach may also enhance usability of a collection. For example, a control program (EISPAC in the case of EISPACK) may automate the selection of elements according to the decision tree and load and execute the selected elements. Another example is the provision of pre- and post-optimal analysis aids in solving optimization problems, for example, a starting point generator.

Validity for a collection is an extension of validity for individual elements accounting for their interactions. Since networks may play a significant role in the validation process soon, we will return to this topic in a later section.

---

(6) Smith, B. T., et al, Matrix Eigensystem Routines - EISPACK Guide, LECTURE NOTES IN COMPUTER SCIENCE, Volume 6 (Second Edition), Springer-Verlag, Heidelberg (1976)

(7) Fletcher, R., "Methods for the Solution of Optimization Problems," Computer Physics Communications 3, 159-172, (1972)

THE ROLE OF NETWORKS IN THE DEVELOPMENT OF ALGORITHMS

Networks can obviously serve as a communications media throughout the mathematical software development process. In the development of theoretical methods, networks can provide access to powerful symbolic manipulation capabilities, for example, MACSYMA, not available among the capabilities offered by most Central Computing Facilities. During the development of computational algorithms, networks offer access to two distinct resources. First, the developer of a computational algorithm may access prototypical implementations of theoretical methods existing at other sites. Second, he may require access to computer systems with varying architecture to evaluate the selection of theoretical methods and their combination into a computational algorithm. For example, in the case of special function evaluation, access to various computer systems may be required to determine values of machine-dependent parameters in expansions. In the case of linear algebra, access to virtual memory or vector processors may provide valuable insight yielding algorithm designs capitalizing on these features. Traditionally, both classes of investigations have been 1) not attempted, 2) done painfully slowly by exchanging tapes and printed output by mail, or 3) done only by groups with local access to required computing systems.

Initial experience with use of the ARPANET by staff at Argonne is encouraging and, indeed, has motivated plans for additional experiments. We will briefly sketch some experience related to the MINPACK Project. While the long-term objective of the MINPACK Project is a systematized collection of programs to solve nonlinear optimization problems, the current emphasis is research on algorithms and software, their performance and structure.

Using the ARPANET to access the CDC 6600 at the LBL Computing Center, a pilot study was pursued (8). Two prototype programs developed for possible inclusion in an early experimental version of MINPACK were subjected to a subset of our standard test problems (9). Davidon

---

(8) Suchoff, S., "The Validation of Optimization Software - A Pilot Study," MINPACK Project Working Paper (1975)

(10) has developed new and very promising variable metric algorithms for minimizing unconstrained functionals. These algorithms make no line searches and permit quite arbitrary step directions, while maintaining quadratic termination and positive definite Hessian approximations. An implementation, OCOPTR, has been extensively tested on the ANL Central Computing Facility's IBM 370/195. Although OCOPTR is still under development, the opportunity to gain additional insight from tests on the CDC 6600 was welcomed. A more mature program was selected as the second example for testing on the CDC 6600. Early in the MINPACK Project (11), a modularized modification, GCFLRV, of the AERE-Harwell Library Algorithm VA08A was developed. VA08A, written by R. Fletcher (12), uses the Fletcher-Reeves version of the conjugate gradient technique.

Since the length of a double precision floating point representation on the IBM 370/195 is approximately equal to the length of a single precision floating point representation on the CDC 6600, all double precision variables, constants, and functions in the IBM version were converted into single precision in the CDC version. These and other minor format changes were accomplished on the ANL CCF's TSO Editor and then the two programs were exported to LBL using the ANL Remote Job Export Terminal (since the link between the ANL IBM 370/195 and the ARPANET was not yet operational). Using the ANL User TELNET to provide terminal access via the ARPANET to the Server TELNET at LBL, the two programs were tested.

---

(9) Hillstrom, K. E., A Simulation Test Approach to the Evaluation and Comparison of Unconstrained Nonlinear Optimization Algorithms, ANL-76-20 (1976)

(10) Davidon, W. C., "Optimally Conditioned Optimization Algorithms without Line Searches," Mathematical Programming 9, 1-30 (1975)

(11) Hillstrom, K. E., MINPACK I - A Study in the Modularization of a Package of Computer Algorithms for the Unconstrained Nonlinear Optimization Problem, ANL-AMD TM-252 (1974)

(12) Fletcher, R., A FORTRAN Subroutine for Minimization by the Method of Conjugate Gradients, AERE-Harwell Report R-7073 (1972)

First, a basic step size

$$H = 2**(T/2),$$

where T is the number of bits in the mantissa of a floating point number was used in tests; thus,

$$HIBM = 3.7253D-9 \qquad HCDC = 5.9605E-8.$$

A second set of CDC tests were run using the former value,

$$HCDC' = 3.7253E-9.$$

Since OCOPTR is still under development, we were not surprised to find discrepancies between the runs on the two systems, for example, dependencies upon basic step size; however, we were encouraged by the extent of the argeement in the solutions. With GCFLRV, general agreement was observed for eight test problems. The questions provoked by the differences are still under examination; moreover, a nonlinear systems solver has been recently exported to LBL for testing.

These experiments with prototype programs for MINPACK draw our attention to two additional mathematical software issues, "portability" and "performance evaluation and validation", which we address in subsequent sections.

PORTABILITY VS TRANSPORTABILITY

There is a common belief that programs written in higher level programming languages, such as FORTRAN or ALGOL, are portable between machines and also compilers for the specific language. If one adopts the definition (13):

> A program is portable over a given range of computers and compilers if without any alteration, it compiles and executes satisfying specified performance criteria on each host in the range,

then portability of quality mathematical software is a myth. Indeed, the level of reliability and robustness

---

(13) Ford, B., and Smith, B. T., "Transportable Mathematical Software: A Substitute for Portable Mathematical Software," IFIP Working Group on Numerical Software, Working Paper (1975)

of mathematical software reflects, at the current state-of-the-art, the developers' attention to the idiosyncracies of the host's hardware and software environment. Thus, the higher the quality, the harder it is to move the mathematical software to a different host, while maintaining the design specifications. Hence, with the exception of some linear algebra programs, there is little worthwhile mathematical software which is truly portable. Indeed, even subroutines in the "ANSI Version" of EISPACK contain up to two machine dependent constants (which are introduced into distributed versions by a processing program) to maintain both accuracy and efficiency. Thus, for practical reasons, the above definition of portability is unacceptable.

Although at Argonne we aim to produce "portable" FORTRAN programs whenever portability can be achieved without sacrificing reliability, robustness, and usability, it is a demonstrated fact that we cannot achieve these goals while strictly adhering to a portable subset of FORTRAN. Complex matrix routines in EISPACK on the IBM 360-370 Systems are an example: to maintain reliability and usability these computations must be done in double precision arithmetic. Since the routines do not fundamentally employ complex arithmetic, the lack of a double precision complex type in portable FORTRAN would not appear to be a serious limitation. However, the routines perform complex division and compute complex absolute value and square root. For robustness, these should be done on the IBM 360-370 Systems by the corresponding FORTRAN double precision complex operations. Overflow/underflow problems which are very difficult to overcome are encountered if these steps are programmed in real arithmetic in FORTRAN. Thus, the IBM 360-370 version of EISPACK is, of necessity, not portable. The approach adopted at Argonne allows us to develop basic prototype programs and utilize automated programming aids to produce both tailored and portable realizations of the prototype, thus, achieving the development and maintenance advantages of portability without sacrificing essential qualities of reliability, robustness, and usability.

There are basically four approaches to the portability problem:

1. primitives - hardware/software system dependent aspects of subroutines are isolated in primitive subroutines;

2. converters - adequate information is imbedded within a basic version of a subroutine through

special comments to allow a processor, the converter, to produce a version for a target host;

3. macroprocessors - macrodefinitions of hardware/software characteristics of a target host and the basic version of a subroutine are combined by a macroprocessor to yield a version for a target host; and

4. abstract form - the basic version of the subroutine is written in an abstract form (or in an operational dialect of FORTRAN and translated by a program into an abstract form) which is subjected to transformations and realized as in a concrete version for the target host.

Adopting the latter three approaches, one acknowledges that some changes in the basic version of a subroutine may be necessary. For development and maintenance, the basic and processed versions should be recognizably similar; hence, the changes should be localized and minimal in number and complexity. These remarks lead to the following heuristic definition (13):

> A program is transportable over a given range of computers and compilers if a basic version can be transformed into a version for each target host which:
>
> 1. compiles and executes to satisfy specified performance criteria;
>
> 2. the changes are performed by an automated processor; and
>
> 3. the changes are limited in number, extent, and complexity.
>
> The degree of transportability is measured by the ease of developing, implementing, and using the processor and the satisfaction of requirement 3.

Although the original GENERALIZER-SELECTOR used in the EISPACK Project was a "converter" approach, it was also the forerunner of the TAMPR System, Argonne's prototype for the "abstract form" approach (14). The converter

---

(14) Boyle, J. M. and Matz, M., "Automating Multiple Program Realizations," Proceedings, MRI Symposium on Computer Software Engineering, to appear.

approach has been refined, extended, and applied to the production of comprehensive mathematical software libraries by IMSL and NAG (15,16,17). The isolation of primitives is used in the PORT Library (18), while the macroprocessor approach has been applied to ALTRAN (19). In the heterogeneous hardware/software environment typified by the ARPANET, examination of these approaches may motivate reformulation of the underlying assumptions (20).

AUTOMATED PROGRAMMING AIDS

An underlying assumption of the mathematical software activity at Argonne is its research orientation. The questions addressed require the participation of highly trained, creative numerical analysts and computer scientists. Experience throughout the EISPACK Project revealed the need for this talent and lack of enthusiasm, among individuals with the necessary talent, for the detailed, repetitive tasks required in the production of a systematized collection of quality mathematical software. Moreover, it was recognized that "transportability transformations" where not the only class of transformations applicable to mathematical software. Consequently, the design of the TAMPR (Transformation Aided Multiple Program

---

(15) Krogh, F. T., "A Method for Simplifying the Maintenance of Software which Consists of Many Versions," Section 914, Technical Memorandum 314, Jet Propulsion Laboratory (1972)

(16) Aird, T., Battiste, E., and Gregory, W., "Portability of Mathematical Software Coded in an ANSI-based FORTRAN," Proceedings, Mathematical Software II, Purdue University (1974)

(17) Hague, S. J., and Ford, B., "Portability - Prediction and Correction," Software Practice and Experience, to appear

(18) Fox, P., "The PORT Mathematical Subroutine Library," Bell Laboratory Preprint

(19) Brown, W. S., ALTRAN User's Manual, 3rd Edition, Bell Laboratories (1973)

(20) Battiste, E. L., "Portability Assumptions," Proceedings, Workshop on Portability of Numerical Software, June 21-23, 1976, Argonne National Laboratory, to appear

Realization) System was initiated.

TAMPR is an example of an automated programming aid for the development of mathematical software. Automated programming aids may be broadly defined as any programs or programming systems that assist in constructing, modifying, debugging, verifying, analyzing, and transforming software, in the present case, mathematical software. The term is usually construed, however, to exclude general text editors and to refer to systems that embody some knowledge of the programs they process.

The TAMPR System provides a very general program transformation capability. For example, it permits software development in a locally-used FORTRAN dialect and transformation to "portable" and "tailored" dialects. It permits development of software in "extreme modular form" and removal of modularity as necessary for efficiency. Moreover, it provides a general program formatting capability which can be used, for example, to display the structure of a FORTRAN program or to unify programs in a systematized collection.

Organizing optimization software in modular form has been a central theme in the MINPACK Project. It has become increasingly evident that no single program realization of an algorithm can meet all the project goals, which range from flexibility and monitoring ability for algorithm experimentation to efficiency and transportability for general use. Thus, alternative realizations are necessary. A set of algorithm development tools, MINKIT, is required to facilitate the development of experimental versions of an algorithm. Once the algorithmic ideas are established through experimenting with and refining the implementation, a manual transformation can be made into a second realization, called the readable version. A third version, called the transportable realization, is necessary to transport codes among computers and compilers and to solve the potential conflict between structure and efficiency. The transformation from the experimental realization to the readable version requires considerable skill and, hence, must be done primarily by the human. In contrast, the transformation from the readable to the transportable realization will be done primarily with TAMPR.

Another example is the use of the BLA's, FORTRAN-callable routines that provide building blocks in the design of portable subprograms for higher level linear algebraic operations (21). Experience indicates use of the BLA's will simplify and improve software

structure and, for some compilers, will increase efficiency. In some cases, however, the BLA's reduce efficiency; therefore, for LINPACK, TAMPR transformations have been written to replace them by near optimal equivalent FORTRAN inline statements. Another example of useful transformations are those written to perform complex to real and single to double precision conversions, thus, reducing by a factor of eight the number of programs to be developed for LINPACK.

We intend to examine the feasibility of using TAMPR via the ARPANET with collaborators to capitalize on these and other examples of commonality, where transformations reduce the cost of developing, validating, and maintaining numerical mathematical software.

Expanding upon the "converter" approach, NAG (17) has developed a Master Library File System (MLFS) consisting of a set of files and three programs operating on these files. The Master Library Files include software variants a) across and between computer/compiler ranges and b) across library updates. The Anti-Editor, Editor, and Extractor/Comparer provide tools for a "predictor-corrector" approach to developing transportable, quality mathematical software through successive refinements of an initial software implementation. Initial software input to the MLFS, however, is characteristic of the converter approach. The software developer is distracted from the objective of developing quality software. He must produce, and embed in obstruse comments, a variety of information necessary for the converter to yield versions suitable for a variety of target hosts. Linking the "abstract form" approach of TAMPR to the MLFS would eliminate a substantial fraction of this detail, while retaining the desirable software maintenance and distribution features of the MLFS. Examination of the feasibility of the combined approach has been proposed via the ARPANET, since both TAMPR and MLFS are major systems not readily exported in operational form.

---

(21) Hanson, R. J., Krogh, F. T., and Lawson, C. L., "A Proposal for Standard Linear Algebra Subprograms," Technical Memorandum 33-660, Jet Propulsion Laboratory (1973)

PERFORMANCE EVALUATION AND VALIDATION

To the participants in the EISPACK and FUNPACK Projects, the most obvious application of networks in the field of mathematical software is facilitating the performance evaluation and validation studies. It is, first, necessary to distinguish between the objectives of algorithm evaluation and those of software validation (22). The objective of algorithm evaluation is to exercise an algorithm through its software implementation to identify its strengths and weaknesses. This provides information for comparing with other algorithms and their software implementations. It also aids verifying that the intent of the algorithm has actually been implemented in the software. The objective of software validation is to insure that an implementation declared to be correct in one computer/compiler environment is correctly transported to another environment and satisfies specified performance criteria. There is, of course, an overlap in these objectives. In the future, we anticipate the distinguishing of an activity in their intersection, namely, software verification, the verification that the software implements the algorithm in a logically correct manner. However, the state of program verification techniques does not yet encompass mathematical software problems and "correctness" of an algorithm implementation must be verified by "exhaustive" rather than logical testing.

The different objectives require different types of testing and test problems. Test problems for algorithm evaluation must reflect actual and expected usage to identify the gross properties of the algorithm's performance. Fundamental to Argonne's approach is the "performance profile" concept introduced by Lyness (23). Performance profile testing uses classes of parameterized test problems. Varying the parameter in a given class changes the difficulty of the problem and, thus, sampling yields statistical summaries of accuracy and efficiency, that is, reliability. On the

---

(22) Cowell, W. R., "The Validation of Mathematical Software," Proceedings, IFIP-INFOPOL-76, International Conference on Data Processing, March 22-27, 1976, Warsaw, Poland

(23) Lyness, J. N., and Kaganove, J. J., "Comments on the Nature of Automatic Quadrature Routines," Transactions on Mathematical Software 3, 65-86 (1976)

other hand, "battery" testing exercises the algorithm against a repertoire of problems. For example, in optimization we consider various topographies, dimensions, and degrees of nonlinearity. These are used to test the reliability and usability of the software implementation. Tests of robustness emphasize user abuse, for example, use of the algorithm for problems which do not satisfy the underlying assumptions. Problems are sought from "real applications" to determine potential difficulties and also to construct "artificial", but more tractable test problems, since application problems typically require extensive computer usage for testing.

Software validation, crudely speaking, seeks to insure that the transport from one computer/compiler environment to another has introduced no additional "bugs" into the original implementation (and presumably removed none). Consequently, test problems are required which exercise the software and reveal minute details of performance. An infamous case in the FUNPACK Project involves the calculation of

```
1 - K**2
```

in the elliptic integral subroutine (24). The straightforward computation, when K**2 is close to 1, is correct on IBM 360-370 Systems, incorrect on the UNIVAC 1110 due to the lack of guard digits, and sometimes incorrect on CDC 6000-7000 Series due to improper rounding. The cure in both cases is to avoid a preliminary alignment shift by calculating instead

```
(.5 - K**2) + .5
```

In its wisdom, the UNIVAC compiler optimized this desirable FORTRAN expression back into the undesirable previous form.

Performance assessment occurs in Argonne's numerical algorithms and mathematical software activities at three distinct levels. First, the algorithm or software developer exercises the software on a collection of problems of his choice obtaining typically large amounts of output to monitor the progress of an algorithm solving each problem. Second, the algorithm is exercised systematically against a

---

(24) Cody, W. J., "The FUNPACK Package of Special Function Routines," ACM Transactions on Mathematical Software 1, 13-25 (1975)

large collection of problems in the "battery" or "simulation" approach and with parameterized families in the "performance profile" approach. Third, the software is made available to cooperating "field test sites" in a test version of a systematized collection. Field testing typically involves exercising the software against examples provided by Argonne, examples selected by the test site representative, and subsequently actual usage by the user community at the test site computing facility. However, this usage is delayed until previous stages have been "successfully" completed.

Field testing holds immediate promise of fruitful application of the network to our problems. Since FUNPACK subroutines are more machine dependent than the subroutines in other PACK's, their first testing on new hosts often discloses minor problems which delay the test schedule. The ARPANET will be used to perform the initial tests on computing systems not available in the ANL CCF. This will not obviate the need for field testing, but it will expedite the process. Since adequate validation techniques are not yet available for optimization routines, the ARPANET will prove useful not only in validation studies, but also in the development of the tools required for these studies. In the case of LINPACK, portable tools exist and, consequently, the ARPANET will serve to accelerate the field testing once it begins. These cases illustrate activities pursued during the past several years in the EISPACK and FUNPACK Projects with significant delays due to problems of exchanging tapes and output.

THE ROLE OF NETWORKS IN THE UTILIZATION OF MATHEMATICAL SOFTWARE

It is the area of mathematical software utilization where the discussions of the potential role of computer networks becomes primarily speculative. It is, however, in this area where the long-term influence on mathematical software is most important, since the objective of developing mathematical software, and hence, the objective of the underlying research, is to meet the needs of the computer user. It is the author's belief that the impact will be primarily upon the "container" of mathematical software, rather than the "contents". Quality mathematical software, the program and the documentation, is structured and implemented to facilitate access via the computer. For example, the documentation for the individual subroutines of EISPACK is computer readable; indeed, generation of this documentation would have been a

practical impossibility without text editing facilities. It is organized in a consistent outline:

1. PURPOSE

2. USAGE

   A. Calling Sequence

   B. Error Conditions and Returns

   C. Applicability and Restrictions

3. DISCUSSION OF METHOD AND ALGORITHM

4. REFERENCES

5. CHECKOUT

   A. Test Cases

   B. Accuracy

For example, under "2.A. Calling Sequence", the calling sequence is displayed and then each parameter is discussed separately. The case for the subroutines is comparable. The subroutines are structured to display the logical flow of the computation and liberally commented to supplement the information intrinsic in the FORTRAN statements.

Thus, 355 pages of the 551 pages in the EISPACK Guide are immediately available as "contents" of a computerized "container" accessible via networks. The remaining 196 pages are overview documentation and performance data which could be re-formatted and made computer readable. The appropriate mode for displaying tables and charts is the critical problem, since these features of the EISPACK Guide summarize large quantities of information or guide the potential EISPACK user to the correct EISPACK path. They are, thus, central to the usability of the collection. Progress has been achieved, for example, at JPL, Bell Laboratories (18), and NAG in developing fully computer readable documentation. However, the development of portable comprehensive documentation typified, on the subroutine level, by the EISPACK and FUNPACK subroutine documents is required. Dependence upon special graphics or typesetting capabilities which are not widely available is inconsistent with the philosophy of transportable software.

Achievement of this milestone, however, must be coupled with the design of the "container" and the process for

accessing documentation and programs. Experience already indicates that the user must have access first to brief overviews of a systematized collection and then successively to more detailed information according to his immediate needs. The problem described here is not significantly different than the problem faced by the User Services Group in any Central Computer Facility which contemplates "online" documentation for browsing by the potential user of a numerical software library. The remoteness implied by network communication only exacerbates an already complex problem.

These remarks have emphasized the user interface and access to mathematical software via a network. Equally important, and perhaps more important in the long-term, is computer-computer interface and access to mathematical software via a network. The possibility of a large application program in execution at one site pausing and submitting a job to utilize mathematical subroutines resident at another site must be recognized. Indeed, RSEXEC, the Resource Sharing Executive operational on TENEX Systems, is an experimental multi-computer executive program. It currently includes not only inter-Host user-user interaction, but also facilities for managing "multi-Host" file directories and for controlling multiple "jobs" on several Hosts. Although inappropriate for large computations, it appears to be a vehicle for experimenting with computer-computer access to mathematical software.

CONCLUSION

The potential impact of computer networks on mathematical software and the process of creating, testing, and using mathematical software will not be determined until more experience is gained. Gaining this experience may, however, be almost as painful as exchanging tapes and output via mail. Indeed, the most discouraging aspect of the initial experiments is already evident. The "obstacle course" faced by the computer user is, first, even more frustrating when the computer system is located several hundred miles away and, second, enlarged by an additional layer of protocols and interfaces. The friend down the hallway with experience and example job control language collections for the local system is missing.

Fortunately, some very promising potential benefits of network utilization are equally evident and provide the motivation for crossing the first "hurdle".

ACKNOWLEDGEMENTS

This presentation is a condensation of ideas gleaned from the research staff in the Applied Mathematics Division at Argonne National Laboratory and from colleagues in the mathematical software community, in particular, B. Ford of the Numerical Algorithms Group, Oxford. The author must, therefore, assume responsibility for any "folding, stapling, or mutilating" of those ideas in his attempt to develop an overview of the interactions among high-capacity computer networks, numerical analysis, and mathematical software.

Consistent Access To Programs

Jonathan B. Postel
Augmentation Research Center
Stanford Research Institute

ABSTRACT

Users of interactive systems utilize many different programs. Each of these programs has a command language to be learned. In addition to the actual commands, there is a style to each command language. For example the way a user indicates that a command is completely specified or that she wants to cancel what has been typed thus far varies from one program to another. Other elements of the command style are recognition mode, prompt and feedback type, and help features.

The user would benefit significantly if all the programs she used had at least a consistent command style. An attempt to provide a consistent command interface to a large set of programs has been made at the Augmentation Research Center of SRI. The approach is based on a Command Language Interpreter (CLI) used as the command parser by all programs produced at ARC.

Each program designer supplies a description of the command language of her program. This description is written in a language called Command Meta Language (CML). The CML description is compiled into a grammar which is used by the CLI to parse the users input.

The CLI makes consistent across the set of programs that use it the characters used to indicate command completion, delete chararcter, delete word, and delete command; the recognition mode; the prompt and feedback type; and the help features. The help features provided by the CLI are quite powerful. At any point in a command parse the CLI can show the user the alternatives remaining. The CLI can show the syntax of a command. The CLI can call a help program supplying as an argument the current command parse state. This help program accesses a data base of descriptions of programs, their commands and argument descriptions, and other explanation of the programs concepts. This data base must be provided by the program designer, of course.

The CLI also provides for user specific options in the command style. Each user has a user profile, a small data table that indicates the users preferences in command style. The user can set certain character mappings to have alternate characters for the standard functions such as delete character. The user can specify which of four recognition modes is desired. The user can select prompting and feedback types.

This approach is being extended from the current implementation on a single machine to the multi-machine ARPA Network environment of the National Software Works (NSW).

Users of interactive systems access many different programs. Each of these programs has a command language to be learned. In addition to the actual commands, there is a style to each command language. For example the way a user indicates that a command is completely specified or that she wants to cancel what has been typed thus far varies from one program to another. Other elements of the command style are recognition mode, prompt and feedback type, and help features.

The user would benefit significantly if all the programs she used had at least a consistent command style. The user can be quite frustrated when the key used to delete the last character is different from one program to the next. This has not been a significant problem when an individual limits herself to one computer system since these conventions are usually administratively enforced. But when a user accesses programs on several computers via a network she may find the conventions of the various computer centers inconsistent and conflicting.

The elements of command style that are most amenable to standardization are control function characters, recognition mode, prompting and feedback types, and help features.

> Control function characters may be quite confusing, for example on one system the ASCII code DEL (the delete or rubout key) may be used to discard the last character entered, on another system the same code may abort the command specification completely. The functions of interest are delete last character, delete last field, delete command (abort specification), and accept or confirm the command specification. Another control function to be

standardized is the method for accessing help information.

Recognition modes may vary from requiring the user to completely enter the command word to automatically recognizing (and finishing) the command word as soon as the user has entered enough to uniquely specify the command word.

Prompting may range from simply returning the carriage when the program is ready for a new command through indicating the type of entity to be entered for each field. Other feedback information may be supplied through "noise words" inserted by the program in the command specification line.

The availability of help information to the user may vary from none at all to structured and inter-linked data bases of descriptive text.

An attempt to provide a consistent command interface to a large set of programs has been made at the Augmentation Research Center of SRI. The approach is based on a Command Language Interpreter (CLI) used as the command parser by all programs produced at ARC [1].

The CLI provides a set of recognition modes, prompt and feedback types, and help facilities. These facilities are the same for any program that is accessed via the CLI. Thus the use of the CLI as the command recognizer for a program makes that program appear at least stylisticly consistent to the user.

Each program designer supplies a description of the command language of her program. This description is written in a language called Command Meta Language (CML) [2]. The CML description is compiled into a grammar which is used by the CLI to parse the users input.

The CML description method allows the program designer a wide variety of command forms for a program's command language. (There is an opportunity for administrative review of command languages of various programs and to

administratively enforce consistency in the use of command words, but this is beyond the scope of the discussion here.) The CML and CLI method of command language definition enable command languages to be readily changed if necessary.

The CLI makes consistent across the set of programs used in conjunction with it the control function characters, the recognition mode, the prompt and feedback type, and the help features.

The help features provided by the CLI are quite powerful. At any point in a command parse the CLI can show the user the alternatives remaining. The CLI can show the syntax of a command. The CLI can call a help program, supplying as an argument the current command parse state. This help program accesses a data base of descriptions of programs, their commands and argument descriptions, and other explanation of the program's concepts. This data base must be provided by the program designer, of course.

The CLI also provides for user specific options in the command style. Each user has a user profile, a small data table that indicates the users preferences in command style. The user can set certain character mappings to have alternate characters for the standard control function characters such as delete character. The user can specify which of four recognition modes is desired. The user can select prompting and feedback types.

The recognition modes provided are:

Fixed -- The user must type N characters of the command word (no more or less) whereupon it is recognized. (N is currently 3.)

Demand -- The user types as much as the user wants of the command word then types a space, if a command word is uniquely determined it is recognized, if not the terminal bell is sounded.

Anticipatory -- The user types exactly enough

characters to uniquely specify the command whereupon the CLI recognizes and finishes the command word.

Terse -- The user types one character whereupon the CLI recognize the command word. Since not all commands are unique on the first letter, one command from the group that start with the same first letter is designated primary, and it is the one recognized. To specify a secondary command the user enters a space then enough of the command to be unique.

The prompting types provided are:

Off -- No prompting is done by the CLI.

Partial -- Only the alternative argument types are indicated.

Full -- All alternative argument types and options are indicated.

The feedback types provided are:

Terse -- No noise words will be displayed.

Verbose -- All noise words will be displayed. The user may also specify the maximum number of characters of any noise word string to be displayed.

This approach is being extended from the current implementation on a single machine to the multi-machine ARPA Network environment of the National Software Works (NSW).

The CLI now runs on a PDP-10 Tenex system in conjunction with NLS. The CLI has been coded in a high level language (L10) that can be compiled for the 10 or cross compiled for the PDP-11. The CLI has been successfully tested on the 11 (using the ELF operating system) in a "typeout" mode. The 11 had no communication protocol implemented between it

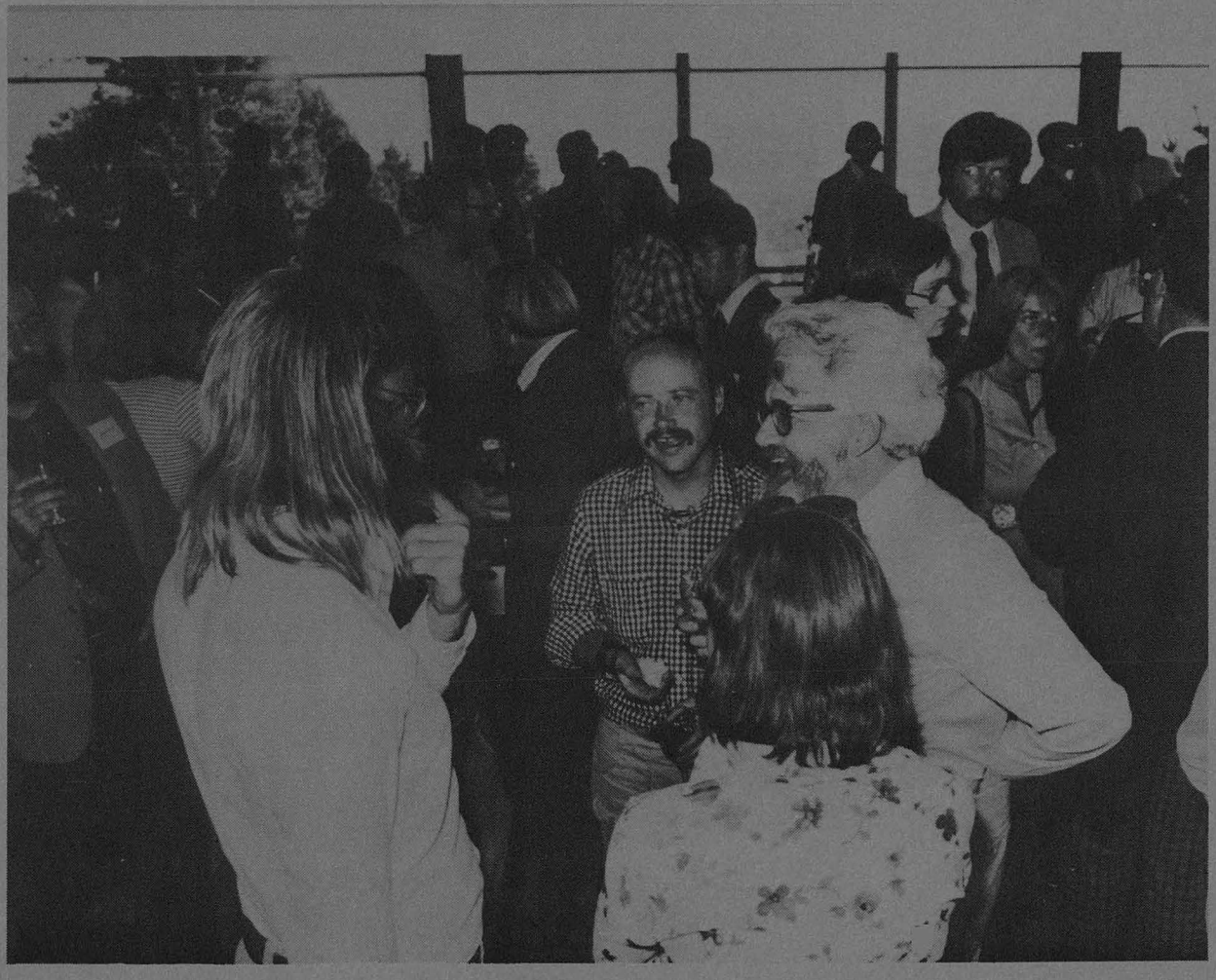
and the 10 so the parsed commands were not actually communicated to the 10.

References

[1] Irby, C. "The Command Meta Language System," ARC Journal Catalog Item 27266, 6-January-1976.

[2] Dornbush, C., K. Victor, C. Irby. "A Command Meta Language for NLS," ARC Journal Catalog Item 22130, 10-January-1975.

III. HIGH LEVEL PROTOCOLS



Extendable Information Formats*

James E. Donnelley
Lawrence Livermore Laboratory
Livermore, California

Abstract

Information formats can often be adapted for use by processes different than those for which they were initially designed. The extent to which a format can be so adapted without semantic modification is a measure of its extendability. There are many information formats used today in communication protocols, data storage structures, etc., that are severely limited in scope because of their lack of extendability. By examining some of these formats and determining modifications which can make them more extendable, a set of design principles can be extracted which may be used to generate more extendable formats.

Keywords: extend, information, format, communication, protocol, data, storage, structure, address

---

* This work was supported in part under Environmental Protection Agency contract #EPA-IAG-D5-E681-DB and in part under Department of Transportation contract #[RA] 76-12. The work was performed under the auspices of the U.S. Energy Research and Development Administration under contract No. W-7405-Eng-48.

Introduction

The need to interface various computer devices has given rise to a number of information handling and format standards. Many of these standards have been approved by standards organizations such as ANSI, NBS, CCITT and ISO, and many have been defacto standards like the EBCDIC character code and the 103 type modem. These standards have eased many interconnection issues. Until recently, however, the translation necessary to handle most nonstandard formats or protocols has been relatively minor (several hundred dollars for a hardware device or perhaps a week programming a translation routine).

The recent explosion in communication facilities has brought with it a proliferation of communication protocols. At the same time, rapid expansion in the amount of stored data has lead to a proliferation of information storage formats as accepted by various data management systems. These activities have prompted several standardization and coordination activities such as the meetings of the IFIP working group 6.1 on packet networks and the IFIP TC-2 conference on data base systems. There are also many defacto standards competing for recognition in these areas. A major difference between these standards and most past standards is the expense associated with their implementation. It is not at all unusual for a hierarchy of communication protocols or a data management system to take 10-20 man years to implement.

The Need for Extendability

In addition to increasing the importance of standards, this greater implementation expense also increases the importance of insuring that the format standards are sufficiently flexible to adapt to changing requirements without major modifications. Extendability has seldom been an important concern in past development. When a new application was faced, a new format was invented. This philosophy of inventing new formats for each application is becoming very expensive, however. Not only is the implementation of each set of format handlers becoming more expensive, but expanded communication facilities are making it desirable to translate between the existing formats. The developing network gateways and data base translators will only yield cost effective access to the available computer resources if the variety of information formats can be held in check. One way to achieve this goal is by utilizing information formats that can be extended when new applications arise.

The Similarity of Extending Communication Protocols and Data Base Storage Formats

The issues involved in extending communication protocols and data storage formats are basically the same. Both are concerned with interpretation of formated information and in each case the issues critical to extendability revolve around the interpretation of address information. In each case, to insure extendability of the information formats, it is important to have:

a. Extendable addressing schemes

b. Transparent communication and information management

c. Controlled redirection of information flow

Examples of Information Format Extension

This last section of the paper contains several examples of information formats and ways to make them more extendable. The intent is to provide principles that can be used in designing new formats so as to avoid the costly redesign and reimplementation efforts often necessitated by lack of extendability.

1. Extendable network addressing

This example concerns the addresses that are used for host computers in any of various communication networks. The addresses that are used on such systems, be they packet switched, message switched, line switched or whatever, nearly always allow addressing only of hosts directly connected to or known to the network.

A familar example of network addressing is the dialing mechanism used with most telephones. By entering a sequence of digits which is extendable in various ways, it is possible to address a great many telephones. One class of telephones that cannot be dialed, however, are those on Private Branch Exchange (PBX) systems. To reach a PBX extension, it is generally necessary to dial an operator and then verbally request the extension in a language understandable to the operator.

To see how the additional gateway protocol needed to reach a PBX extension causes problems, consider the situation faced by a computer with an automatic telephone dialer. Such a system can reach the operator, but how is it to reach the extension? The computer is being asked to be aware of and to explicitly handle

the additional gateway protocols to reach its destination.

Much more desirable from the computer's point of view is the ability to simply dial a further extended sequence of digits to reach the extension. Just as signals entered into the network as touch tones might have to be translated to pulses to reach their final destination, the digits addressing the extension may have to be translated into a visible display for an operator to effect the gateway connection.

A similar problem occurs in most digital communication networks. Some networks that have had fixed length fields for host addresses have already experienced expensive growing pains. Even more serious, however, is what happens when addresses constrained to hosts directly connected to the network become incorporated into higher level protocols.

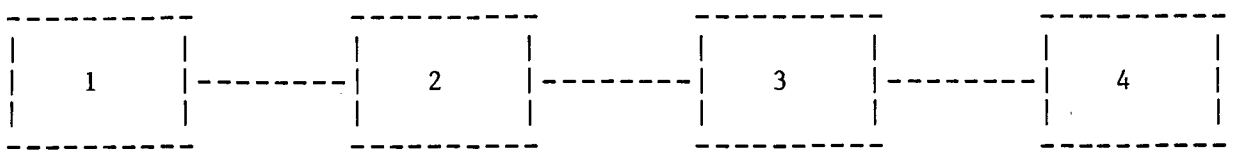
The current situation on the ARPA network [ARPA] is a case in point. Systems that are connected indirectly to the ARPA network through gateways are network nonentities. They do not have network addresses and are therefore excluded from the current higher level ARPA net protocols. A similar difficulty is faced by ARPA net hosts supporting virtual machine systems. Such systems cannot participate in the existing high level network protocols.

To satisfy a direct dialing application, some sites put in DDD telephones in addition to their PBX system. Similarly, some ARPA net sites have proposed additional hardware interfaces to the same host to supply addresses for indirectly connected systems. These are stop gap measures.

Solutions to this extendable addressing problem can be found at many levels. A high level solution based on the idea of a gate is given in [DCCS]. A low level solution suggested by the telephone analogy is given here. This mechanism uses the concept of "opening" a connection seen in many existing networks.

The key to the mechanism is the extendable nature of the address used in the "connect" message. A "connect" message contains two lists, a "to" list and a "from" list as pictured in figure 1. When a system receives a "connect" message with a nonempty "to" list, it is responsible for forwarding the connection. It does so by sending a "connect" message to the system addressed by the first element of the "to" list. The message contains the remainder of the "to" list and the "from" list augmented by the address of the host that the "connect" message was received from. Each system receiving a "connect" message is also responsible for returning to the source a small open connection number (OCN) uniquely identifying the connection and for forwarding data messages addressed with the OCN.

Figure 1 - Extended Network Addressing



to: A(3,2), A(4,3) to: A(1,2) to:
from: from: A(4,3) from: A(1,2), A(2,3)

A(I,J) denotes the port address of host I from host J

A host may be required to translate the unchanging portions of the "to" and "from" lists for interpretation by the next host, but the semantic content of those portions should not change.

This mechanism allows a host to address ANY other host in a uniform way. It has the additional advantage of limiting the address information passing into the network with each message to the small open connection number (an exception, of course, is the "connect" message). This mechanism is similar in many respects to the mechanisms for opening files on systems with directed graph directory structures. A principle suggested by this example is:

P1. All addresses should extend to address ANY destination.

2. Transparent terminal option negotiation

Many existing communication networks offer facilities for allowing different types of terminals to access interactive programs on remote host systems. A difficulty often appears in such networks when a program requires a terminal with specific characteristics (e.g. local echo, audible bell, graphic capability, extended character set, etc.). To allow terminals to access as wide a variety of programs as possible, it is often desirable to try to resolve any differences between what the program expects and what the terminal is with translation software. For example, if the program requires that the terminal echo its own characters, but the terminal does not have hardware local echo, software can be provided to echo the characters on the terminal. If, however, the program requires that there be no local echo and the terminal has hardware local echo, it is impossible for further translation to help.

To insure a proper match between terminal and program whenever possible, many networks support software to negotiate terminal options. There is an extendability problem that often appears in these negotiation protocols. Motivated perhaps by the fact that mismatches are not aggravated by the communication network unless

the terminal and program are on separate computers, protocol designers often have the option negotiation take place between the respective computer systems. If a terminal on system 1 is to use a program on system 2, for example, the two systems carry out the negotiation. When a connection is made for terminal interaction, the systems discuss the the nature of the connection to insure that it appears to system 2 as a terminal understood by 2.

The extendability problem can arise if a terminal on system 1 is to be connected to a program on system 3 via system 2. If systems 1 and 3 accept graphics terminals, but system 2 has never heard of one, the negotiation will never successfully get past the connection between 1 and 2.

The problem here is that the option negotiation is inherently an end to end phenomenon involving the terminal and the program. Only software close to the terminal and close to the program should be involved. Even in a single computer system it is meaningful for a terminal handler to negotiate for options with a program front end, but all software in between should transparently pass the negotiations. This example illustrates:

P2. Whenever possible, pass on information uninterpreted.

3. Redirecting operating system communication

The last example concerns the communication between a program and its supporting operating system. It is occasionally desirable to adapt a program that normally requests resources directly from an operating system to use other resources. For example, it might be that the file space required by a particular data base is not available on a local system. In such a case it would be useful to be able to redirect the request for storage from the local operating system to a remote system where the storage might be available on a mass storage device. Such a redirection would be particularly useful if it could be done without modification to the local file system and DBMS.

To allow such redirection of requests, some operating systems are beginning to allow user routines to handle operating system requests in a controlled manner. In the most flexible cases, all system requests are passed through standardized ports or gates that can be redirected to any desired resource handler. Such redirection can allow controlled sharing of any resources by programs that were intended for a much more restricted domain without modification of the programs [DCCS]. This example points to the need for:

P3. Whenever possible allow redirection of requests.

Conclusions

Much of the cost of implementing interpretors and translators for various information formats can be avoided by adapting fewer extendable formats. To allow adaptation of existing implementations and to insure the adaptability of future development, extension mechanisms must be understood. It is hoped that the example mechanisms and the principles given herein will contribute to this understanding.

References

[DCCS] J.E. Donnelley, A Distributed Capability Computing System, Lawrence Livermore Laboratory, Report UCRL-77800

[ARPA] L.G. Roberts and B.D. Wessler, "Computer Network Development to Achieve Resource Sharing," AFIPS Conference Proceedings 3, 36, pp. 543-549

Notice

A Host-Front End Protocol
(Abstract)

Gary R. Grossman

Center for Advanced Computation

University of Illinois

Urbana, Illinois

The overall design of a protocol which enables a front end to perform high level protocol functions for a host is presented. Problems encountered in providing an interface to such protocol services through an intermediary are discussed, including the necessity for interfaces at multiple levels of abstraction. Proposed solutions to these problems are presented. The possible impact of front-ending on high level protocol design is discussed.

Frontend - Backend Split Programs

Jonathan B. Postel
Augmentation Research Center
Stanford Research Institute

Large interactive programs are composed of two distinct types of code: the command parsing and user feedback code, and the function execution code. The command parsing part is called the frontend, while the execution part is called the backend.

The frontend is the interactive command parser, it collects command words and arguments from the user, helps the user with prompting and feedback. The frontend must to be very responsive to the user to prevent user frustration. In character at a time interactive program it is typical that the user enters one or two characters of a command word (and perhaps a character to force recognition), and the program finishes the command word and prompts for the next command word or argument. This type of interaction is very powerful and effective as long as the user is not delayed in the specification of a command.

The backend is the execution part, it computes the function specified by the user. The backend is called upon by the frontend and supplied with arguments for the basis of the function computation. The backend returns its results to the frontend for presentation to the user if appropriate. The backend often manipulates data bases and otherwise stores and retrieves information. The results of a call on the backend may depend on earlier calls on the backend in the current or earlier sessions.

Most of these large very interactive programs were designed to be used by people local to the computer where the program is run. Few of these programs were designed with the performance characteristics of heavily loaded timesharing computer systems taken into consideration. Several of these programs are now being used in the ARPANET environment on heavily loaded

computer systems. That these factors tend to cause delays in service to the user can not be disputed.

One of the major problems with timesharing computers and packet switching networks is that the interruptions in service do not occur at what the user considers natural boundaries. "Interruptions in service" means the normal scheduling of the shared resources (e.g. cpu cycles). It is disconcerting to the user that the scheduling mechanisms do not (usually cannot) take into account the users state. To the user it is frustrating to have a service delay occur during command specification. The interactive interplay of entering, echoing, and prompting requires smooth continuous service. Once command specification is completed the user is more likely to accept the delays caused by communication networks and timesharing.

In a situation where the user of the program is located at a distance from the host computer where the program executes, as could be the case in the ARPA Network, there are significant advantages to providing a mechanism for interactive command specification local to the user. This was recognized early in the development of the ARPA Network [1]. In an environment where the user has some computational resource close by, but the major computational resource is a distant host, it becomes feasible to bring the interactive command specification to the users location.

The Augmentation Research Center (ARC) of SRI is engaged in an attempt to provide such a capability. One of the requirements of this effort is a high-level procedure oriented protocol for use between processes in the computer network [2].

To see why a high-level protocol is desirable, examine a typical interactive program. As stated above it is composed of code for interactive command specification (the frontend) and code for executing the specified functions (the backend). Generally the code is mixed up and is structured by command. But one can usually identify a point where a command has been recognized and a execution function is begun. This point is often

a subroutine call. The interface between the frontend code and the backend code is a subroutine or procedure call interface. Thus if we are to separate the frontend and backend into separate modules a procedure call interface must be identified. If the frontend and backend are to be further separated by a computer network it is essential that a procedure call oriented protocol be provided.

Major advantages result from using a procedure call oriented protocol between the split frontend and backend. The backend expects to be called with arguments, and to return results. These arguments and results can be passed between the frontend and backend in a machine oriented format rather than a people oriented format. The arguments and results can be encoded for efficiency or convenience. The frontend can parse and recognize the function to be executed, it can collect and encode the arguments to that function, it can call that function (via the protocol) supplying the arguments in the proper form. The backend needs no parser, the calls on it already specify the primitive function to be carried out. The backend need not translate arguments from user language to internal language, such coding can be done by the frontend.

A significant advantage to the user of a frontend - backend split program when the frontend is run local to the user is that command recognition can be completed without any communication with the backend. This improves the responsiveness to the user and reduces the cost of communication.

An economic argument can also be made for the frontend - backend split mode of operation. The cost of processing is dropping much faster than that of communication. Thus one should optimize the cost of remote computing by minimizing the communication involved (without reducing the value of the service). The frontend - backend split mode allows the use of powerful interactive systems with substantially reduced communication costs. Costs are reduced because communication occurs only at the completion of command specification, and at the completion of command

execution. Communication overhead costs are also reduced because the communication packets are more fully utilized, since the command and all its arguments can be transmitted at once.

References

[1] Crocker, S. "Host Software," RFC 1, NIC Catalog Item 4687, 7-April-1969.

[2] White, J. "A High-Level Framework for Network-Based Resource Sharing," RFC 707, NIC Catalog Item 34263, 14-January-1976.

A Network-Wide Virtual Programming Environment

James E. White
Augmentation Research Center
Stanford Research Institute

INTRODUCTION

Early ARPANET protocol work focused on the design of a network-wide inter-process communication facility that would provide a foundation for other, higher-level protocols. This bootstrapping strategy produced:

(1) A Host-Host Protocol that enables processes to exchange messages [1].

(2) Higher-level protocols that define the particular messages by which transactions in various application areas are effected [2].

Protocols in this second category (for example, the File Transfer Protocol) interface a "user" process that requires a service to the "server" process that provides it. Each server process consists, at least conceptually, of the set of primitives or subroutines (for example, RENAME-FILE) that constitute the service, and the interface software required to interpret messages from the user process, dispatch to the appropriate subroutine, and formulate a reply upon its return. Similarly, the user process may contain like subroutines (RENAME-FILE again) whose implementation involves composing the required message, transmitting it to the server process, and returning to the caller the information contained in the server's reply.

N ALTERNATIVE APPROACH

SRI's Augmentation Research Center (ARC) has developed a prototype protocol and software framework for network-based distributed systems that eliminates the need for such application-specific interface software in user and server processes, and that raises applications protocols above the level of message

formats to the more abstract and familiar level of procedures and their calling sequences [3]. This Distributed Programming System (DPS) [4, 5, 6, 7, 8] involves:

(1) A high-level, application-independent model of distributed systems that defines a network-wide virtual programming environment in which processes can communicate with one another at the procedure call level.

(2) A network-wide protocol that implements the model, enabling processes to invoke arbitrary named procedures in one another and to supply and retrieve their arguments and results.

(3) Interface software, provided by each installation and link loaded with (or otherwise made accessible to) its applications programs, which interfaces one program to another via the protocol.

The Model

The DPS model views a process as a collection of remotely callable procedures. Each procedure is invoked by name, can be supplied a list of arguments, and returns to its caller both a boolean outcome, indicating whether it succeeded or failed, and a list of results (an error code and error message, in the case of failure). The arguments and results of procedures are modeled from a small set of primitive data types, listed below:

LIST: A list is an ordered sequence of data objects. A LIST may contain other LISTs and so can be employed to construct arbitrarily complex, composite arguments or results.

CHARSTR: A character string is an ordered sequence of ASCII characters, and conveniently models a variety of textual entities, from short user names to whole paragraphs of text.

BITSTR: A bit string is an ordered sequence of bits and so provides a means for representing arbitrary binary data (for example, the contents of a word of memory).

INTEGER: An integer is a fixed-point number in the range [-2**31, 2**31-1], and conveniently models various kinds of numerical data, including time intervals, distances, and so on.

INDEX: An index is an integer in the range [1, 2**15-1], and can be used to address a particular bit or character within a string, or element of a list.

BOOLEAN: A boolean represents a single bit of information, and has either the value true or false.

EMPTY: An empty is a valueless place holder within a LIST or parameter list.

The Protocol

The protocol that implements the model defines a transmission format for each of the seven data types above and requires that parameters be encoded in that format whenever they are transported between processes. The protocol also specifies the inter-process messages by which remote procedures are invoked. Themselves modeled using DPS data types, these messages take the following form:

```
LIST (CALL,   tid,    name,    arguments)
      INDEX=1 [INDEX] CHARSTR  LIST

LIST (RETURN, tid,    outcome, results)
      INDEX=2 INDEX   BOOLEAN  LIST
```

The first message invokes the procedure whose NAME is specified using the ARGUMENTS provided. The second is returned in eventual response to the first and reports the OUTCOME and RESULTS of the completed procedure. The presence of an optional "transaction identifier" (TID) in the CALL message constitutes a request by the

caller for an acknowledging RETURN message echoing the identifier.

The Run-Time Environment

The protocol above is implemented on each host by a "run-time environment" (RTE) that provides the applications program with a collection of primitives, implemented either as subroutines or system calls, that it can employ to manipulate the remote process. The RTE contains the code required to properly format outgoing messages in accordance with the protocol, interface with the local NCP, and parse the incoming responses. Because the protocol is application-independent, this code can be written once per installation, rather than as part of each program.

CONCLUSION

A distributed programming system like that outlined above has great potential for stimulating the sharing of resources within a computer network. First, it would reduce the cost of adapting existing resources for network use by eliminating the need for the design, documentation, and implementation of specialized delivery protocols for accessing the server process. Second, it would encourage the use of remote resources by eliminating the need for application-specific interface software within the user process. And finally, it would encourage the construction of new resources built expressly for remote access, because of the ease with which they can be offered and used within the network software marketplace.

REFERENCES

1. Mc Kenzie, A. A., Host/Host Protocol for the ARPA Network, Bolt Beranek and Newman Inc., Cambridge, Massachusetts, January 1972 (SRI-ARC Catalog Item 8246).

2. Crocker, S. D., Heafner, J. F., Metcalfe, R. M., Postel, J. B., "Function-oriented Protocols for the ARPA Computer Network," AFIPS Proceedings, Spring

Joint Computer Conference, Vol. 40, pp. 271-279, 1972.

3. Watson, R. W., Some Thoughts on System Design to Facilitate Resource Sharing, ARPA Network Working Group Request for Comments 592, Augmentation Research Center, Stanford Research Institute, Menlo Park, California, November 20, 1973 (SRI-ARC Catalog Item 20391).

4. White, J. E., "A High-Level Framework For Network-Based Resource Sharing," Submitted for publication in the AFIPS Conference Proceedings of the 1976 National Computer Conference (SRI-ARC Catalog Item 34263, NWG RFC 707).

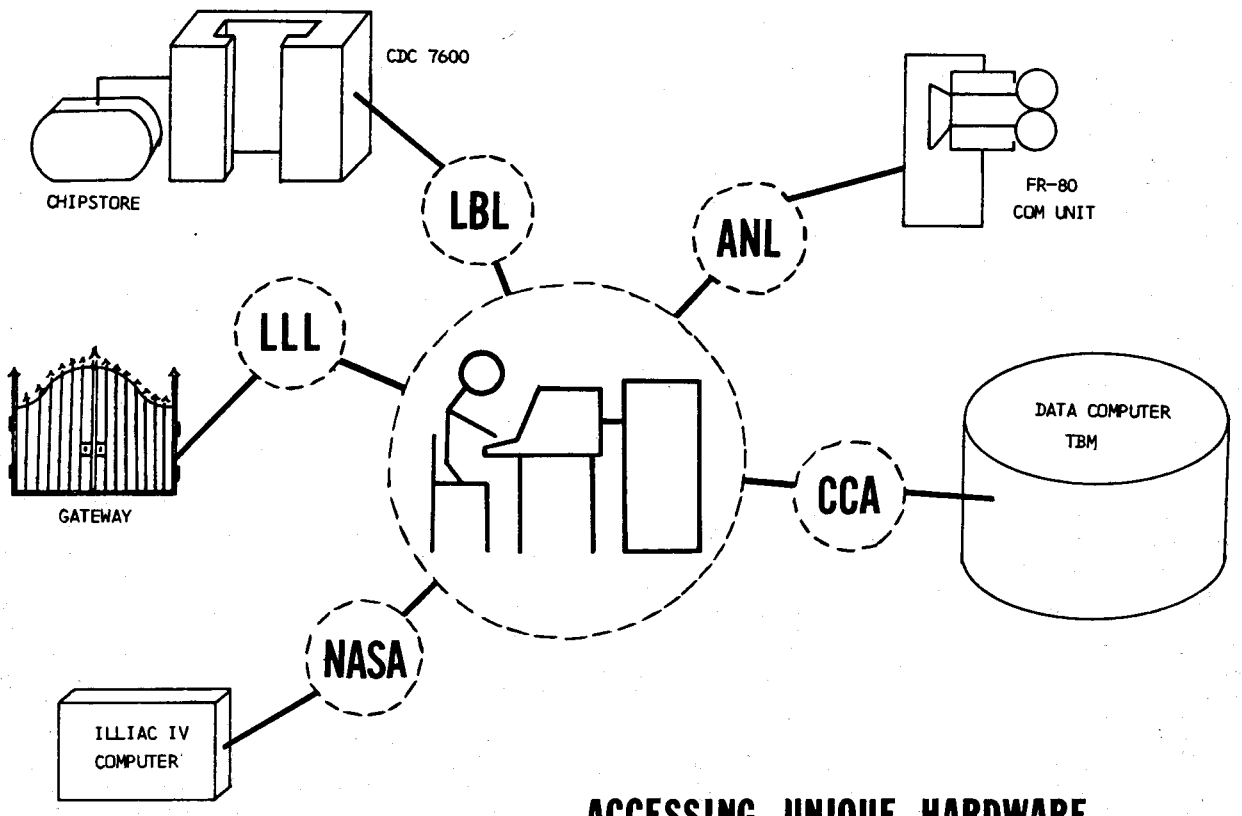
5. White, J. E., "Elements of a Distributed Programming System," Submitted for publication in the Journal of Computer Languages (SRI-ARC Catalog Item 34353, NWG RFC 708).

6. White, J. E., DPS-10 Version 2.5 Implementer's Guide, Augmentation Research Center, Stanford Research Institute, Menlo Park, California, August 15, 1975 (SRI-ARC Catalog Item 26282).

7. White, J. E., DPS-10 Version 2.5 Programmer's Guide, Augmentation Research Center, Stanford Research Institute, Menlo Park, California, August 13, 1975 (SRI-ARC Catalog Item 26271).

8. White, J. E., DPS-10 Version 2.5 Source Code, Augmentation Research Center, Stanford Research Institute, Menlo Park, California, August 13, 1975 (SRI-ARC Catalog Item 26267).

IV. DISTRIBUTED DATA BASES



ACCESSING UNIQUE HARDWARE

Data Distribution Strategies*

Peter A. Alsberg

Center for Advanced Computation
University of Illinois at Urbana-Champaign

Abstract

The Center for Advanced Computation is developing distributed data management technology for exploitation in the World Wide Military Command and Control System (WWMCCS). The WWMCCS community already has a distributed data base. Data is stored at several dozen sites around the world, usually co-located with a major military command. Data communications between these sites is by U.S. mail, special courier, and a very limited capability store-and-forward network called AUTODIN I. The Defense Communications Agency is in the process of building the AUTODIN II system. AUTODIN II will be based on ARPA network technology and will be capable of supporting vastly improved data communications between these remote sites. In the current WWMCCS environment, remote data access is infeasible. Hence, when data must be shared it is done by duplicating a data base and transmitting it to an alternate site. The time lags involved in this process can be considerable. Problems with the current environment are discussed.

With AUTODIN II on the horizon and an experimental three-node Preliminary WWMCCS Intercomputer Network in operation, "distributed data base" has become a household phrase in the

---

* This work sponsored by the Defense Communications Agency, Command and Control Technical Center, from the contract DCA-100-75-C-0021.

Department of Defense. Unfortunately, it can mean dramatically different things to different people. The interpretations range from providing terminal access to a remote command's computer system, and hence to its programs and data, up to concepts of computer-to-computer data transmission, and finally to highly automated, fully transparent network virtual data systems.

The Center for Advanced Computation is examining distributed data management techniques for a wide variety of feasible distribution strategies. Four strategies are of particular interest:

1. a single copy strategy with remote access,
2. a multiple copy strategy using a single master with multiple backups,
3. a multi-copy strategy using multiple masters, and
4. a multi-copy strategy using front-ending and caching techniques.

Single copy systems have many advantages. They are conceptually simple and hence easy to construct and manage. This is what is typically envisioned when a distributed data base is proposed. It allows the distributed data base to be viewed as a more conventional data base in which some of the segments are stored on a network virtual peripheral. That is, data segments are actually located on another host and are accessible via the network. Some theoretical work has been done on the problem of where to locate a data base segment in a network. The realities of the WWMCCS environment may impose constraints which preclude technically optimal allocation strategies. The single copy scheme offers

reasonably easy management of the distributed data base and storage requirement are minimal. Unfortunately it has poor reliability.

Single master with multiple backup architectures are desirable when greater reliability is required than that supported by single copy schemes. Military command and control is an obvious application. The backups can be maintained in a variety of ways. Two of the more common ones considered are remote journalling and active backup. In a remote journal scheme, updates are not actually applied to the backup data base when they are transmitted to the backup site. Instead, they are simply journalized. The backup data base is brought up to date after a failure occurs and the backup copy must be pressed into service. This relatively passive form of backup can offer signficant improvements in reliability at relatively low cost in terms of network traffic and processor loading and storage requirements. The backup data base can usually be stored off line. However, careful analysis of the overall availability of this kind of scheme has shown that passive backup can actually reduce overall data base availability. This is primarily due to the long times required to bring an off line data base up to date via the journal. This has led to modifications to the basic scheme for periodic update, etc. At another extreme are active backup strategies. In this case the backup host attempts to apply updates to an on-line copy of the data base as rapidly as possible. This approach substantially improves both reliability and availability but imposes significant additional processor loads and storage requirements on the network system. Hybrid passive and

backup systems have been proposed. These have a single master, several active backups and several passive backups.

Multiple master systems permit updates to be sent to any site. The receiving site will be automatically coordinated with other master copies. These systems increase complexity, response time, processor overhead, and network traffic over an active backup strategy. At the same time they fail to improve reliability or availability over the active backup approach. Hence they are always inferior to active backup approaches and do not appear to be a viable design alternative.

The fourth architecture considered is a cache architecture. In this approach there may be several front-end machines between the user's terminal and the large remote shared data base. For example, consider a three-layer system. A large host may contain a hundred million byte file which is shared by the entire user community. Front-ending this is a set of mini-computers with a few million bytes of storage each. Terminals are then attached to the mini-computers. The terminals themselves may have storage for a few hundred thousand bytes of data. Portions of the data base are paged out into the front-end then to the terminal. This approach seems to offer some advantages in reliability, response time, and, surprisingly, in cost. The cost advantages appear to stem from the fact that the amount of processor time consumed by a typical I/O operation is similar whether a maxi or microprocessor is involved. However, in one case a million dollar processor may be tied up and in the other case a thousand dollar processor is involved. The main host machine is still needed because of the

current economic infeasibility of maintaining a hundred million byte data base at each terminal. It should also be noted that the main host, front-end and terminals could all be connected by common carrier network. At each level where a shared data base is involved any of the previously discussed single and multi-copy strategies could be exploited.

Proposal For A Network INGRES

by

Michael Stonebraker

Electronics Research Laboratory
and
Lawrence Berkeley Laboratories
University of California, Berkeley, Ca.

ABSTRACT

This paper presents plans to extend the capabilities of an operational relational data base system, INGRES, so that it offers a powerful data access capability in a computer network. Besides proposing to hide all details of the net from the user, we also plan to allow portions of a single relation to be stored on several computers. Where portions of a relation are stored would also be invisible to the user. Also presented are examples of the utility of such a user "view" of a network. Research problems which must be solved to efficiently support this "view" on top of the operating system to operating system communication facility provided by, for example, the ARPANET are also discussed.

1. INTRODUCTION

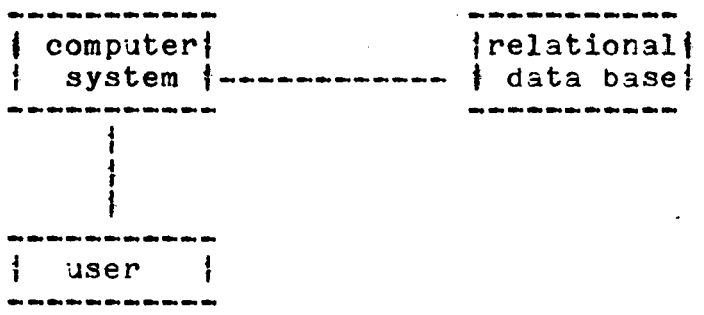
This paper discusses the problems inherent in extending the capabilities of an operational data base system, INGRES, [STON76, ZOOK76, HELD75], so that it offers a powerful data access capability in a computer network. These problems arise because of the "view" of a network that we suggest giving to an INGRES user. This particular "view" is explained in Section 2. Then in Section 3, a series of examples are presented which indicate the utility of the proposed "view". Section 4 discusses 3 research problems which must be explored in order to efficiently support our proposal.

2. THE INGRES USER'S VIEW OF A NETWORK

INGRES currently presents a user with a relational view of data and allows him to interact with his data in a high level non procedural data sublanguage, QUEL, plus assorted utility commands. All details of the storage organization and access paths to data

are hidden from a user and may be changed at will without impact on the user interface.

The current version of INGRES supports a relational data base on a single computer system, (which may be a PDP-11/40, 11/45 or 11/70 running the UNIX operating system [RITC74]) as noted in Figure 1.

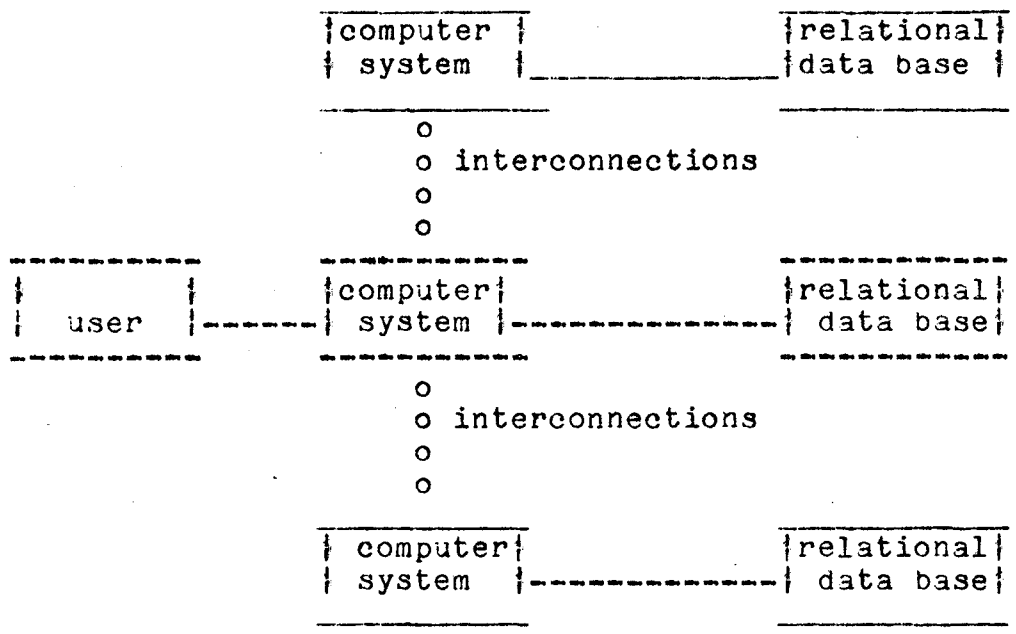


The Current INGRES User View

Figure 1

We propose to expand the capabilities of INGRES so that it can run in the environment of Figure 2 with the following capability:

> A user will not be able to tell the difference between Figure 1 and Figure 2.



A Network of Data Bases
Figure 2

Basically we propose to make the network transparent to the user. This approach has the following properties:

1) a user will view the network as if all data bases were merged and stored on a single machine

2) a user does not need to know where his data is stored. In fact a performance consideration will be where to physically store data and whether to store multiple copies.

3) portions of a single relation can be stored on more than one machine.

4) this strategy is in keeping with the INGRES philosophy of hiding storage details.

Note that this "view" differs dramatically from previous proposals for network data bases (eg [LEVI75, CASE72, WHIT70, CHU69,]) in the following way:

> Most of the work cited above is on optimization and all authors assume interactions are with a file at a single site or multiple copies of the same file at different sites. Very different issues arise when property 3 is

required as noted in Section 4.

Note that this approach differs noticeably from the view taken by the Data Computer project [MARI75]. There, all data is on a single machine (the data computer) and the user must be aware where his data is (i.e. on the data computer).

The implementation environment we propose is a collection of PDP-11's each running UNIX. Three UNIX systems at Berkeley are currently being interconnected and UNIX to UNIX protocol is being implemented by the UNIX designers. This setting will be used unless an alternative is available.

We wish to stress that the operating system protocol and architecture for the net (e.g. ring, packet switching, message switching) is only of concern as it affects performance.

3. JUSTIFICATION OF THE PROPOSED VIEW

It is our contention that the features discussed in Section 2 (especially capability 3) are appropriate ones to provide for a network of computer systems each of which has data bases of common interest. We illustate this contention with an example.

Suppose each military installation keeps on its computer system a relation concerning the inventory of parts at its facility. An example relation is shown in Figure 3 for one such installation. Suppose further that these machine are connected in a computer network.

	PART#	Quantity on hand	Location
PARTS	jeeps	5	installation X
	gasoline	10000	installation X
	pencils	300	installation X

A Sample Relation
Figure 3

The following interactions are desirable:

a) interactions with the local data base; e.g. lower the gasoline supply by 1000 gallons at installation X.

b) interaction with a single non local data base; e.g. does installation Y have more than Z gallons of gas

c) interactions concerning two data bases on two

machines; e.g. transfer 1000 gallons of gas from installation X to installation Y.

d) interactions concerning the whole network; e.g. find which installations have more than 1000 gallons of gas or find the total amount of gas at all installations.

e) interactions concerning a data base at an unknown location. e.g. find the quantity and location of part Z (wherever it is).

It should be clearly noted that presenting a user at any location with a view of the PARTS relations which is the union of the PARTS relation kept on each machine (property 3 of Section 2) facilitates interactions a)-e). Only by making the network transparent can interactions d) and e) be done conveniently.

4. RESEARCH AREAS

The problems which must be explored to support such a user view of a network are:

1) naming and storage of system catalogs
2) concurrency control
3) performance

We discuss each problem in turn.

1) Naming and Storage of System Catalogs

INGRES maintains a set of system catalogs describing the data base it manages at a particular site. A network of INGRES systems can, of course, each maintain catalogs for data at their own site. However, capabilities 2) and 3) of Section 2 require that a user who wishes to interogate a relation W not be confronted with several possible W's on individual systems with individual meanings. This requires naming of relations to be unique for the network. An efficient mechanism to ensure this must be found.

Moreover, one must face the problem of searching for W. It is very costly to interrogate each machine in the network for the presence of W. Alternately, keeping a "Master System Catalog" for the network on one machine creates a possible performance bottleneck (since every request must go through this machine) and a reliability problem (this machine cannot fail). Keeping multiple master catalogs entails both a performance penalty and a headache ensuring they are mutually consistent.

2) Concurrency

Capabilities 2) and 3) of Section 2 raise serious problems in a multiuser, multicomputer environment concerning concurrent update by users. This problem is illustrated by example.

Consider the PARTS relation discussed earlier and suppose that it is stored on several computers (say each installation has each row of PARTS concerning its own location). Lastly, suppose the following two concurrent updates are to be executed.

U1 decrease by 1 the quantity on hand of all parts less than 10 at all installations.

U2 change bolts from part 12 to part 6 at all installations.

It is easy to see that care must be exercised to ensure that either:

a) one is subtracted from the number of bolts at all installations (i.e. U2 logically precedes U1)

or

b) the number of bolts at all installations is not affected (i.e. U1 logically precedes U2)

Ensuring data base consistency under multiple concurrent update on a single computer system has recieved some study [STON74, GRAY75] and is a hard problem with perhaps high overhead. However, on a single system such updates in the worst case can be run sequentially and backout is possible. On a network both possiblilities are more remote.

3) Performance Considerations

In a network one has the option of storing data on more than one machine (subject to guaranteeing its consistency). One also can choose where in the network to process interactions. One can transfer statements in the data sublanguage around the network and process them remotely. Data (results) would only have to be transmitted if a user wished them printed. One can also assemble needed data on a single machine. Even more possibilities exist when a user wishes to simultaneously interact with two or more relations each of which is on more than one machine. Optimization of such interactions on a single machine has been studied in [WONG76, SMIT75, STON76b]. However, the robustness of such results to a network are questionable.

We conclude this section by illustrating two options for a specific example.

Consider the PARTS relation from Section 2 subject to the interaction.

> "Find the part numbers of parts with quantity on hand greater than 100 at 2 or more locations"

Option 1 — Assemble the entire PARTS relation on one machine then issue the above query

Option 2 — broadcast the following query to each machine

> "Find and return part numbers with quantity on hand greater than 100"

When all results are assembled issue one local query

> "Find the part numbers with a COUNT greater than 1"

This second option will lower communication costs by doing some processing at each site and appears desirable.

REFERENCES

CASE72 — Casey, R., "Allocation of Copies of a File in an Information Network", Proc 1972 SJCC, AFIPS Press, 1972.

CHU 69 — Chu, W., "Optimal File Allocation in a Multiple Computer System", IEEE Transactions on Computers, October, 1969.

GRAY75 — Gray, J.N.,Lorie, R.A., and Putzolu, G.R. "Granularity of Locks in a Shared Data Base", Proc. 1975 VLDB Conference, Framingham, Mass., Sept., 1975.

LEVI75 — Levin, K and Morgan, H., "Optimizing Distributed Data Bases - A Framework for Research" Proc 1975 National Computer Conference, AFIPS Press, 1975.

MARI75 Marill, T and Stern, D. "The Datacomputer- A Network Data Utility", Proc 1975 National Computer Conference, AFIPS Press, 1975.

RITC74 Ritchie, D.M. and Thompson, K. "The UNIX Time-Sharing System," CACM, Vol. 17, No. 3., March, 1974.

SMIT75 Smith, J. and Chang, P., "Efficient Execution of Queries in a Relational Algebra", CACM, October, 1975.

STON74 Stonebraker, M., "High Level Integrity Assurance in Relational Data Base Systems", University of California, Electronics Research Laboratory, Memo. ERL-M473, August, 1974.

STON76 Stonebraker, M. et al "The Design and Implementation of INGRES", ACM Transactions on Data Base Systems, Sept. 1976.

STON76b, Stonebraker, M. "A Comparison of Links and Secondary Indices in a Relational Data Base System", Proc. 1976 IEEE Conference on Software Engineering, San Francisco, Ca., October 1976.

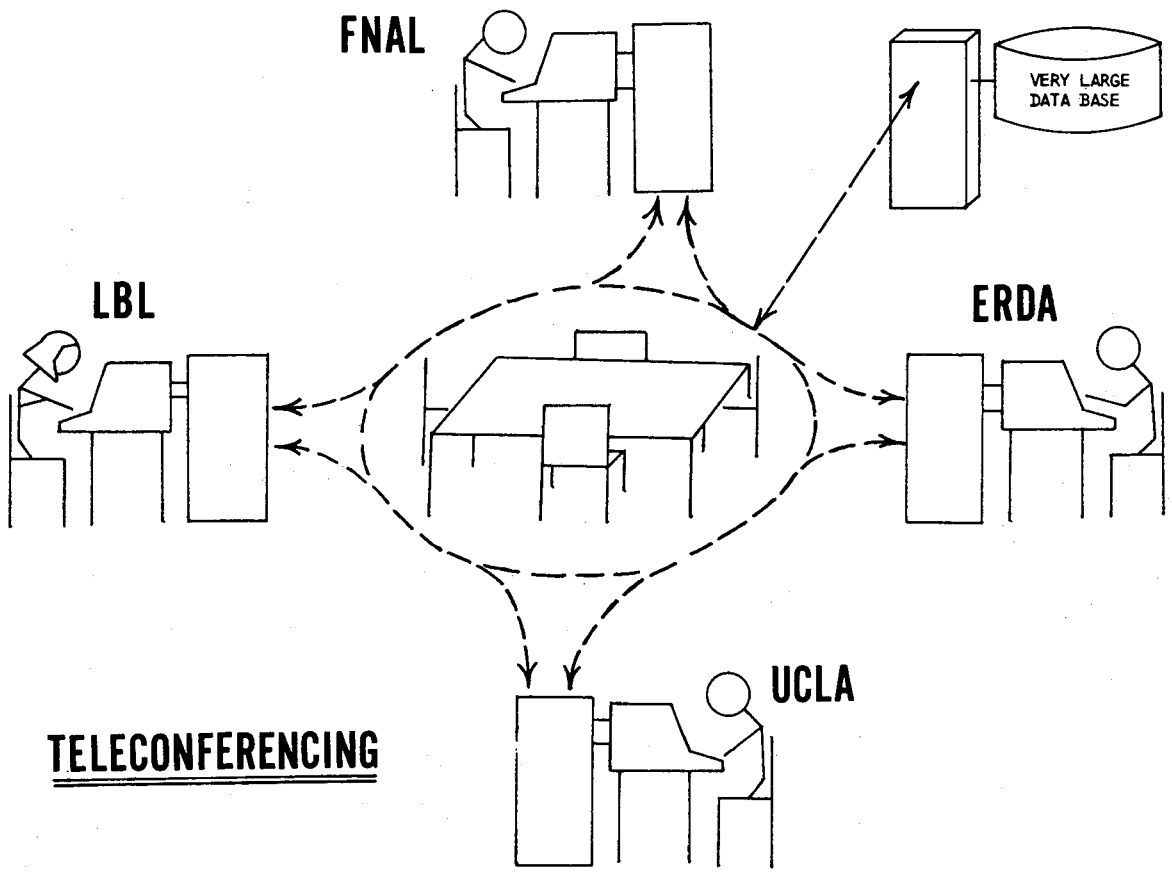
THOM76 Thompson, K. (private communication)

WHIT70 Whitney, V., "A Study of Optimal File Assignment and Communication Network Configuration", Ph.D dissertation, University of Michigan, 1970.

WONG76 Wong, E., and Youssefi, K., "Decomposition- A Strategy for Query Processing", ACM Transactions on Data Base Systems, Sept. 1976.

ZOOK76 Zook, W., Youseffi, K., Kreps, P., Held, G. and Ford, J., "INGRES- Reference Manual", University of California, Electronics Research Laboratory, Memo. No. ERL-M585, April, 1976.

V-A. TELECONFERENCING AND NETWORK MAIL



TELECONFERENCING

TDA Message System

Thomas Marill

Computer Corporation of America

Abstract

TDA is a computer based, terminal oriented system, allowing for the communication, preparation, editing, filing, and retrieval of messages. The system is based on the mailbox concept: messages are held in an "electronic mailbox" until "picked up". This communication process is asynchronous -- the sender sends, and the recipient receives, each message at a time convenient to himself -- and is therefore unlike telephone communications, in which both parties must be simultaneously present. Likewise, the process is location independent. The sender need not know the whereabouts of the recipient; the recipient can read his message at any location having a suitable (fixed or portable) terminal. The TDA system provides an integral system, allowing users to store messages in multiple files and to retrieve selectively from these files. A layman oriented text editor is provided as part of the service, allowing messages to be corrected or changed, and parts of messages to be included in others. Facilities for distribution lists and for multiple addressed messages are provided.

The system is implemented in a scalable architecture based on minicomputers (PDP 11). The smallest configuration uses a single CPU and disk, and can serve a user population of about 400. It can be scaled upward homogeneously to a multiple CPU/multiple disk system serving thousands of users.

Currently the system is operating at CCA and has been used for internal corporate communication for about six months.

Notes on the Development of Message Technology

T.H. Myer
D.W. Dodds

Bolt Beranek and Newman Inc.
Cambridge, Massachusetts
June 4, 1976

I. INTRODUCTION

This paper sketches the past, present, and possible near future of computer message systems. For purposes of discussion, we have excluded other forms of computer assisted communication (such as real time "linking" between computer terminals), and all types of non-computer communication. We and others have coined the phrase "message technology" to describe the engineering of computer message systems, and we shall use that term in this paper, even though its connotations may seem somewhat broader than the topic in hand.

For the most part, we shall be concerned with message systems accessible through the ARPA computer network (ARPANET for short), though there are message systems that operate over other networks, or no network at all.

Our perspective in what follows is the experience we have gained through work in this area at BBN, over the past five years or so. During the early part of this period we worked on an informal basis with relatively modest experimental programs. More recently, however, our work has intensified, and now continues under the name "Project Hermes"*.

In what follows, we will look briefly at the early development of message systems, and then take a somewhat more detailed look at their present state using the Hermes system, the main product of Project Hermes, as an example.

---

* Project Hermes is supported jointly by the Department of Defense Advanced Research Projects Agency, by the Naval Electronics Systems Command and by the U.S. Army Materiel Development and Readiness Command.

II. EARLY DEVELOPMENT

Message communication by computers and computer networks is a relatively new development, dating back to the spread of time shared computer systems in the 1960's. As far as we know, Ray Tomlinson of BBN was the first to experiment with message transmission via interlinked networks of computers.

Ray's first experimental program, SNDMSG, helped the user create and "send" messages of fixed format through an interactive dialogue. SNDMSG embodied a concept of message delivery based on many layers of underlying ARPANET technology. Each message was "packaged" as a file, moved across the network (or within a host computer) by standardized file transfer programs and appended to a special "inbox file" in the file directory of each recipient. Early message reading programs allowed the user to extract, print and delete the messages in his inbox. In the years following SNDMSG, standard network-wide formats and protocols were agreed on with the result that reasonable message service is now available between widely differing host computers on the ARPANET.

SNDMSG and the related programs that quickly followed introduced the ARPA research community to a communication medium with hitherto unknown properties and a geographic spread equal to that of the network itself. Network service was an immediate success. Message flow grew in volume to become the most visible (if not the heaviest) traffic component on the network. Use of the service has had a substantial impact on the organizations involved, stimulating dramatic shifts of dependence away from the traditional media (postal service, telephone).

A surprising aspect of the message service is the unplanned, unanticipated and unsupported nature of its birth and early growth. It just happened, and its early history has seemed more like the discovery of a natural phenomenon than the deliberate development of new technology. All that is changing, however.

We in the ARPA community (and no doubt many others outside it) have come to realize that we have in our hands something very big, and possibly very important. It is now plain to all of us that message service over computer networks has enormous potential for changing the way communication is done in all sectors of our society: military, civilian government, and private. As a consequence, the further development of this technology is now under deliberate and organized

attack.

Although we can't prove it, we suspect that computer message service owes its initial success to a combination of properties found together in no other communication medium - telephone, postal service, teletypewriter, etc. These properties include

Electronic speed.

Decoupling between sender and receiver
(no need for both to be "present").

The ability to apply computer processing to aid in message composition, reading, storage, and retrieval.

Geographic independence -- one can not only send messages from any "station" on the network, but access one's received messages as well.

III. THE PRESENT

Project Hermes took off from the point of departure represented by SNDMSG and the other early programs. Our work has concentrated on software support for the user at the sending and receiving ends of the message service. For the time being the underlying delivery service (mostly invisible to the user) has remained unchanged, although there are good reasons for building an improved delivery service in the future. Our belief is that the Hermes system is reasonably representative of the state of the art in production oriented message systems available on the ARPANET.

Hermes combines in a single program most of the tools needed to cope with messages, together with some capability for general office functions. The major Hermes modules are:

A Message Writer.
A Message Reader.
A Storage and Retrieval Subsystem.
Text Manipulation Tools.
A User Profile.

Before describing these major functional areas, a few words are in order on the human interface to Hermes. Since Hermes was designed for operation on Tenex systems, we adopted the conventions of the Tenex Executive language. Thus, Hermes is a command oriented system, with the same style of interaction found in the Tenex Executive. Commands (and many of their arguments) can be typed in full, abbreviated (down to

the minimum required for disambiguation) or extended (Hermes echoes the remainder of a disambiguated command). At any point in the entry of any command or argument, typing "?" will yield a tabulation of the legitimate choices available at that point.

The Message Writer.

Hermes generates messages conforming to the current ARPANET standard format through either of two techniques. The user may enter messages in a command driven mode that makes it possible to create message parts in any order, review, change or delete the parts already entered. The message writer provides a format function that "fills" and (optionally) justifies ragged text, and provides access to standard, Tenex supported text editors. Once complete, a message is handed over to the delivery service by a "send" function for transmission to users on the local or foreign host computers. A partially completed draft can be set aside prior to sending and shifted back into the message writer at a later time for final completion.

The message writer can also prompt the user for input of message parts through a "form" mechanism. Message forms determine what fields are to be included in a message, and the order in which their input is to be prompted. A form can also store fixed information, (such as addressee lists, file copy destinations, often used subjects) to be added to selected fields of each message created through that form. Hermes supplies various built-in-forms and also a specialized editor through which the user can create forms of his own.

The two input modes can be employed independently, in isolation from each other or used in tandem. Having completed a prompted input sequence, one can review and modify it through commands of the message writer. A prompted input form can be "activated" at any point in command mode.

The Message Reader.

Inbound messages are appended to a special "inbox" file in each user's directory. The message reader provides for parsing the contents of this file, extraction and output of the messages contained. Message sequences can be selected for output by specifying their ordinal numbers in the message file, by searching on the full or partial content of any field* , by date relative to (on, before, after) any of the date bearing message fields, or by various boolean combinations of these selectors. For example

>PRINT 1,3,7, From Smith, Subject Weather/After 5-2-76

would select messages 1,3,7, all those from Smith or about weather, and then narrow down the selection to those messages sent after the date indicated.

As well as message selection, output commands in the message reader provide format control, and choice of output destination. The same basic form mechanism that generates prompted input sequences is applied in an "inverted" mode to control output format. Destination, which can be a file or output device, is controlled by underlying mechanisms of the Tenex operating system. The message reader also provides for deleting messages from the inbox, status markings ("seen", "unseen", etc.) and other housekeeping functions.

Storage and Retrieval.

Hermes provides a two level storage system. Messages inthe user's inbox can be dispatched to other files in the user's workspace for sorting and classification at a gross level. Within any file, messages can be organized into named "folders" representing overlapping subsets of the messages contained. Hermes provides a specialized editor for inserting messages into a folder, sorting its contents (based on any of several message fields) etc. The messages to be added to a folder are specified through the same search/selection tools as available in the message reader.**

In order to retrieve messages from a file, the message reader is "slewed around", so to speak, and brought to bear on that file. All of its output and housekeeping functions then apply to the messages and folders within the file selected. One can include one or more folders in a sequence of messages to be output, or narrow one's selection bounds to just the messages within a single folder.

Text Manipulation.

All fields of current ARPANET messages contain text information, and consequently it is useful to have

---

* A simple substring match technique is applied to most fields.

** These tools are applied in a uniform fashion throughout Hermes, whenever messages need to be selected.

various mechanisms in a message system for shifting and manipulating text objects. Hermes provides a number of such tools.

An ordinary Tenex file can be copied into any field of a message. This provides an alternative to message forms for storing address lists, standardized subjects, etc. More importantly, it also makes it possible for messages to serve as carriers for preexisting text documents such as reports, data in ASCII format or even computer programs in their source state. We are clearly approaching an electronic parcel post.

Text can be moved out of messages as well as into them. Any field of a draft message can be copied out into a Tenex file. This allows the message writer to serve as the means for creating specialized data such as address lists, and also as a general document preparation tool.

Some rather powerful results can be achieved when it is made possible to dissect incoming messages and reuse their parts. Hermes contains the primitive functions to copy any field of a received message into the corresponding buffer of the message writer. These currently form the support for three user operations.

Through an "explode" function a received message can be copied in its entirety into the buffers of the message writer. The resulting information can be sent out as a new message, with or without changes, moved back into a message file, or it's parts separately transferred into other files.

The underlying primitives also support forward and reply operations in which selected fields of a received message are automatically mapped into an outgoing message.

User Profile.

We and other system builders have found that message processing is a sufficiently rich and complex activity that different users will have marked differences of opinion on how a system ought to behave. Within broad limits, the more one can cater to a user's tastes, the happier he'll be.

Hermes provides tailorability for the individual user through a series of mechanisms, whose state with respect to any given user is kept in his "profile".* These mechanisms include a series of "switches" that alter, one way or another, the behavior of many Hermes commands, default objects that allow the user to supply

standard choices for command arguments he chooses to omit, and various other information including the message forms supplied by Hermes or created by the user.

Human Factors Aspects.

We have faced some interesting human factors problems in building Hermes. Perhaps most challenging is the desire to build a system that could satisfy the advanced user without mystifying the novice. In our efforts to achieve such a system we have used a number of techniques including the defaulting of ommitted arguments, the provision of prompted as well as command driven message input, and the segregation of relatively advanced functions such as message form or folder creation to specialized "work spaces" that can be ignored entirely by the novice. We are considering additional techniques for the future, in particular the sequential enabling of command groups as the user gains in proficiency.

Another intriguing problem was the likelihood that many users (or would be users) would neither possess our hard copy manual nor the means to obtain such hard copy from computer files. Consequently, we are putting a substantial effort into on-line user aids usable with an ordinary computer terminal. These include the "?" feature mentioned previously, a more elaborate "describe" command that provides detailed information on many aspects of the Hermes, and a tree-structured conceptual introduction to the system. We have especially high hopes for the latter as an introduction to Hermes; it allows the user to explore the documentation in expanding layers, in much the same way as he is likely to learn the system itself.

IV. FUTURE POSSIBILITIES

In spite of its popularity and heavy use, there are shortcomings in todays message service from which one can infer its future development. Each of these represents one way in which we haven't yet fully exploited the underlying potential of computer and network technology.

---

* The user profile is implemented as an ordinary Tenex file, but this fact is generally of no concern to the user.

Automated Office Functions.

Today's message systems usually provide for communication in isolation from other related services. For example, development up to now has largely neglected the fact that communication is just one aspect of the automated office. Thus, as time passes we expect to see a joining of forces between message technology and the movement toward office automation.

In the Hermes project, we are pursuing this goal in the near term by broadening our file management system to handle general office records, as well as messages; by extending the message composer to provide general text manipulation tools; and by developing mechanisms that will allow Hermes to serve as a project management tool.

We can also forsee other extensions that will take longer to achieve. Current message systems can only handle text. We expect that future messages will serve as carriers for arbitrary data types including structured graphics, facsimile, and compressed speech.

Current message systems provide various functions to help the user, but any procedure not anticipated and built in by the system designers must be carried out by hand. We expect that future systems will include simple, high level programming languages to make it easy for application programmers or even users to create new functions.

Collaborative Message Processing.

Current message services provide communication between isolated senders and receivers. Even when users share the same computer it is awkward or impossible to achieve collaborative message authorship or collective handling of received messages.

As an example, I received a message from Jones that needs a prompt answer. Smith, a colleague, has also received this message, and may have answered it, but how can I find out? At present, it takes a rather tedious search of my "inbox" to find Smith's reply, if it exists. I would much prefer Smith to have linked his reply to Jones' message so that the one could have led me directly to the other.

One can think of many examples like this in message drafting and message reading. What they all suggest is a means for establishing hierarchical associations between messages and enabling one collaborator to

convey these associations to another.

One solution is to borrow from the technology of data management. The message system becomes a global data base. To create a message is to add a record to the data base. To "deliver" it is to make that record accessible to the recipients. Associated structures of messages are built by creating index structure over the data base. Access control mechanisms limit the distribution of messages, associations between messages, and other information.

We know of at least three message sysems (our own included) that are headed in this direction. In our view, it is a good near term solution but not a long term solution. It has two key limitations: First, we don't yet have the technology to effectively manage richly structured data bases when they spread across the bounds of more than one computer.

Second, even when we acquire that technology, there will be situations in which lengthy or unpredictable time delays will make the data base model inoperable. As one example, a military message network may suffer equipment failures and communication blackouts that impose random patterns of time delay across the system. A second example was suggested by Arthur C. Clarke in a recent talk at M.I.T. Clarke points out that as we move out into space, speed-of-light delays will sooner or later force an end to real time communication.

In either of these situations, it is hard to see how a distributed data base could be effectively maintained without chaotic asynchronization among its parts. We expect that it will be necessary to return to the old style of business in which messages are thought of as physically moving across space/time gaps. Under these conditions, collaborative message processing will have to be achieved by other means, which we regard as an important research area.

Message Automation.

Message technology has barely tapped the automation potential of computers. In all but a few cases, messages are created and dispatched by human users.

An inspection of most organizations will reveal stereotyped, message oriented operations that make prime targets for some form of automation. This is especially true if one broadens one's outlook to consider classes of message traffic beyond interpersonal communicaton on general topics. We have

described these classes of operation as "structured communication" and coined the phrase "transaction structures" to describe the software mechanisms that will support their automation. Here are some examples of structured communication:

While on vacation, my incoming messages are answered automatically. Each sender is notified of my absence and expected return date.

Requests for Social Security reports are handled by messages dispatched to and answered by an automated reporting system.

Spare parts orders are conveyed as messages to an order filling/inventory control system.

Complex, but well structured paperwork flow patterns, such as the processing of time sheets or purchase requisitions, are supported by machine directed message transmission.

As we see it, developing the technology for message automation depends on a systhesis of tools - some existing, some yet to be created:

We will need a message delivery service that can support interprocess as well as interpersonal communication. (A more powerful delivery service is also needed to support the other development areas discussed previously.)

Message automation will exploit the technology of "daemon" or "persistent" processes, especially for fielding and responding to messages.

It will be necessary to develop a specialized programming language and a run time support system to define and operate transaction structures. These developments will draw on such research areas as concurrent processing theory, and distributed support systems (for example BBN's Resource Sharing Executive).

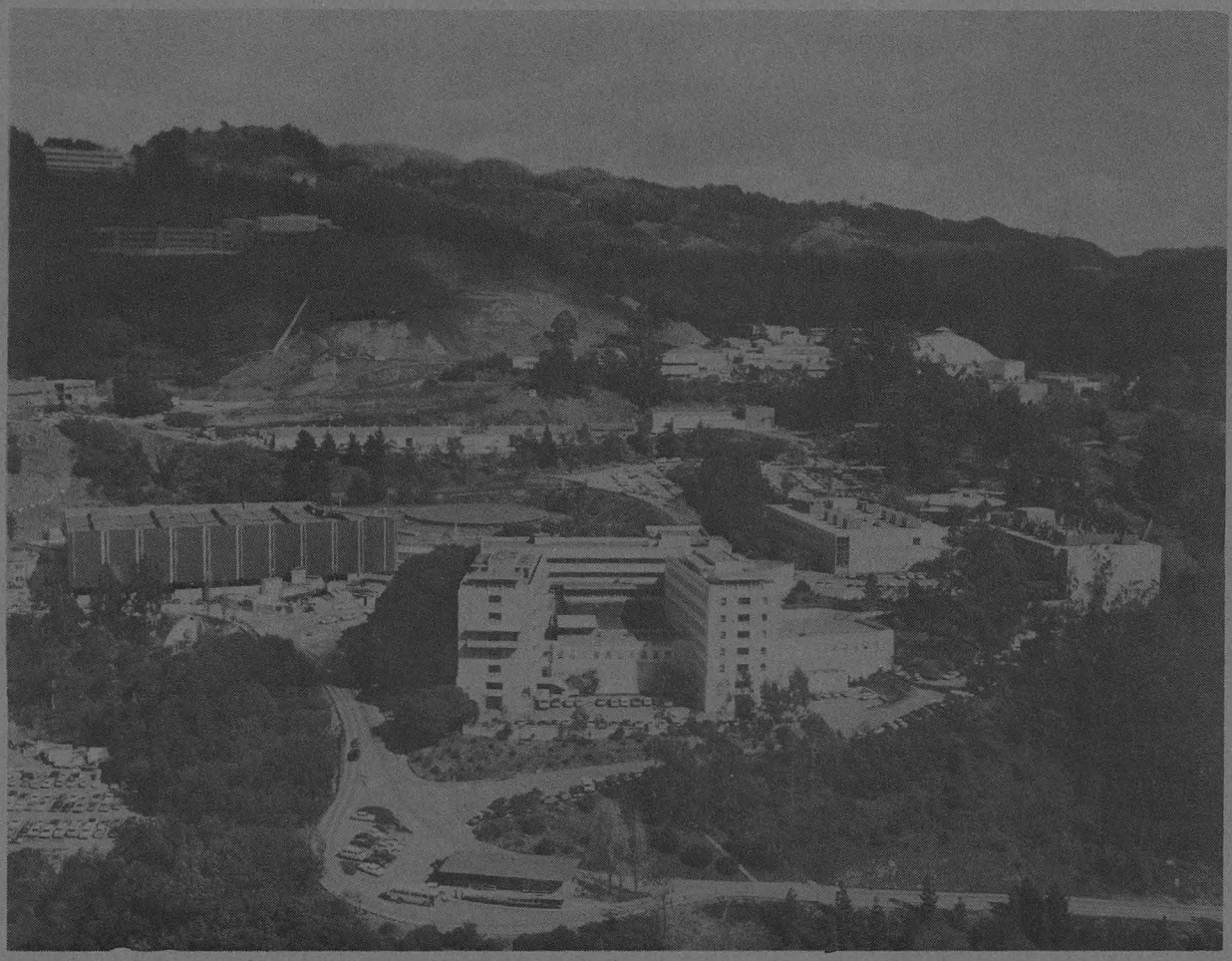
V. CONCLUSION

We have tried to sketch the current technology of message systems and some important areas for further development. This topic seems large to us since we are involved in it full time. However, if one moves back and looks at the full scope of the communication revolution now underway, one realizes that message technology is but a small "piece of the action". Other important questions include: Where are message systems

going in terms of usage? Will they replace first class (and bulk rate) mail? How will they be commercially exploited and regulated? How do message systems fit with the notion of the "information pipeline" that delivers two way television, newspapers, and all other information into the home or office of the future?

ACKNOWLEDGEMENTS

This paper describes the work and thoughts of a collaborating group at BBN. The two authors are serving as reporters for that group, but the ideas and results accomplished belong to the group as a whole. Austin Henderson, in particular has played a key technical role in Project Hermes. Other contributors include Ron Brachman, Jerry Burchfiel, Carl Feehrer, Charlotte Mooers, Dick pew,, and Frank Ulmer.



V-B. DATA TRANSLATION AND EXCHANGE STANDARDS

AN EXTENSION OF THE ANSI Z39.2 STANDARD TO GENERAL INFORMATION EXCHANGE

A. A. Brooks

Computer Sciences Division
at Oak Ridge National Laboratory
Union Carbide Corporation, Nuclear Division*

INTRODUCTION

The Intralaboratory Working Group for Data Exchanger has as its objective the exchange between ERDA laboratories of data related to the regional studies program. One of the several problems addressed is the computer-oriented format for the actual exchange process. Several approaches such as a common information system, a common simple data base system, variable FORTRAN formats, sender-supplied read routines and various recording media were considered. Given differences in computer hardware, computer software and other practical considerations, it was agreed that magnetic tape offered the best compromise for an exchange medium and that a mutually acceptable recording format interfaceable to a variety of computer systems were the appropriate near term solution. Consideration of the several current exchange standards (ANSI Z39.2 - 1971, UNISIST WC.74/WS/20, IAEA-INIS-9 (Rev. 2), TID 4581 (R2), MARC - MIF; all of the ISO 2709 type) revealed a specificity to a limited content, yet represented a wide spread commonality steming from ISO 2709. Therefore it was decided to continue in this direction and produce a standard modeled upon ANSI Z39.2 but extended to broad classes of both textual and numerical information defined not on the subject content but rather on the type and structure of the information. At the same time, a means of identifying the subject content of the information was to be included as a part of the standard.

GENERAL DESCRIPTION

The aim of this standard is to establish machine independent formats for the transfer of a wide variety of

*Prime contractor for the U.S. Energy Research and Development Administration.

**The IWGDE has two representatives from the Regional Studies programs at BNL, LLL, LBL, PNL, ANL and ORNL.

hierarchically structured alphanumeric information on magnetic tape. The standard utilizes several of the ANSI standards and generalizes the ANSI Z39.2-1971 Standard for Bibliographic Interchange on Magnetic Tape as a vehicle for the interchange of a wider variety of information. The standard encompasses a model for structured fields which permits a definition of a different structure for each field, including an open ended set of yet-to-be-defined data types and structures. The standard also anticipates the use of existing and future systems of the ISO 2709 type and provides the means to identify such systems and to include them as subsystems within this standard. It is inherent in the model for this standard that each logical record represent all of the information which is associated with an individual whose identity is persistent over the life of the data base, and that this information be storable as defined data elements in hierarchical storage structures. Each logical data record has a unique identifier field which has a prescribed tag (...001) and the field is located in a unique position. The higher levels of the structure are accessed through a system of tags and pointers. The structure can accept a planar relation and non-cyclical identified directed subgraphs of chained linked data bases. If required, several planar relations can be collapsed on a common principal key to produce their hierarchical equivalent. Cyclic directed graphs expressing structure among individuals can be represented by including a successor (predecessor) field to contain identifier field entries which can be used to restore the original directed graphs.

The rationale of the standard is (1) to define a set of content-free standards for those portions of the standard which are solely computer-oriented, (2) to adopt a content-free version of ANSI Z39.2-1971 for the logical record structure, (3) to define a model of defined hierarchical structures as the storage structure for each field, (4) to define the means of codifying the type of data element stored in each field, and (5) to provide the means of transmitting a field by field description of the data base on the same magnetic tape.

The modus operandi to produce this standard has been to use applicable portions of ANSI Z39.2-1971 verbatim deleting the unwanted restrictions and conventions for bibliographic information and then augmenting the remainder to produce the desired standard such that ANSI Z39.2-1971 and others of the ISO 2709 family can be a subset of this standard. This has resulted at times in unused control characters which are retained to maintain similarity to other systems.

An essential element of this standard is a data base description file (DDF) which permits the designation of the

defined subsystem which is to be invoked and if required, on a field by field basis, the designation of a specific occurrence of the many possible structural forms permitted.

The terminology of ANSI Z39.2-1971 has been retained wherever possible even though a more appropriate term for a general case might have been devised. In these cases the definition of the term has been augmented to reflect its broader meaning.

It is anticipated that this standard will result in automated, hardware-dependent software which will process all or part of the standard. It also assumes control card override or replacement of the data description file by the user when a data base requires said practice.

The following is a synopsis of the major features of the standards. The current draft of the standard is available on microfiche for those who have an interest in the detailed description.

SPECIFIC FOUNDATION STANDARDS

The following standards form a foundation for this standard in their respective areas:

1. Magnetic Tape Standards - ANSI X3.22 - 1967.

2. Physical Block Standard - Type S, ANSI X3.27 - 1969. Proposed Revision X3L5/419T (6-19-74).

3. Character Set & Codes - ANSI X3.4 - 1968.

STRUCTURE OF LOGICAL RECORDS

The structure of the logical records is identical to the ANSI Z39.2 - 1971 as shown in Fig. 1-4 with some exception in the use of control characters:

Leader-Byte 5 (DDF only)	-	subsystem control to designate the use of an existing standard or subsystem
Byte 6	-	character set control (if non ANSI)
Byte 11	-	field control count - used in description of data element structure and content type
Entry Map Byte 3	-	length of tag field

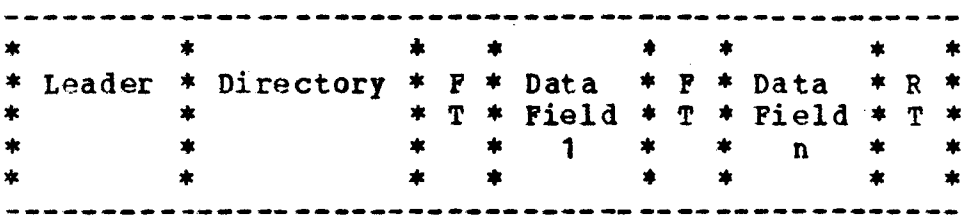
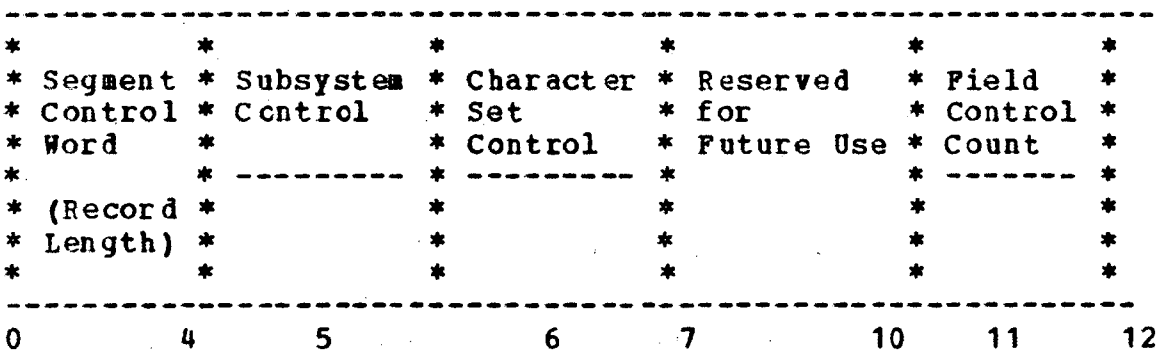


Fig. 1. Schematic Representation of the Logical Record of the Interchange Format.



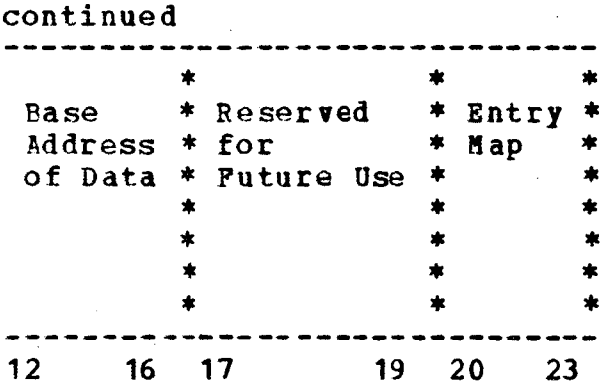
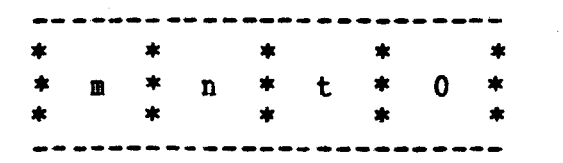


Fig. 2. Schematic Representation of the Leader (DDF contents above line in Bytes 5, 6, and 11.)



m = length (in characters) of the "length of field" portion of each entry in the directory
n = length (in characters) of the "starting character position" portion of each entry in the directory
t = length (in characters) of the "tag" portion of each entry in the directory
0 = undefined; reserved for future use

Fig. 3. Entry Map.

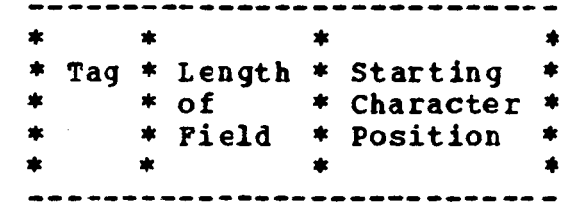


Fig. 4. Structure of Each Entry in the Directory

DATA DESCRIPTION FILE

The data description file is a single record in the previously described formats and contains controls on a field-by-field basis for the resolution of its own and its corresponding data record structure. Its data fields contain not data but text which is a description of the data in a corresponding structure.

DATA FILE DATA ELEMENTS

The data elements under consideration for the data file (DF) are given below:

Controls E/0: (Leader-Byte 5/Byte 6) Unstructured Text

Controls E/8:

Controls Data Type	Controls Auxiliary	No. of Delimiters	Data File Content for Subfield
10	00	0,1	Mixed Numeric Array (a)
10	01	3	Mixed Numeric Array/ Cartesian Labels (e)
11	00	0,1	Integer Array (a)
11	01	3	Integer Numeric Array/ Cartesian Labels (e)
15	00	2	Titled Numeric (b)
20	00	0,1,2	Delimited Text (c)
20	01	3	Delimited Text/ Cartesian Labels (e)
25	00	2	Titled Text (b)
30	00	1	Defined Hierarchies (d)

(a) The lack of a delimiter will indicate the presence of a single number. A data field representing an n-dimensional array shall consist of a delimited string containing:

1) the integer n,
2) a sequence of n integers giving the range of each dimension, and
3) a delimited sequence of data elements representing the content of the array in order of most rapidly varying first indices and in the usual I, E, D, and F FORTRAN formats.

(b) Titled information shall consist of the delimited string

{'tttt' @(1) 'vvv...' @(2) }...

where 'tttt' is the title of an attribute whose current value is 'vvvv'. '@(1)' and '@(2)' are the specified delimiters. The value of 'vvvv' shall be numbers for Type 15 and text for Type 25.

(c) Delimited text is limited to a two level hierarchy. Each element may be named or a single name may exist for the entire set of information. The data field of a delimited hierarchy takes the form

@(1)A(1)@(2)B(1,1)@(2)B(1,2)@(1)A(2)@(2)B(2,1)...

where "@(1)" and "@(2)" are delimiters in descending order, the A's and B's are titles in the DDF and the B's are current values in the information string in the DF and where the postscripts (i,j) have been added for the purposes of this example to denote the positioning of the data element in the hierarchy.

(d) The data field of a defined hierarchy takes the form

(i,j,k,...=xx...) "value"

where the fixed delimiters are "(", ",", "=", and ")"; i, j, and k are positional indices; xx... is a type declaration and "value" is the data element title in the DDF and the information string in DF. The defined types are

SI - integer
SE - decimal number
LI - list of integers
LE - list of decimal numbers
ST - single string of characters
LT - multiple strings of characters

Where multiple data elements within a hierarchical element are specified (i.e., LX), they shall be separated by the specified variable delimiter. Multiple hierarchical elements shall be separated by the specified delimiter. Where no type is specified, i.e., (i,j,k...), the default type shall be ST.

e) The Cartesian label variant of 10, 11, and 20 shall contain in the DDF a Cartesian label formed from the enumerated label vectors for the "columns" of a multidimensioned array, i.e., (a(1)@(2)...@(2)a(n))@(1)(b(1)@(2)...@(2)b(m))@(1)... where @(1) is the first delimiter of the data element

control, @(2) is the second delimiter of the data element control, a and b are label vectors of "column" identifiers, a(i) and b(i). The DF shall contain a list of data fields delimited by the third delimiter and in the same order as the expanded Cartesian label of the DDF. An optional "table" title may be designated by a pre- and post-delimited (@(3)) character string prior to the Cartesian label in the DDF.

STATUS OF STANDARD

The components of the standard are subject to two approvals; one for consideration and a second for adoption. The current (5-5-76) is given below, and the entire standard is subject to a "literary" edit before final publication.

I. Approval for implementation:
Sections 1-5, except 3.4.2, and appropriate appendices, except for examples.

II. Approval for final consideration
Section 3.4.2 - Data Elements and Controls

III. Submitted to work group
Section 3.4.2 - Extension to Cartesian Lables - A. A. Brooks, ORNL

IV. In preparation
a) Cartesian Product Augmentaticn of Extended Tables - J. Nardi, J. Heller
b) New Hierarchical Nomenclature - D. Richard, LBL

A MACHINE INTERPRETABLE DESIGN FOR PHYSICAL AND LOGICAL DESCRIPTION OF SEQUENTIALLY ARCHIVED DATA

J. Heller, J. Nardi

February, 1976

Informal Report

Abstract

This paper describes a method for the storage of hierarchical data on magnetic tape with its associated physical and logical description. It treats both data comprised of variable length records with variable length and number of fields and data comprised of fixed records with fixed fields.

The following notation is used throughout this discussion

⌀	blank
|	or
::=	"is defined as"
⟨⟩	Enclose required user supplied values
[ ]	Enclose optional components
{ }	Enclose values or components, one of which must be specified.

We suggest here a data exchange standard implemented under the general ANSI standards for magnetic tape. In this discussion we rule out all considerations of networks and deal only with hierarchies.

We introduce first the concept of a 4-tuple by which all data can be described with respect to both content and structure. A 4-tuple

f ::= ⟨ tag, value, partition, up-pointer⟩

where

tag	is an assigned name of an element of the data set.
value	is the data associated with the tag
partition	is the node name at which the tag occurs
up-pointer	is the parent of the node given in partition.

Consider now the description of hierarchical data. First define the following headings and delimiters

⟨tag del⟩::= ;;

⟨field del⟩::= ==

⟨record del⟩::= 0==

⟨node head⟩::= ++

⟨up-pointer head⟩::= ++

⟨up-pointer del⟩::= ++

Then we define

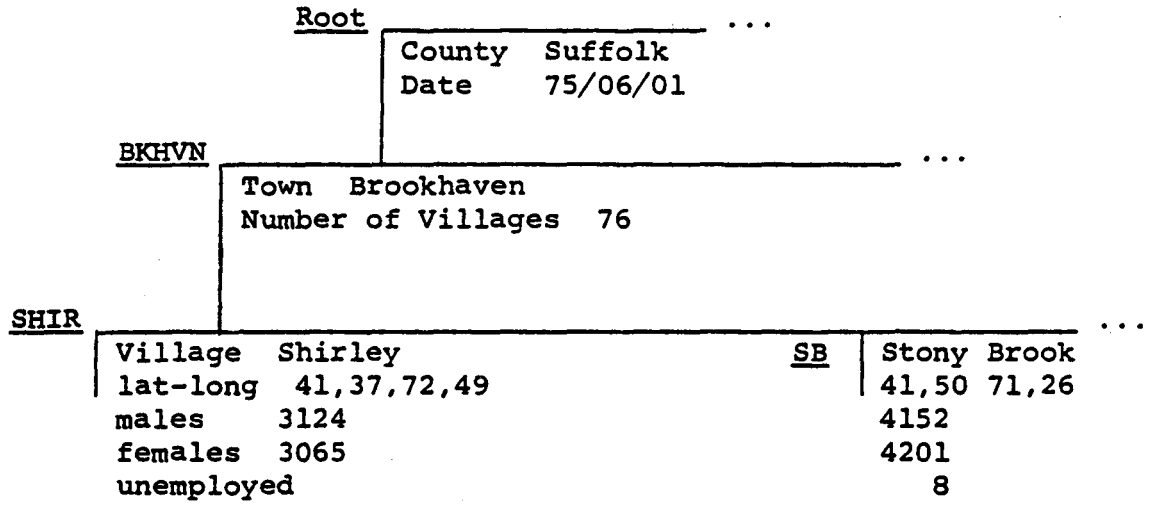
⟨stored field⟩::= ++⟨node name⟩++{⟨up-pointer⟩|Ø}++

{⟨tag⟩;;⟨value⟩==}

And

⟨stored record⟩::= {⟨stored field⟩...}0==

As an example consider the following hierarchical record.



where the underlined names are node names with their associated tags and values.

The stored record would then be represented by the following continuous character string.

1) ++ROOT++b++COUNTY;;SUFFOLK==DATE;;75/06/01==

2) ++BKHVN++ROOT++TOWN;;BROOKHAVEN==NUMBERbOFbVILLAGES;;76==

3) ++SHIR++BKHVN++VILLAGE;;SHIRLEY==LAT-LONG;;41 37b72,49==MALES;;3124==FEMALES;;3065==

4) ++SB++BKHVN++VILLAGE;;STONYbBROOK==LAT-LONG;;41,50b 71,26==MALES;;4152==FEMALES;;4201==UNEMPLOYED;;8==0==

(The numbers to the left are for illustration only to indicate the stored fields for each node name.) Note that the up-pointer for ROOT is indicated by a blank. Note also that although the stored fields 1, 2, 3 and 4 occur in their natural order it is not a necessary condition for reconstructing the hierarchical relationships. A computer program may interpret the above character string if we define the following local value given by

⟨local value⟩::=⟨tag name⟩;;⟨data type⟩[,⟨units[(qualifier)]⟩]==

where

tag name::=⟨node name⟩

data type::=⟨{CHAR|INT|FPT|...}⟩

unit;;=⟨unit of measurement(qualifier)⟩

For the previous example we would then have

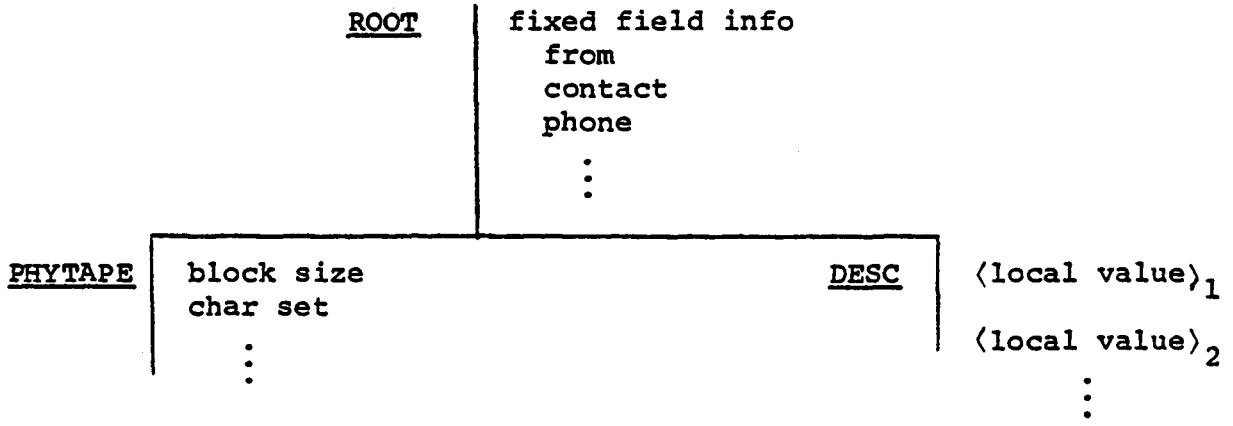
COUNTY;;CHAR==DATE;;CHAR==TOWN;;CHAR==

NUMBER␣OF␣VILLAGES;;INT==VILLAGE;;CHAR==

LAT-LONG;;CHAR,DEG-MIN==MALES;;INT==

FEMALES;;INT==UNEMPLOYED;;INT,PERCENT(MALES)==

The local value character string would precede the stored records and be embedded in a record we call the Physical Logical Description (PLD). The PLD is itself a hierarchical structure.



The PLD is then a record which is constructed using the 4-tuple concept except for the first fixed field which is defined as follows:

$\langle len_f \rangle \langle len_1 \rangle \langle \{local | global\} \rangle \langle len_2 \rangle \langle tag\ del \rangle \langle len_3 \rangle \langle field\ del \rangle \ldots$

where

len_f - is a four character field containing the total numbers of remaining characters in the fixed field.

local - indicates the data blocks contain the 4-tuple description.

global - indicates that the PLD contains a generalized 4-tuple description (to be discussed shortly).

len_i - is a two character field containing the number of characters to follow for a given description of any agreed upon parameter in the field for some agreed upon order.

The PLD for our example would then appear as follows:

```
003205LOCAL02;;02==030==02++02++02++
++ROOT++Ø++FROM;;CENSUSØBUREAU==
CONTACT;;JOHNØSMITH==PHONE;;(202)123-4567==
++PHYTAPE++ROOT++BLOCKØSIZE;;1024==
CHARØSET;;ASCII==++DESC++ROOT++
COUNTY;;CHAR== ... UNEMPLOYED;;
INT,PERCENT(MALES)==0==
```

where the ellipsis indicates the remaining part of the local value description given on page 4. The description described above is well suited to data comprised of variable length records with variable length and number of fields.

Consider now the following data

1.13281

0.52825

2.37893

A recording of these numeric values on tape would be nothing more than a bit string until some meaning were assigned to each of the rows.

Thus, suppose that the data actually represented average precipitation, average temperature and average humidity for each of three cities - New York, Denver and Miami - for the month of January for the year 1970. We could now pictorialize the data as

YEAR	MONTH	CITY	AVG. PRECIP.	AVG. TEMP.	AVG. HUM.
1970	Jan.	New York	1.1	32	81
1970	Jan.	Denver	0.5	28	25
1970	Jan.	Miami	2.3	78	93

Obviously, we may have many years for all the months for many cities.

Note that data of this type is comprised of fixed records with fixed fields.

We now record on tape the continuous bit string representing the above numeric values and suggest a method for globally describing the meaning of that bit string.

We describe the components of the above pictorialization as follows:

EXTERNAL DOMAINS			RECORDING DOMAINS		
YEAR	MONTH	CITY	AVG. PRECIP.	AVG. TEMP.	AVG. HUM.
EXTERNAL VALUES			RECORDED VALUES		

where a domain is equivalent to a tag.

Then

⟨external field⟩::= ⟨external domain⟩⟨external value⟩

⟨recorded field⟩::= ⟨recording domain⟩⟨recorded value⟩

and

⟨external record⟩::= min(1){⟨external field⟩}

⟨recorded record⟩::= min(1){⟨recorded field⟩}

Note that the recording domain is ordered. The order of the recorded records implies a preassigned ordering of the external records and is arbitrary subject to the following constraint: all values for an external field for a given external domain must be exhausted in a predefined order before the first (or any subsequent) in that predefined order is repeated again for a given external record. For example, suppose we pre-define the order of months to be JAN,FEB,...,DEC. Then once an ordering of recorded records for the external records ⟨1970,JAN,NEW YORK⟩, ⟨1970,FEB,NEW YORK⟩, ... has begun it must continue until the recorded record for ⟨1970,DEC,NEW YORK⟩ is completed.

Suppose now we consider an external domain a vector. We may then write

month = (jan,feb,...,dec)

In general, say we have the vectors

$$m = (m_1, m_2, \ldots, m_m)$$

$$c = (c_1, c_2, \ldots, c_n)$$

$$r = (r_1, r_2, \ldots, r_p)$$

Define the following operations.[1]

$$m \cdot X(c) = (mc_1, \ldots, mc_n)$$

$$= ((m_1, \ldots, m_m)c_1, \ldots, (m_1, \ldots, m_m)c_n)$$

$$= (m_1c_1, \ldots, m_mc_1, \ldots, m_1c_n, \ldots, m_mc_n)$$

and say m is left distributed over c.

$$mX \cdot (c) = (m_1c, \ldots, m_mc)$$

$$= (m_1(c_1, \ldots, c_n), \ldots, m_m(c_1, \ldots, c_n))$$

$$= (m_1c_1, \ldots, m_1c_n, \ldots, m_mc_1, \ldots, m_mc_n)$$

and similarly say c is right distributed over m. Note that

$$m \cdot X(c) = cX \cdot (m)$$

$$mX \cdot (c) = c \cdot X(m)$$

if the elements of m and c commute, i.e., if the domains of an external domain are interchangeable.

A vector z, the concatenation of two vectors, is defined as

$$z = (c \dot{+} r) = (c_1, c_2, \ldots, c_n, r_1, r_2, \ldots r_p)$$

---

(1) The notation $\cdot X, X \cdot$ and $\dot{+}$ is used in various texts dealing with the theory of matrices. We use it here for our own convenience.

Then

$$mX\cdot z = (m_1 z, m_2 z, \ldots, m_m z)$$
$$= (m_1(c \dot{+} r), \ldots, m_m(c \dot{+} r))$$
$$= (m_1(c_1, \ldots, c_n, r_1, \ldots, r_p), \ldots,$$
$$m_m(c_1, \ldots, c_n, r_1, \ldots, r_p))$$
$$= (m_1 c_1, \ldots, m_1 r_p, \ldots m_m c_1, \ldots, m_m r_p)$$

$m\cdot Xz$ is undefined.

If for any three vectors m, c and r we have

$$mX\cdot(c) \dot{+} mX\cdot(r)$$

we say a disjoint space exists for m over c and r and define

$$mX\cdot(c;r) = mX\cdot(c) \dot{+} mX\cdot(r)$$

A similar definition is made for

$$m\cdot X(c;r) = m\cdot X(c) \dot{+} m\cdot X(r)$$

Finally the recording domain, ρ,

$$\langle \rho \rangle ::= \langle \text{domain} \rangle \,|\, \langle \text{order} \rangle \,|\, \langle \text{length} \rangle \,|\, \langle \text{data type} \rangle \,|$$
$$[\langle \text{units}[(\text{qualifier})] \rangle] \,|, \ldots$$

For example, the recording domain for the above data would be

AVGØPRECIP/1/3/FPT/INCHES/,AVGØTEMP/2/2/
INT/DEGREES(F)/,AVGØHUM/3/2/INT//

By way of example we illustrate how a global description for fixed record, fixed field data is accomplished. Suppose we have two years, 1970 and 1971, monthly city data as shown above. In addition we also have for the same years, monthly regional data,

say for four regions called NW,SW,NE, and SE. The recording domain, ρ_2, for the regional data is

ρ_2 = TOTØPRECIP/1/4/FPT/INCHES/,MEANØ
TEMP/2/3/FPT/DEGREES(F)/

Pictorially represented, the regional data might be shown as

YEAR	MONTH	REGION	TOT PRECIP	MEAN TEMP
1970	Jan.	NW	08.3	38.2
1970	Jan.	SW	02.1	63.4
1970	Jan.	NE	15.8	47.8
1970	Jan.	SE	22.4	65.9
⋮	⋮	⋮	⋮	⋮

We shall denote the recording domain for the city data as ρ_1.

Suppose now we wish to write the recorded values such that they conform to the following ordering of external records

1970 JAN NEW YORK
DENVER
MIAMI
NW
SW
NE
SE
.
.
.

	DEC	NEW YORK
		DENVER
		MIAMI
		NW
		SW
		NE
		SE
1971	JAN	NEW YORK
	⋮	
	DEC	SE

The global description representing this ordering would be

$$\text{Year X}\cdot(\text{MONTH X}\cdot(\text{CITY}*\rho_1 \dot{+} \text{REGION}*\rho_2))$$

where the asterisk indicates that the recording domains ρ_1 and ρ_2 are to be applied to the recorded values for the external records indicated.

Denoting the values of the domains YEAR, MONTH, CITY and REGION by subscripted y, m, c and r respectively, an expansion of the global description yields the following ordering of external records

$$(y_1(m_1(c_1,c_2,c_3,r_1,r_2,r_3,r_4),\ldots,m_{12}(c_1,\ldots,r_4)),$$

$$y_2(m_1(c_1,\ldots,r_4),\ldots,m_{12}(c_1,\ldots,r_4)))$$

$$= (y_1m_1c_1, y_1m_1c_2, \ldots y_1m_1r_4, \ldots, y_1m_{12}c_1, \ldots y_1m_{12}r_4,$$

$$y_2m_1c_1, \ldots, y_2m_1r_4)$$

or

1970	JAN	NEW YORK
1970	JAN	DENVER
	⋮	
1970	JAN	SE
	⋮	
1970	DEC	NEW YORK
	⋮	
1970	DEC	SE
1971	JAN	NEW YORK
	⋮	
1971	DEC	SE

Each time a city is encountered a ρ_1 would be associated with it and likewise a ρ_2 each time a region was encountered. As another example, suppose the global description were

$$\begin{aligned}
&\text{YEAR X}\cdot(\text{MONTH X}\cdot(\text{CITY}*\rho_1;\text{REGION}*\rho_2))\\
=\ &\text{YEAR X}\cdot(\text{MONTH X}\cdot(\text{CITY}*\rho_1)\dot{+}\text{MONTH X}\cdot(\text{REGION}*\rho_2))\\
=\ &(y_1(m_1(c_1,c_2,c_3),\ldots,m_{12}(c_1,c_2,c_3),m_1(r_1,r_2,r_3,r_4),\\
&\ldots,\ m_{12}(r_1,r_2,r_3,r_4)),y_2(m_1(c_1,c_2,c_3),\\
&\ldots,\ m_{12}(r_1,r_2,r_3,r_4)))
\end{aligned}$$

which indicates that the recorded values occur for the following indicated external record ordering.

1970	JAN	NEW YORK
		DENVER
		MIAMI
	⋮	
	DEC	NEW YORK
		DENVER
		MIAMI
	JAN	NW
		SW
		NE
		SE
	⋮	
	DEC	NW
		SW
		NE
		SE
1971	JAN	NEW YORK
	⋮	
	DEC	SE

As a final example consider

YEAR X·(MONTH·X(CITY*ρ_1);MONTH X·(REGION*ρ_2))

= YEAR X·(MONTH·X(CITY*ρ_1))÷YEAR X·(MONTH X·(REGION*ρ_2))

= ($y_1(mc_1,mc_2,mc_3)$, $y_2(mc_1,mc_2,mc_3)$,

$y_1(mr_1,\ldots mr_4)$, $y_2(mr_1,\ldots,mr_4)$)

$$= (y_1((m_1,\ldots,m_{12})c_1,\ldots,(m_1,\ldots,m_{12})c_3),$$

$$y_2((m_1,\ldots,m_{12})c_1,\ldots,(m_1,\ldots,m_{12})c_3),$$

$$y_1((m_1,\ldots,m_{12})r_1,\ldots,(m_1,\ldots,m_{12})r_4),$$

$$y_2((m_1,\ldots,m_{12})r_1,\ldots,(m_1,\ldots,m_{12})r_4))$$

which represents the following ordering of external records.

1970	JAN	NEW YORK
	⋮	⋮
	DEC	NEW YORK
	JAN	DENVER
	⋮	⋮
	DEC	DENVER
	JAN	MIAMI
	⋮	⋮
	DEC	MIAMI
1971	JAN	NEW YORK
	⋮	
	DEC	MIAMI
1970	JAN	NW
	⋮	⋮
	DEC	NW
	⋮	⋮
1971	DEC	SE

If we impart the interpretation ↓ to X· and ·X and ⍋ to ⍒ and ; then we see that any hierarchical data structure may be described.

The PLD for globally described data (example 1) would then be written as follows:

```
003306GLOBAL02;;02==030==02++02++02++
++ROOT++Ø++FROM;;BNL==CONTACT;;
JOHNØDOE==PHONE;;(516)345-4122==
++PHYTAPE++ROOT++BLOCKØSIZE;;1024==
CHARØSET;;ASCII==++DESC++ROOT++
YEAR;;1970,1971==MONTH;;JAN,FEB,MAR,
APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,DEC==
==CITY;;NEWØYORK,DENVER,MIAMI==
REGION;;NW,SW,NE,SE==*RD1(2)
;;AVGØPRECIP/1/3/FPT/
INCHES/,AVGØTEMP/2/2/INT/DEGREES
(F)/,AVGØHUM/3/2/INT//==*
RD2;;TOTØPRECIP
/1/4/FPT/INCHES/,MEANØTEMP
/2/3/FPT/DEGREES(F)/==
*DESC;;YEAR X·(MONTH X·(CITY
*RD1⍒REGION*RD2))==0==
```

Note that it is entirely possible for a recording domain to be assigned to more than one external record or for an external record to possess more than one recording domain.

---

(2) We recommend that *RDi,i=1,2,... be a standard designation for a recording domain used in the *DESC field.

One obvious advantage of the above method is the ease with which new recorded values may be concatenated to the existing tape. For example, suppose the data for year 1972 is obtained. It would only be necessary to add the external value 1972 to the external domain YEAR in the PLD and add the recorded values to the tape in the proper ordering.

Current investigation is underway to develop the foregoing concepts in a set theoretic formalism. Current efforts also involve developing a parsing algorithm for assigning the proper external record to each accessed recorded record.

On the Importance of Common Standards for Logical Structures, Data Formats, and Query Languages

by

Arie Shoshani
System Development Corporation
Santa Monica, California

Abstract

In this paper, we describe the considerations which led us to the conclusion that common standards are necessary in order to achieve desirable properties in the areas of distributed data bases and automatic data restructuring.

It is argued that a user-oriented common logical structure can span across the different views of data base organization (network, relational and hierarchical), and that a query language and a data format based on this common logical structure are sufficient to provide the desirable properties in the areas mentioned above. This paper includes a description of approaches that significantly simplify the problems in these areas by taking advantage of the common standards.

I. INTRODUCTION

Recent advancements in computer networking technology bring about the potential of using computer systems in new cooperative ways. One of the most exciting and promising areas is that of distributed data bases. There are many reasons for the need to have data distributed, but the most prevalent is the organization of data according to their functionality, thus allowing for local applications to be performed efficiently, while still permitting global operations to take place. For example, consider several hospitals being put on a computer network. Most of the processing needed will be done locally for every hospital, but some global operations, such as statistics, summaries, or search for an appropriate donor, could be performed over the network involving several or all of the hospitals. Similar situations and applications can be imagined in enterprises such as banking, inventory management, libraries, and law enforcement. Another need for distributed data might arise when very large data bases exist. In that case, data can be distributed over several facilities for the purpose of parallel access to the data.

In order to facilitate distributed data bases, it might be necessary to transfer data bases from one application environment to another across computer network nodes. In addition, if a dynamic network data management is to exist, it is necessary that data organized by existing application systems can be converted and transferred into new more advanced systems. These are some of the reasons for the search of generalized tools for data base conversion and transfer.

We make the assumption that most data bases of the future will be organized under generalized data management systems. Under this assumption, we examine in the next sections the properties that are desired in order to facilitate distributed data bases and

generalized data base conversion. Next, we describe features of common standards for logical structures, data formats, and query languages, and point out how the adoption of such standards can contribute greatly to solutions in the distributed data bases and data base conversion areas.

2. DESIRED PROPERTIES IN DISTRIBUTED DATA BASES

Distributed data bases impose several problems that are beyond the technology of computer network communication. In order to achieve distributed data base systems, it is necessary to smooth out the differences between the (possibly) disparate data management systems (DMSs) that manipulate data on the network. It is also necessary to interface these systems in such a way that a user will be unware of the fact that he is dealing with different systems across a network. Finally, when a user deals with data that is physically distributed over several systems, he should be able to think and refer to it as a single data base.

There are several approaches to making all data available to all users on a computer network. The most obvious ones are the standardized and the centralized approaches. In the standardized approach, all DMSs on the network are required to be the same (possibly on different hardware), thus imposing the same query language, logical structures, and data base organization to be the same across the network. In the centralized approach, a central facility provides all data management services for the entire network. There are clear advantages to these approaches, but environments where such limitations can be imposed on a community of users, are very few and uncommon. In addition, these approaches stifle a dynamic growth of data services, since each change of data management services affects the entire community of users, rather than being localized.

Another approach, which can be called the integrated approach, is one that permits individual systems to exist on the different nodes of the computer network. In this approach, the integration of different DMSs into a cooperating working unit can be achieved by providing appropriate interfaces between the systems. It will be shown later that the adoption of common standards simplifies or even eliminates the need for some interfaces. The above approaches were discussed in more detail in [1].

Some of the properties that are desired in distributed data base systems are discussed below.

a. Allow different DMSs to exist on the network. Different DMSs exist, because they offer different cost effective features suitable for certain applications. For example, a system designed for fast retrieval requires index mechanisms which tend to slow down updating.

b. Allow data existing on different systems to be shared. Mechanisms for correlating existing data bases are necessary if data bases need to be physically distributed.

c. Allow evolutionary integration of the DMSs. When a new DMS is added to the network or replaces an existing DMS, it should cause a minimum of disruption to the network.

d. Fail-soft properties. A distributed data base system should allow for a degraded service in case of a local failure.

e. The additional cost for achieving distributed data base systems in terms of response-time and implementation should be small relative to the cost of data base management and insignificant relative to the benefits achieved.

3. DESIRED PROPERTIES IN DATA BASE CONVERSION

In the programming languages area, higher level languages were developed in order to separate the algorithmic part of a program

from physical implementation considerations. Unfortunately, there was no equivalent trend in the data base area. When a data base needs to be moved to another environment (different hardware, operating system, and DMS), the physical characteristics of the data base have to be dealt with directly. The reason for this is that physical characteristics of data base organization are embedded in the data base itself. For example, pointers for efficient retrieval are often embedded in the data records or in indexes which constitute part of the data base.

The solution to this problem lies in supplying facilities equivalent to compilers as part of data management support utilities. Most DMSs have a load or generate facility that can load a data stream and generate a data base according to internal linking and indexing requirements. A similar facility for getting the data base (or a portion of it) out of the DMS into the original data stream is necessary in order to achieve data independence from physical data base considerations. If a standard format for data stream is selected, then data bases could be transferred between systems through the logical data stream. Of course, logical restructuring of the data might still be necessary, but this process can be done independent of physical considerations of data base organization. Furthermore, a data base could be output, restructured, and then loaded into the same DMS in its new logical organization.

From the discussion above, it follows that a desirable property for data base conversion is the independence of data from physical characteristics imposed by DMSs, operating systems, and hardware. In order to achieve this independence, another property is implied: that there is an underlying common logical structure between the DMSs involved. As will be discussed in the next section, common standards for logical structures form the basis for common standards for data formats.

4. COMMON STANDARDS

The subject of standards is distasteful to many people for two main reasons. First, the details are often boring and seem to be immaterial. It is indeed unimportant, for example, whether a certain code is assigned to the letters of the alphabet (except maybe for their statistical use). What is important is the fact that certain areas achieve great advantages if standardized. Second, standardization sometimes imposes unnecessary restrictions. This is a very important point, since overstandardization can indeed restrict in unnecessary ways. The challenging task in standardizing an area is to select the smallest subset of elements to be standarized, yet achieving the desired properties sought after.

It is in this light that we view the next sections. We believe that standardization in the data base area should be limited to logical properties of data structures, data formats, and query languages. An attempt to standardized physical properties, imposes unnecessary limits on the implementation of systems and inhibits the dynamic change of the data base technology.

4.1 DATA STRUCTURES

The recent controversy over the relational vs. network views of data [4], accentuates the need for a common standard in the data structures area. There are two main differences between these views. First, the relational view of data [2] emphasizes logical properties of data structures, while the network view, as expressed in the CODASYL Data Base Task Group Report [3], includes features which describe physical properties of the data as well. For example, the CODASYL data description language (DDL) includes features for specifying how data are to be organized in physical storage, such as "areas" or "search keys." Second, in the relational view, the relationships between sets of entities (relations) are implicit, while in the CODASYL network version these relationships are made explicitly through the "set" definition facility. For example, if

in a relational model two relations about employees and their children were defined, then the relationship between employees and children will exist only if there was a column in the children relation containing some identifying information about employees (say, his man number). This is an implicit way of specifying relationships.

There is an apparent advantage to having implicit relationships. Under this representation, queries can be asked spanning more than one relation, as long as there is a way to correlate these relations through some common columns. If these relationships were made explicit and access limited only to explicit relationships (as is under CODASYL network), then there might be some potential relationships that cannot be used. However, in actual implementaions of relational systems (such as INGRES [5]), certain relationships are being chosen for indexing in order to avoid exhaustive search of the involved relations. If a query involves a relationship which is not indexed, then the cost for its processing is significantly higher. Similarly, a logical structure that permits explicit specification of relationships is not limited to them and could permit queries through implicit relationships, possibly at a higher cost (depending on implementation techniques). The main point is that explicit or implicit relationships should not depend on the projected implementation but on their usefulness to the user.

It has been observed before (for example, in the DIAM model [6]) that it is most natural for people to think about data as sets of entities, each entity having a set of properties belonging to it or describing it, and relationships between the entities. Furthermore, it is important to know whether the relationships are one-to-one (e.g., employee-car, assuming he has only one car for his exclusive use), one-to-many (e.g., employee-children), or many-to-many (e.g., Department-cars, assuming that departments can have many cars, and that cars are shared by many departments). In a less likely event, a relationship may be unspecified since it changes in time.

We believe that a common standard for logical data structure should follow this line of representation. There should be a facility for specifying entities and their properties (including a facility for describing the format of values) as it is done in the relational view, and there should be a facility for explicit expression of relationships without going into the physical details of the CODASYL version. Physical details should be left to a data base administrator (DBA) who is responsible for the organization of the data base in the most cost effective way for the projected applications using the data. A common standard for logical structures along the lines described above will accommodate both the relational and the network views, as well as hierarchical representation of data (which can be thought of as a special case of networks).

4.2 QUERY LANGUAGE

In order to determine whether a standard query language can be specified, one needs to determine to what degree the functional properties of query languages are dependent on the particular DMSs involved. In studying several query languages, we have concluded that the major features of query languages are based on the underlying logical data structure [7]. This fact is also evident in observing the power of relational query languages (such as SEQUEL [8]), which are based on logical structures only. Clearly, a query language which is quite powerful and useful can be defined for a standard logical structure based on the guidelines in the previous section.

Functionally, query languages are based on two basic functions: entry-functions, in which a collection of entries from entity sets are selected according to a selection criterion (e.g., salary > 1500), and value-functions, which operate on these entries to produce a set of values (e.g., employee name). When we add conjunctive operators and let entry-functions and value-functions interact recursively, the resultant query language is quite powerful. Such a query language was developed in [7] for hierarchical data structures. Unlike

relational query languages which require that every relationship used is expressed explicitly in the language (usually by equating the appropriate columns from the associated relations), a query language based on logical structures with explicit relationships does not need this facility. This simplifies the use of the language for queries involving explicit relationships.

It is sometimes useful to allow for more than one set relationships to exist between the same two entity sets. For example, for the two entity sets about "people" and "cars," one can define two set relationships about "owner of car" and "driver of car." Assuming that not all owners drive their own cars, these two set relationships are indeed different. In such a case, a facility for specifying which set relationship to use, is necessary when a query involves these entity sets.

Another facility that may be chosen for incorporation in a common query language, is the use of "correlation variables." Such a facility exists in relational query languages, and is used to express linkage between two (or more) parts of the query. For example, suppose that an entity set (relation) exists for employees and it contains information about employee salaries and employee managers. In expressing the query for "who are the employees that earn more than their managers," one needs to use a "correlation variable" for an employee in order to link his salary to the salary of his manager. (This example was discussed in [8].)

4.3 DATA FORMATS

Once a standard logical structure is specified, standard data formats are merely a way of representing a data stream for a given logical structure. Following the guidelines for standard logical structures given in Section 4.1, we need to specify a format representation for instances of entities and their properties and for instances of set relationships. The notation and formats can be chosen quite arbitrarily, but some general guidelines are stated next.

There must be a mechanism to identify the beginning and end of each entity set and each set relationship. The order of the entity sets and set relationship in the data stream is immaterial. There must be a mechanism for identifying the beginning and end of each instance occurrence in entity sets and set relationships. Finally, there must be a mechanism for identifying the beginning and end of each value within the instances. Because values and instances may be of variable size, this mechanism should be in the form of a special marker or a count of characters.

The representation of values should be always in character format, since the standard format must be independent of any specific machine representation, such as binary or hexidecimal. For character codes, one can select the standard ASCII code.

Data streams are not self-sufficient; they can only be interpreted by using the logical structure description of the particular data base. Therefore, a data base represented in a standard form should contain its logical data description in the standard logical form and the data stream in the standard format. This combination of information represents a data base independent of any specific implementation, DMS, operating system, or hardware. In this form, it could be transferred between systems, restructured if necessary, or stored in an archive for future reference.

5. AN APPROACH TO DISTRIBUTED DATA BASES

Consider a conceptual view of distributed data bases on a computer network: the set of all users regardless of their physical location on the one hand, and the set of all DMSs regardless of their physical location on the other. We wish to allow each user to potentially access any data base on the network. In this section, we point out the advantages gained by using common standards to achieve this goal. We also point out the implications of adopting these common standards on the distributed system.

Let us assume at first that a user needs to access a single data base residing on a single DMS. If he uses the standard query language and the target DMS accepts this query language, then he can query in this way any data base on the network. In reality, when dealing with existing DMSs, a query translation from the standard query to the particular DMS query language is necessary. This translation could be done externally to the DMS or internally within the DMS. It can be expected that as the distributed data system grows in use, more DMSs will be built or modified to accept the standard query language. Note that there are no implications on the internal workings of DMSs and their organization of data bases as a result of accepting the standard query language. On the user side, it is often desirable to use different forms of query languages in order to accommodate different types of users (casual user, sophisticated user, computer-naive user, etc.). User interfaces to the standard query language could provide users with different query forms such as natural language, templates, graphical inputs, etc. Indeed, the standard query language could be a concise formal language for the purpose of transmission over the network only.

The data extracted by the DMS in response to the query can then be returned to the user in the standard data format. This implies that individual DMSs are no longer required to have report generation facilities. This function should be performed at the user end by invoking a utility program. Utility programs of this kind could be local to a network node or could be a global service available somewhere on the network. A default case can be provided by the user interface to display the response in a predetermined format.

In order to isolate the user from the physical location of the data base, a directory can be provided to map data base names into locations. The user interface can use this directory to direct the query to its proper destination DMS.

Suppose now that several data bases are physically distributed and that these data bases or a part of them need to be treated and thought

of as a single data base. With the availability of a standard query language and a standard data format, this need can be achieved by correlating the data bases in a (central) directory. The logical data structure of the combined data base can be described and a mapping between it and the different logical descriptions of the data bases provided. A query of a user to the combined data base will result with a set of queries sent to all the appropriate DMSs. The responses, which return in the standard format, will have to be combined into a single response. We do not mean to imply that the techniques for generating multiple queries from a single query and the correlation of the responses are simple, but rather to point out that the availability of common standards could help make this task manageable.

6. AN APPROACH TO DATA BASE CONVERSION

From the discussion in Section 3, it follows that the approach to data base conversion and transfer is along the lines of a standard data format. According to this approach, DMSs should have a facility for "loading" and "unloading" a data base from and to the standard data format. Since the standard data format has only logical characteristics, data represented in that format contain no characteristics that are dependent on the DMS involved. If no restructuring of the logical structures is required, then the data base transfer from one DMS to another requires nothing more than "unloading" it from the source DMS and "loading" it in the target DMS.

It is often necessary to change the logical structure of a data base or a portion of it, either to fit it better for a new application or to reorganize it for purposes of efficiency. By going through the standard data format (using "load" and "unload"), the restructuring process can be independent of any physical considerations about the data base organization. Specifications for the physical organization of data are made separately by a data base administrator. Furthermore, since the restructuring facility is independent of any DMS, it can become a utility program available for all restructuring needs.

In today's reality, however, different DMSs require different formats for input data streams and with few exceptions have no "unload" facilities. As described in [9], this requires source and target reformatters to map data to and from a standard data form. Data from the source system is extracted by using the query facility, then the source reformatter converts it to the standard form. After the data is restructured, it is reformatted again to the format acceptable by the load program at the target DMS. It is clearly advantageous to adopt a standard data format and avoid the reformatting process.

7. CONCLUSION

There are many problems in the distributed data bases area that are still open and that were not mentioned here, such as security, who manages the (central) directory, and responses to users in case of failures. They are beyond the scope of this paper. Our purpose was to express our belief that if there is any hope for achieving truly distributed data systems, common standards for logical structures, query languages, and data formats along the guidelines discussed here must be developed and adopted. In the area of data base conversion and transfer, standard data formats can have a great impact on the independence of the data from DMSs.

8. REFERENCES

1. Shoshani, A., Data Sharing in Computer Networks, WESCON Conference, September 1972.
2. Codd, E. F., A Relational Model of Data for Large Shared Data Banks, Comm. ACM 13 (June 1970), pp. 377-387.
3. CODASYL, Data Base Task Group Report, April 1971.
4. Date, C. J. and Codd, E. F., The Relational and Network Approaches: Comparison of the Application Programming Interfaces, 1974 ACM SIGMOD Workshop.
5. Held, G. and Stonebraker, M., Storage Structure and Access Methods in the Relational Data Management System, INGRES, Pacific 75 Conference, San Francisco, pp. 26-33.

6. Senko, M. E. et al., Data Structures and Accessing in Data-Base Systems, IBM Systems Journal, Vol. 12, No. 1, 1973.

7. Shoshani, A. and Spiegler, I., The Integration of Data Management Systems on a Computer Network, AIAA Computer Network Systems Conference, April 1973, Huntsville, Alabama.

8. Astrahan, M. M., and Chamberlin, D. D., Implementation of a Structured English Query Language, Comm. ACM 18 (October 1975), pp. 580-588.

9. Shoshani, A., A Logical-Level Approach to Data Base Conversion, Proceedings of ACM SIGMOD Conference, May 1975, pp. 112-122.

AN APPROACH TO DATA MIGRATION IN COMPUTER NETWORKS

Nan C. Shu
Vincent Y. Lum
Barron C. Housel

IBM Research Laboratory
5600 Cottle Road
San Jose, California 95193

ABSTRACT: This paper presents an approach for data migration in a computer network. The approach views data migration as a three-step process: In the first step, a conversion interface utilizing source system facilities transforms the "raw" source data into a normal form of linearized files. In the second step, the linearized files are processed by a converter/translator where the restructuring takes place. The result is a set of linearized target files. In the third step, a conversion interface utilizing the target system facilities, loads the linearized target files into the target system resulting in the final physical target data. To achieve data conversion/translation using this approach, two languages have been developed: (1) a data description language, DEFINE, and (2) a mapping specification language, CONVERT. Both languages are high level and non-procedural, and are believed to have the power to deal with most situations encountered in data migration. Scenarios are described for using this method in a computer network.

RJ 1703 (#25136)
January 13, 1976
Computer Science

I. INTRODUCTION

An important development in recent years is the advances in the computer networking. Take the ARPA network (APRANET) for example. Under the development by the Advanced Research Projects Agency, it has expanded, within a few years, from a 4-site system on the West Coast to a 40 site system distributed across the United States, linking over 60 host computers to provide computer resource sharing among several thousand ARPA-related research workers [1]. Effective sharing of resources [2] is a common goal of computer networks so that a user of any one system can freely make use of the resources of any other system in the network. Data, of course, is one of the important resources to be shared. Consequently, it is paramount that a general methodology for data migration in computer networks be developed.

Support of data migration between two distinct network nodes is in general quite complex. It requires either data reformatting or data restructuring or both.

Reformatting is the kind of transformation necessitated by a change in the physical or encoding properties of data (e.g. data representation from ASCII to EBCDIC). Until recently, most of the work on network migration have dealt with data reformatting. The Data Reconfiguration Service [3] and the DSCL Project [4] have demonstrated its feasibility at both the field and the record levels.

Restructuring, on the other hand, involves changes in data structure. Here, the problem is more complex due to the need to specify how the instances of source items from one or more files are to be mapped into the instances of target data of one or more files. The complexity varies according to the extent of restructuring and the criteria specified for the restructuring to take place.

It is our belief that, in step with the growth of data base systems and the broadening of application spectrum, data migration involving restructuring will be done more frequently. For example, crime reports gathered and maintained in various cities inherently constitute a data base of more than local interest. It is natural to think of these related data files at different sites as elements of a single unified data collection [5]. However, more often than not, the local data files are managed by disparate information management systems and cluttered with non-essential data (from a different application's point of view). To treat them as elements of a single unified data collection would require at least the capabilities to decompose the local files, extract the relevant information, map the extracted data into a structure suitable for the application in hand, and combine the data obtained from multiple sources into one collection. In other words, when fully implemented, support for network data migration should provide capabilities comparable to those commonly desired in a general methodology for data conversion and restructuring.

Data conversion and restructuring, in general, is a complex problem requiring more of our attention than it has received in the past [3-18]. To simplify this task, the authors have proposed [15-17] a conversion model whereby linearized files are the normal forms of data to the converter/translator. To achieve data translation using this approach, two high-level non-procedural languages have been developed: (1) a data definition language, DEFINE, capable of describing the linearized source and target files, and (2) a translation definition (or mapping specification) language, CONVERT, which provides very powerful and highly flexible restructuring capability for the mappings between the source and target data. This paper explores the plausibility of applying this methodology to data migration in the computer network environment.

II. A GENERAL DATA CONVERSION MODEL

Researchers in the area of data conversion have recognized that the difficulties in data conversion arise from the fact that the data structures are both system and application dependent. One practical approach [6, 15] is to utilize existing source and target system facilities (e.g. access methods) to obtain a system-independent representation of the user data. In this way, many of the system-dependent gross incompatabilities at the storage level (e.g. access mechanism, indexing organization, data compression and decompression methods) and at the physical level (e.g. device-dependent control information) can be handled by the source and target systems, while the conversion at logical level (e.g. hierarchies, relations, data justification in a field, etcetra) can be handled by the conversion system. Such a system will be easy to use by the average user who knows the semantics and the logical structures of his data, but does not have the detailed knowledge of how the data are stored or how to access them.

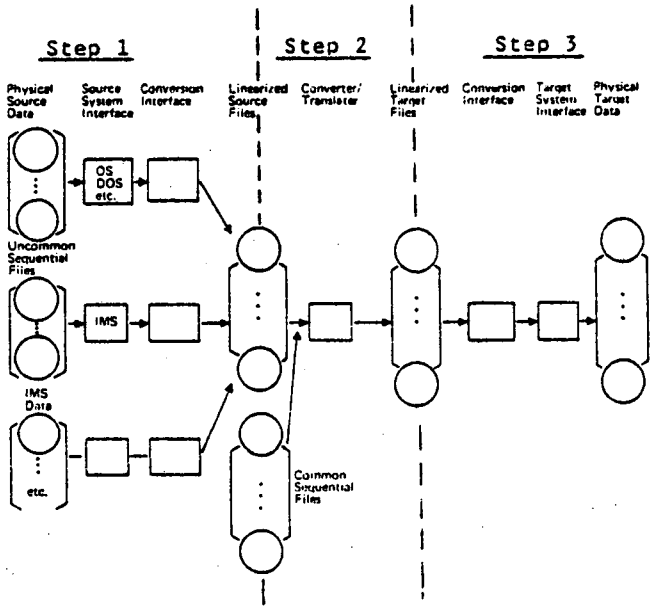


Figure 1 Three stpes of Conversion

This approach leads us to view data translation as a three step process as shown in Figure 1.

In the first step, a conversion interface utilizing source system facilities transforms the "raw" source data into a set of linearized files. In the second step, a converter/translator restructures this set of linearized files into a different set of linearized target files. In the third step, a conversion interface utilizing the target system facilities loads the linearized target files into the target system resulting in the final physical data.

Thus, the linearized file can be viewed as the normal form of data for the converter/translator. At this time it is appropriate to discuss further what is a linearized file and why it is a natural choice for a normal form of data.

We define a linearized file to be a file belonging to that subset of sequential files describable by our data definition language. It may have a flat or hierarchical record structure. It may contain self-defining data, terminators of different kinds, multiple record types within the same file, and repeating or non-repeating groups belonging to the same logical record but appearing in separate physical records. It does not, however, contain any system-dependent accessing or alignment information. If direct addressing exists, it must be replaced by symbolic addressing. In this manner, most of the sequential files, which are preponderant at this time in the real world, can be directly used as input to our conversion model.

In case the data comes from a database with complex structure, we expect the users to use the source systems' support, including utilities, to decompose the data base into linearized files, stripping all the unneeded control information and replacing physical or direct pointers by symbolic or key pointers. For a given

structure, the linearized files can be expected to be different depending on the user. For instance, consider the example as defined in Figure 2(a). One can create the linearized files for this data base as given in Figure 2(b) where each physical data base becomes one linearized file. Alternatively one may want to create a set of linearized files as given in Figure 2(c). While it seems there are an arbitrary number of ways to define linearized files, in reality the limitations existing in a system and its support and the naturalness of the resulting files will dictate the choice. This choice, however, has only a very small impact to the conversion process because using the translation definition language, one can alter this choice easily.

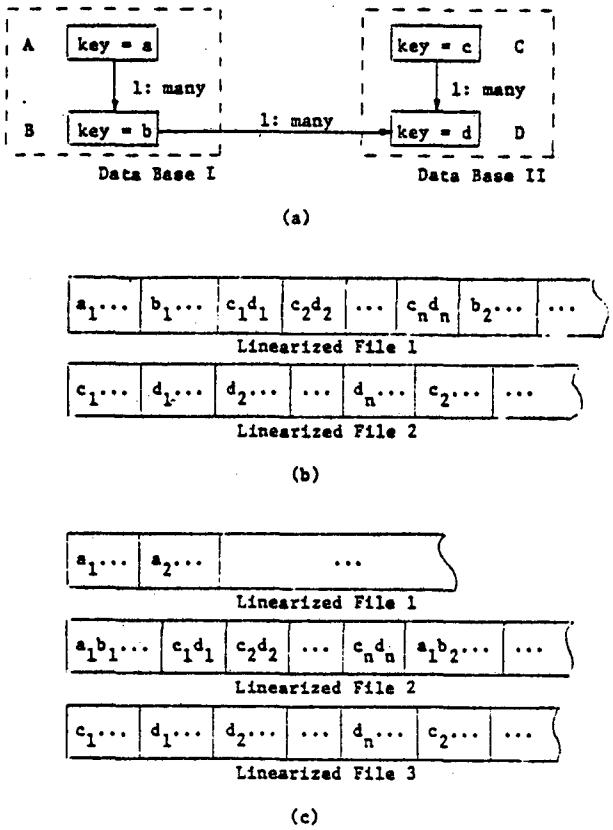
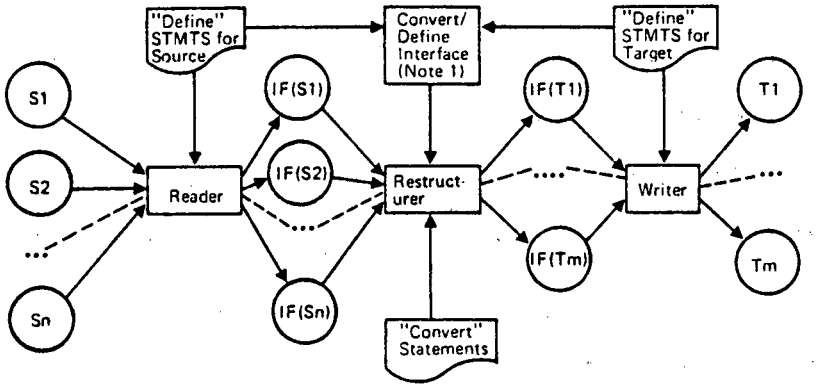


Figure 2 Linearization of a Data Base

In a similar manner the target structure will be defined in terms of linearized files. These files should be defined in such a way that easy loading is possible.

The heart of the system is the converter/translator (in the Step 2 of the conversion process) which produces linearized target files from the linearized source files. The inputs to this component are the descriptions for the source and target using DEFINE and the specification for the restructuring using CONVERT. A more detailed schematic of this process is given in Figure 3.



Note 1: Logical views of the files are extracted from the define specifications.

Figure 3 Model of Translation

Conceptually, the READER using the DEFINE descriptions for the linearized source files (S1,...,Sn), reads each file and translates it into an internal form (IF) suitable for manipulation by the restructurer. From the DEFINE descriptions, logical views of the files, called "templates" are extracted. Next, the restructurer component, driven by the CONVERT specifications, generates target files which are also in IF (i.e. IF(T1),...,IF(Tm)). The WRITER, using the DEFINE target descriptions, then transforms the internal forms to the desired linearized target files T1,...,Tm.

III THE PLACE OF THE GENERAL MODEL IN THE NETWORK DATA MIGRATION

In computer networks, given the capability of data translation, two basic approaches to data migration between heterogeneous hosts have been identified by Kimbleton and Schneider [2]:

"• Transmission of a data item request to a remote site followed by return of the requested item; and

Retrieval of the file containing the requested item from a remote site followed by local access to the desired data item."

Now, let us explore how the general methodology that we have described above can be adopted in either of these two approaches so that the full capabilities of data conversion/restructuring can be provided. To simplify the problem structure, we shall assume that the facilities for data reconfiguration (e.g. DRS, DSCL, etc.) are available so that we need not be concerned with the machine-dependent incompatabilities such as character and word sizes, internal code sets, numeric data type representation, parity checks, etc. We shall further assume that each host has the converter/translator required for the step 2 of the conversion process, and also has the conversion interface capable of making the transformation between the host-system supported files and the linearized files if the former are not already in linearized forms.

As an example, let us visualize the network as shown in Figure 4, where C denotes host computer system and IMP denotes Interface Message Processor. Let us suppose that a user at node T wishes to conduct a nation-wide salary survey based on information stored at nodes S1, S2,...S5. He plans to access the salary data along with some other vital information (such as individual's age, education, skill, sex, length of employment, etc.) from these remote sites. He knows that this information is available at these nodes, but does not know in which files do these fields reside, nor how are they structured.

(It is likely that they are structured differently in different nodes).

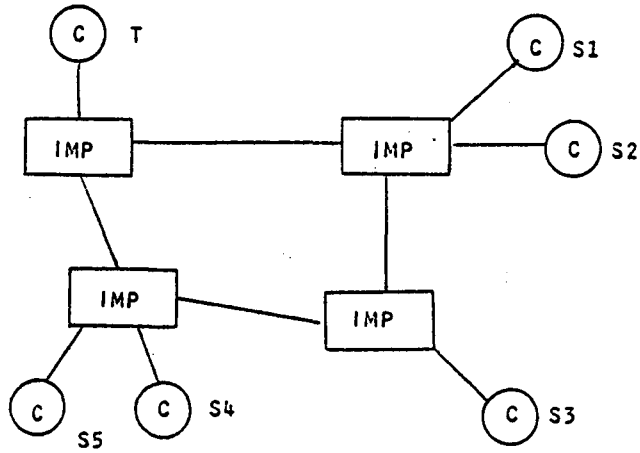


Figure 4. A Computer Network Example

If he decides to take the first approach, data conversion would be done at the source sites. The following scenario can be visualized.

1. The user at node T will send a request to each of the source nodes for the data that he is interested in. Since he does not expect the source nodes to know the data structure that is appropriate for his purpose, he will send them, along with the request for data, a description of his desired target file in DEFINE statements.
2. At each of the source sites, the target description sent by node T will be compared with the DEFINE description of the (conceptually) linearized source files containing the requested data. Assuming that there is no ambiguities in the semantics of data, á translation analyst at the source node can proceed to write the mapping specifications in CONVERT.
3. A conversion interface at the source node will be used to transform the source files into linearized files, if they are not already in linearized forms.

4. The linearized files and the CONVERT statements, along with the DEFINE statements for both the source and the target data will be inputted to the converter/translator at the source site.
5. The extracted and restructured data in linearized form produced by the converter/translator will be transmitted to the user at the target node T.
6. Depending on the application program running on node T, the user may either use the linearized files directly, or he may invoke a conversion interface at his own node to cause the linearized files to be transformed into the desired physical forms.

On the other hand, if the second approach is adopted, the actual restructuring will take place at the target site. One can envision the following steps.

1. The user at the Node T will send to each of the source sites a request for description of the desired data.
2. The source site will send back to the requester at T, the DEFINE description of the source files containing the requested data.
3. At the target site, each of the DEFINE descriptions of the source files will be compared with the description of the desired target structure. Assuming that there are no ambiguities of the semantics of data, the requester at node T will proceed to write the mapping specifications in CONVERT.
4. A request for data will then be sent to the source nodes.
5. At the source node, a conversion interface will be used to linearize the requested source files if they are not already in linearized forms.
6. The linearized files will then be transmitted to the requester at node T.

7. Using the DEFINE statements (for both the source files and the desired target form) and the CONVERT statements (for mapping), the converter/translator at node T will be invoked to extract and restructure the relevant data into linearized target file(s).

8. Same as the last step in the first approach.

A modification of the second approach is the construction of a network-wide directory. One may then replace the steps 1 and 2 above by looking up the source descriptions in this directory. In this case, one anticipates that the conceptually linearized files have been described, using DEFINE.

It seems that in most cases, the first approach will be more efficient since one needs to transmit only the desired data. Hence the volume of data transmission will be less. However, it does put the burden of conversion on the sender instead of the requester. In either case, the notion of using the linearized file as a normal form of data as well as the two languages, DEFINE and CONVERT, could play an important role in the network migration.

IV. CONCEPTUAL DATA REPRESENTATION

We shall now proceed to discuss the data description language DEFINE and the mapping specification language CONVERT. Before we do that, we must understand the representation of the data as viewed by the translation analysts and the conversion system's Data Restructurer. A little thought on this subject convinces us that it is paramount to define a representation or form that is simple, capable of representing different data structures and familiar to data conversion analysts. While there are many candidates for this role, most of them can be eliminated because they are too restrictive or require too much learning. Our final choice was the tabular form for hierarchically structured data. All data structures will be viewed in this tabular form, which from now on will be referred to simply as the Form.

Let us take a simple example to illustrate the tabular form, or simply the Form. Figure 5(a) represents the schema of a hierarchical data structure. The actual data may be organized as COBOL records, in which case the information on education, skill, or child may be repeating groups, or the data may be organized as a tree as in some data base systems (e.g., IMS [19]). In either case, the data can be represented by the Form as shown in Figure 5(b). Note that additional visual aids can be included in the Form's outlay if so desired by the user.

The use of the Forms to represent data and its structure is believed to possess many advantages. Included among these are that the Forms are easy to visualize, they can represent other data structures readily (see Section VII) and they are easy to manipulate. We shall see later that one can define a very simple and powerful translation definition (or mapping specification) language to operate on these Forms.

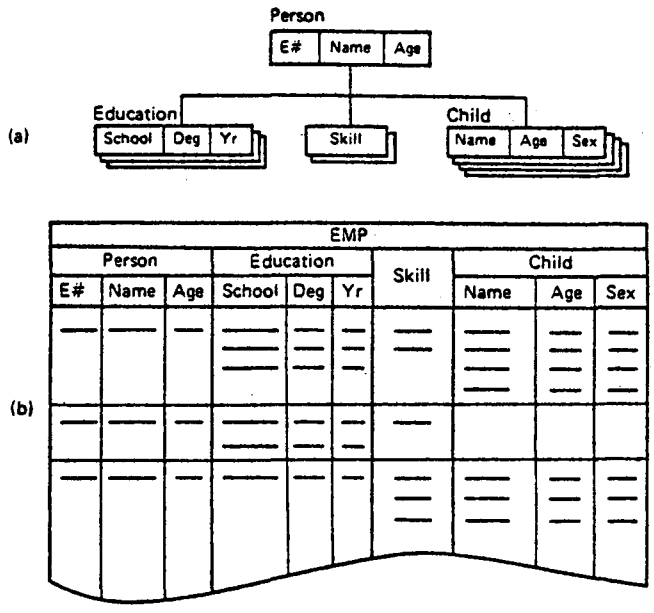


Figure 5. Translation Analyst Views Data Conceptually in Terms of 'Forms.'

V. DATA DESCRIPTION LANGUAGE - DEFINE

Having described our model for data translation (Figure 3), we see that a data description language is needed for two purposes. The first is to provide means for describing a wide spectrum of hierarchically structured linear files to enable them to be converted by the READER (or WRITER) to (or from) the conversion system's internal form. The second is to provide the basis for extracting logical views of the files for use in the restructuring process.

As in all data description languages DEFINE provides constructs for describing encoding characteristics of data items such as character and numeric representations (e.g. BCD, EBCDIC, etc.). However, the unique features of the language lie in its rich facilities for stating many of the self-describing characteristics of files which normally must be decoded in the procedure portion of the application program which access the file. Specifically, facilities are included for specifying: a) optional data, b) self-defining data, c) complex termination criteria for variable length and repeating data, and d) validation constraints which must be satisfied for successful data translation. The language has no facility for describing the access mechanism of the data. Each file name in the data description provides a symbolic linkage to data and resource description statements (e.g. JCL) for the operating system under which the translator runs.

Details of the DEFINE language can be found in [16]. Here we shall only use an example to illustrate the general flavor of DEFINE and reveal some of its typical constructs.

Consider a parts-supplier (PTS) file with variable length records whose fields are P#, DES, S#, CN, and UC, standing for part number, description, supplier number, company name, and unit cost, respectively. The last three fields are grouped together to form a supplier repeating group. Further,

the file is so structured that each logical record (Figure 6(a)) ends with a $ sign. A DEFINE description for this file is given in Figure 6(b).

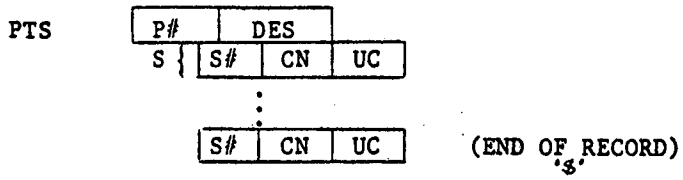


Figure 6(a)

```
DATA DESCRIPTION:
    DECLARATIONS:
         CHAR CODE IS EBCDIC;
         NUMERIC ENCODING IS IBM 370;
         CONSTANT BLANK IS HEX'40';
    END DECLARATIONS;

    SOURCE FILE DESCRIPTION (PARTS-SUPPLIER):
         GROUP PTS:
              OCCURS FROM 1 TIMES,
                   FOLLOWED BY END-OF-FILE;
              KEY IS P# WITHIN PTS;
              ITEM P#:
                   CHAR (PICTURE IS 'AA99');;
              ITEM DES:
                   CHAR (PICTURE IS 'A(5)',
                        JUST IS LEFT
                        AND PAD CHAR IS BLANK);;
              GROUP S:
                   OCCURS FROM 1 TIMES,
                        FOLLOWED BY '$';
                   KEY IS S# WITHIN S;
                   ITEM S#:
                        DEC (PICTURE IS '9(4)');;
                   ITEM CN:
                        CHAR (10);;
                   ITEM UC:
                        BINARY (15);;
              END S;
         END PTS;
    END FILE DESCRIPTION;
END DATA DESCRIPTION;
```

Figure 6(b) DEFINE Description of PTS File.

VI. TRANSLATION DEFINITION LANGUAGE - CONVERT

From the logical views extracted from the DEFINE description of data

structures, a translation analyst can visualize his data in view of the Form (Figure 5) and proceed to write mapping statements for manipulating his data into the format required in the target description. In order to achieve this goal, a translation language must have the following capabilities:

(1) Combining components of different files to form new files according to some specified criteria.

(2) Decomposing a file to form different files according to some specified criteria.

(3) Rearranging the order of instances within a file in some manner.

(4) Altering values of instances in different manners. Sometimes the new values may be derived arithmetically from the old values and other times new values may be arbitrarily defined from the old ones. An example of the former is the changing of payroll from hourly pay to weekly pay. An example of the latter is that the field 'sex', previously containing 'male' or 'female', will now become '1' or '0'.

(5) Formation of complex data structures or decomposition of them.

These are but some of the minimum capabilities required from a translation definition language. In short, the language should have all the capabilities of a data manipulation language that is required to handle the commonly encountered practices in data conversion.

The data mapping and restructuring facilities in CONVERT are provided by a set of Form operators. They include SELECT, CASE-SELECT, SLICE, GRAFT, MERGE, SORT, ELIM-DUP, CONSOLIDATE, a set of built-in-functions (SUM, MAX, MIN, AVG, and COUNT) and assignment. Each of these Form operators operates on one or more Forms (or components of them) and produces a Form as a result.

In this paper we shall only discuss the operation SELECT and GRAFT. A detailed description of other operators can be found in reference [17, 18].

We shall use some simple examples to illustrate these two operations. As shown in Figure 7(a), PTS is a parts-supplier file described in Figure 6. INV is an inventory file containing P# (i.e., part number) and QH (i.e., quantity-on-hand). SUP is a supplier file having CN (i.e., company name), S# (i.e., supplier number) and CA (company address).

1. SELECT

This operation selects part(s) of a Form if specified conditions are satisfied. For example, let us suppose that we want to create a new file consisting of part numbers of the parts supplied by suppliers located in San Jose, together with the corresponding part descriptions and their suppliers code numbers and names. This could be achieved by stating the SELECT operation as follows:

```
SELECT (P#, DES, S#, CN FROM PTS WHERE PTS.S# = SUP.S# AND SUP.CA = 'SJ');
```

In this case, PTS is the source file from which a target Form consisting of P#, DES, S#, and CN are to be constructed. However, not all instances in the source file will produce an image in the target because we are interested in only those instances where the suppliers are in San Jose. Since the information about the location of a supplier appears only in the SUP file, we must find the tie between the PTS Form and the SUP Form through the use of some common information which in this case is S#. Hence, we have the specified conditions stated as shown. The resulting Form is shown in Figure 7(b).

2. GRAFT

GRAFT combines two or more Forms into one Form when specified conditions are met. Since GRAFT operates on two or more Forms, it should be apparent that matching conditions are needed to tie the Forms together. For example, suppose we wish to form one file from the PTS and INV files such that the resulting file will have the information of the PTS file plus the quantity-

PTS

P#	DES	S		
		S#	CN	UC
2	X	4	AB	5
		2	BB	4
3	XX	4	AB	2
		1	XB	3
7	Y	7	C	7

INV

P#	QH
2	10
3	17
4	5
7	20

SUP

CN	S#	CA
AB	4	SJ
BB	2	MV
C	7	SF
D	3	LA
XB	1	SJ

(a) Source Forms

(b) Result of

```
SELECT(P#,DES,S#,CN FROM PTS
   WHERE PTS.S#=SUP.S#
   AND SUP.CA='SJ');
```

P#	DES	S#	CN
2	X	4	AB
3	XX	4	AB
		1	XB

(c) Result of

```
GRAFT(INV ONTO PTS WHERE
   PTS.P#=INV.P#);
```

P#	DES	S			QH
		S#	CN	UC	
2	X	4	AB	5	10
		2	BB	4	
3	XX	4	AB	2	17
		1	XB	3	
7	Y	7	C	7	20

(d) Result of

```
GRAFT(INV ONTO PTS WHERE
   INV.P# PREVAIL PTS.P#);
```

P#	DES	S			QH
		S#	CN	UC	
2	X	4	AB	5	10
		2	BB	4	
3	XX	4	AB	2	17
		1	XB	3	
4	-	-	-	-	5
7	Y	7	C	7	20

Figure 7. Examples of Form Operations

on-hand (i.e., QH) obtained from INV. This can be stated as GRAFT (INV ONTO PTS WHERE PTS.P# = INV.P#). Note that there are two tying fields: P# of PTS and P# of INV. Only the one in the Form after "ONTO" will appear in the resulting Form. The result is shown in Figure 7(c).

This way of stating conditions is useful in most of the cases. However, there are situations where some of the data exists only in some (not all) of the files that we are interested in. For example, P# = 4 exists in INV but not in PTS. By stating "PTS.P# = INV.P#" as the satisfying condition, we have excluded the inclusion of P# = 4 in our new file. What if for some reason we wish to include all P#s in our new file, leaving the missing information blank? To achieve this, we use the "PREVAIL" clause to specify the conditions.

The PREVAIL clause, in general, takes the following format:

$$f_1, f_2, \ldots \text{ PREVAIL } [f_j, f_k, \ldots f_n]$$

where $f_1, \ldots f_n$ are the names of the fields whose values are to be "matched". The names on the left hand side of the key word "PREVAIL" are considered to be the prevailing fields. The union of the instances of the prevailing fields determine the instances to be included in the resulting Form. Take the PTS and INV files for example. The statement GRAFT (INV ONTO PTS WHERE INV.P# PREVAIL PTS.P#); produces the result shown in Figure 7(d).

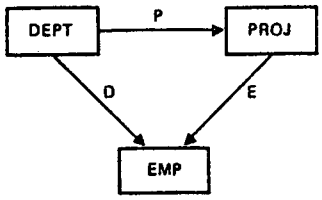
VII APPLICATION EXAMPLE

Earlier we have mentioned that other data structures can be expressed in linearized files, and hence the Forms. In this section we will discuss how this can be done. In addition, we will give an example how CONVERT can be used to specify the mapping in a data translation.

Consider the example of a DBTG[20] type of data structure as shown in Figure 8(a), with nodes DEPT, PROJ, and EMP and edges D, P and E where D links all employees belonging to the same department, P links all the projects in a

department, and E links all the employees working on the same project.

Our approach in dealing with this type of complex data structure is to decompose it into a family of hierarchies each of which can be represented as a Form. This decomposition, however, is not unique for a given structure. For instance, one way to decompose the structure in Figure 8(a) is to have each node corresponding to a Form as in Figure 8(b) where the connecting information, designated by each named edge, is embedded in the Forms.



(a) Data Structure

DEPT

D#	MGR	P#	E#
55	SMITH	P1 P2 P3	541 551 552 553 554 555 556
54	JONES	P4	542 543 544

EMP

E#	NAME	EDUCATION		SKILL
		DEG	YR	
-	-	- -	- -	-
-	-	-	-	-

PROJ

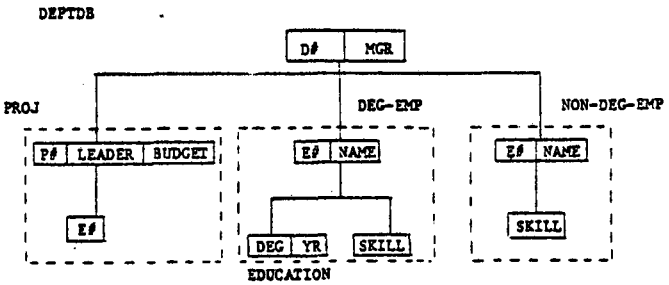
P#	LEADER	BUDGET	E#
P1	541	100K	551 552 541
P2	554	200K	553 554 555
P3	-	50K	555 556
P4	542	300K	542 543 544

(b) Each node and its edges represented as a Form

Figure 8. Example of a DBTG type data structure

Alternatively, each named edge can be in itself represented as a Form. Of course, a combination of both techniques is equally applicable.

To illustrate the application of the CONVERT language, let us suppose that the data structure decomposed into a representation as shown in Figure 8(b), will be reorganized such that the result will become one file with the structure shown in Figure 9.



D#	MGR	PROJ				DEG-EMP					NON-DEG-EMP		
		P#	LEADER	BUDGET	E#	E#	NAME	EDUCATION		SKILL	E#	NAME	SKILL
								DEG	YR				

Figure 9 Target Structure of Example

A translation specification for this example may be stated as follows:

```
DEMP ← SELECT (D# FROM DEPT,
               E#, NAME, EDUCATION, SKILL FROM EMP WHERE
               DEPT.E# = EMP.E# AND EMP.DEG ¬=0);
NEMP ← SELECT (D# FROM DEPT,
               E#, NAME, SKILL FROM EMP WHERE
               DEPT.E# = EMP.E#  AND EMP.DEG = 0);
TDEPT ← GRAFT (DEMP, NEMP ONTO DEPT WHERE
               DEPT.D# PREVAIL DEMP.D#, NEMP.D#);
DEPTDB ← GRAFT (PROJ ONTO TDEPT AT P# WHERE
               DEPT.P# = PROJ.P#);
```

There are obviously other ways of specifying the mappings in this translation. The CONVERT language is powerful enough that each translation analyst can specify the mappings in a way most natural to him.

IX. CONCLUSION

We have described a general methodology and model for data conversion. Our approach assumes that the source and target systems play an important role in the conversion process by transforming the source data into linearized files to be used as inputs to the conversion system and transforming the conversion system output into desirable target formats. The application of our method requires the users to describe the data structures of the linearized input and output files and to specify the mappings between the source and target data. As a result, we have defined two lanuages for this purpose: DEFINE for data description and CONVERT for mapping specification. Both languages are high level and nonprocedural by current standards and can handle most of the situations commonly encountered in data conversion.

Potentially, this general methodology for data conversion is applicable to data migration in the computer networks where the need for data restructuring arises from 1) transfer of structured data among disparate information management systems; or 2) materialization of different views of data for disparate applications. It is believed that if the capability of general data conversion is incorporated in a network operating system, the application spectrum of network users will be broadened.

ACKNOWLEDGMENT

The authors are grateful to S.R. Kimbleton and W. Chu for helpful discussions and to W. F. King III for his guidance and encouragement.

REFERENCES

[1] Frank, H. and Chou. W., "Network Properties of the ARPA Computer Network", Networks, 4, pp. 213-239 (1974).

[2] Kimbleton, S.R. and Schneider, G.M., "Computer Communication Networks: Approaches, Objectives, and Performance Considerations" Computing Survey (Sept. 1975).

[3] Anderson, R.D. et al., "The Data Reconfiguration Service - An Experiment in Adaptable Process to Process Communication", Proceedings of Symposium on Problems in the Optimization of Data Communication Systems, Oct. 1971, Palo Alto, Calif., pp. 1-9.

[4] Schneider, G.M., "DSCL - A Data Specification and Conversion Language for Networks" Proceeding of ACM SIGMOD Workshop, San Jose, Calif. May 1975.

[5] Casey, R.G., "Design of Tree Networks for Distributed Data", AFIPS Conference Proceedings 1973 National Computer Conference and Exposition, Vol. 42, pp. 251-257.

[6] Shoshani, A., "A Logical-Level Approach to Data Base Conversion", Proceedings of ACM SIGMOD International Conference on Management of Data, May 1975, pp. 112-122.

[7] Su, S. and Lam, H., "A Semi-Automatic Data Base Translation System for Achieving Data Sharing in a Network Environment," Proceedings of ACM SIGFIDET WORKSHOP, May 1974., pp. 227-248.

[8] Fry, J.P., Smith, D.P. and Taylor, R.W., "An Approach to Stored Data Definition and Translation," Proceedings of ACM SIGFIDET Workshop on Data Description and Access, 1972.

[9] Smith, D.P., "A Method for Data Translation Using the Stored Data Definition and Translation Task Group Languages," IBID.

[10] Fry, J.P., Frank, R.L. and Hershey III, E.A., "A Developmental Model for Data Translation," IBID.

[11] Sibley, E.H. and Taylor, R.W., "A Data Definition and Mapping Language," CACM, Dec. 1973, Vol. 16, No. 12.

[12] Ramirez, J.A., Rin, N.A. and Prywes, N.S., "Automatic Generation of Data Conversion Programs Using a Data Description Language," Proceedings of 1974 SIGMOD Workshop on Data Description, Access and Control, Ann Arbor, Mich., May 1-3, 1974.

[13] Lin, S. and Heller, J., "A Record Oriented, Grammar Driven Data Translation Model," Proceedings of 1974 SIGMOD Workshop on Data Description, Access and Control, Ann Arbor, Mich., May 1-3, 1974.

[14] Senko, M.E., Altman, E.B., Astrahan, M.M. and Fehder, P.L., "Data Structures and Accessing in Data Base Systems," IBM Systems Journal, Vol. 12, No. 1, 1973.

[15] Lum, V.Y., Shu, N.C., and Housel, B.C., "Data Translation. Part I: A General Methodology for Data Conversion and Restructuring". IBM Research Report RJ1525, July, 1975.

[16] Housel, B.C., Smith, D.P., Shu, N.C. and Lum, V.Y., DEFINE: A Nonprocedural Data Description Language for Defining Information Easily," Proceedings of ACM Pacific 75 Symposium, San Francisco, Calif., April 17-18, 1975. pp. 62-70.

[17] Shu, N.C., Housel, B.C. and Lum, V.Y., CONVERT: A High Level Translation Definition Language for Data Conversion," CACM, Oct. 1975, pp. 557-567.

[18] Housel, B.C., Shu, N.C., "A High-Level Manipulation and Query Language for Hierarchical Data Abstractions", Proceedings of Conference on Data: Abstraction, Definition and Structure. Salt Lake City, Utah, March 22-24, 1976.

[19] IMS/360, Version 2, System/Application Design Guide. IBM Manual SH20-0910-3, 1972.

[20] CODASYL Data Base Task Group, April 1971 Report.

REPRESENTATION OF HIERARCHICALLY STRUCTURED DATA IN THE PROPOSED ERDA EXCHANGE STANDARD

David R. Richards

Computer Science and Applied Mathematics Department

Lawrence Berkeley Laboratory

INTRODUCTION

The ERDA Inter-Laboratory Working Group on Data Exchange (IWGDE) is developing a standard modeled after ANSI Z39.2-1971 for exchange of data via magetic tape.[1] This paper proposes an extension to the IWGDE standard that makes it possible to represent an hierarchical structure relating the data elements in a record.

Hierarchical structure is natural for many applications and the most widely-used DBMS's support this type of data structure, so it is essential that an information exchange technique preserve such structure information in an efficient and completely general manner.

In the following discussion we will ignore all questions of physical representation such as tape labels, physical block structure, spanning of multiple volumes, etc., and concentrate on the logical structure of a data transmission tape. (At this level, the standard could be implemented on a different medium entirely, e.g., file transfer in a computer network.)

THEORY

The underlying idea of this proposal is that once the generic structure of an hierarchical data record has been defined, a particular example of such a record has a natural linear representation as an ordered list of data element occurrences.

To see this, consider a data base whose records have six data elements A, B, C, D, E, and F whose hierarchical relationship is shown in Figure 1.

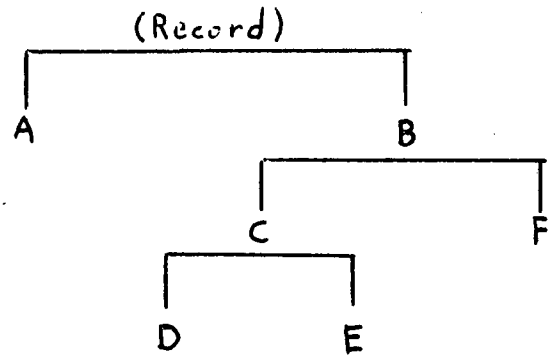


Figure 1

In a particular occurrence of this record type, any of the data elements may occur on arbitrary number of times within each occurrence of its parent data element. (Of course, if a data element does not occur at all, its subordinate data elements cannot occur either.) Figure 2 shows a particular occurrence of the record type of Figure 1.

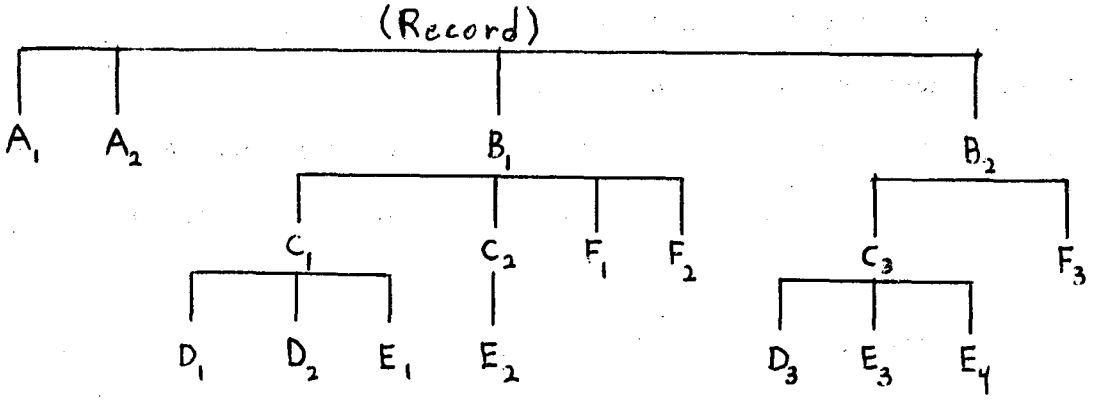


Figure 2

These data element occurrances may be written in a linear list preserving all structure information if they are ordered in a left-list or top-down-left-right sequence.[2] That is, the data element occurrences are listed in the order in which they would be encountered while traversing the tree of Figure 2 in adherence with the following recursive rule:

1. Take in turn each branch of the tree from left to right, traversing each subtree encountered according to the same rule.

Thus, the data element occurrences in the record of Figure 2 would be written:

$$A_1A_2B_1C_1D_1D_2E_1C_2E_2F_1F_2B_2C_3D_3E_3E_4F_3$$

Actually, the following less restrictive ordering rule is sufficient:

2. Each data element occurrence follows the occurrence of its parent data element to which it is linked and precedes any following occurrences of that parent data element.

A tree represented by a list ordered according to rule (1) or (2) may be reconstructed by the following rule:

3. Traverse the list from left to right, linking each data element occurrence to the last-encountered occurrence of its parent data element.

This algorithm requires only that the process implementing it know the parent-child relationships between data elements in the generic record structure.

IMPLEMENTATION

In the IWGDE standard, an exchanged data base comprises two files. The first, the data description file (DDF), provides a description of the data contained in the second, the data file (DF). The logical record format of both files is the same (the DDF has only one logical record) and is diagrammed in Figure 3.

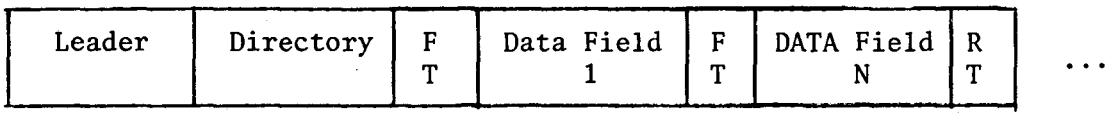


Figure 3

The leader contains control information. The directory contains an entry for each data element occurrence with its tag, field length, and starting position as shown in Figure 4.

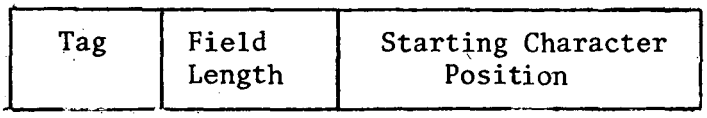


Figure 4

Consider first the DF. Hierarchically structured data can be represented simply by ordering the director entries according to rule 1 or 2. above. The actual data fields need not be ordered since they are pointed to by their corresponding directory entries. (In practice, they would probably be ordered in the same way as the directory entries.)

The present standard requires that the directory entries be ordered by increasing tag value to facilitate locating a particular tag in the directory. Hence, some means must be provided to indicate which tag ordering is being used. This could be done using one of the so-far unused characters in the DDF leader.

The DDF record is a representation of the generic DF record. That is, each possible data element tag occurs exactly once in the DDF directory while its corresponding data field contains the data element name and format description. If the records are to have hierarchical structure, the parent-child relationships must be specified in the DDF also.

One of the tags reserved by the standard for special use is...ØØØ. It occurs only in the DDF record and the corresponding data field contains the data base name. If the records have hierarchical structure, it is proposed that this field (or another reserved-tag DDF-only field) also contain a list of ordered (TAG, PARENT-TAG) pairs, where PARENT-TAG identifies the parent of the data element identified by TAG. If the data element identified by TAG is at the uppermost level in the hierarchy (record level), there is no pair for TAG in the list. (Thus, the list is completely absent for unstructured records.) This is sufficient information to define an hierarchical record structure. The DDF directory entries need not be given any particular order, but probably can be handled most easily by translation programs

if they are ordered in accordance with rule (1).

This method for representing hierarchical structure allows implementation of the standard to proceed in three stages:

Level Ø: Unstructured Data Elements

Level 1: Structured Data Elements

Level 2: Structured Records

The three levels of implementation are both upward and downward compatible. A tape written at level 1 could be read at level Ø without decoding the data element structure. Likewise, a tape written at level 2 could be read at levels Ø or 1 as if there were no record structure information present.

SUMMARY

To extend the proposed ERDA data exchange standard so that hierarchical structure information may be preserved, three changes are necessary.

1. A DDF leader character must be assigned to carry information on whether or not hierarchical structure is present.

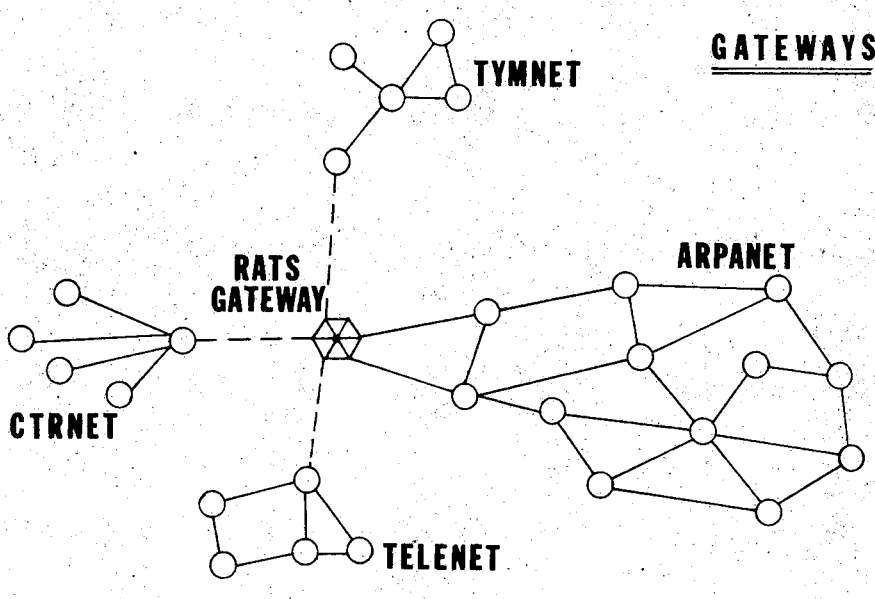
   If structure information is present,

2. The DDF field with ... ØØØ must include (TAG, PARENT-TAG) pairs for all data elements not at the topmost level of the hierarchy.
3. The directory entries in each DF record must be ordered in left-list sequence.

REFERENCES

1. A. A. Brooks, "An Extension of the ANSI Z39.2 Standard to General Information Exchange," Proceedings of the Berkeley Workshop on Distributed Data Management and Computer Networks, 1976, preceding paper.

2. James Martin, Computer Data-Base Organization, Prentice Hall, 1975, pp. 317-319.



V-C. GATEWAYS AND LOCAL NETWORKS, PART I

Transmission Systems Tradeoffs in Ring-Structured Digital Systems

Michael R. Lyle

and

David J. Farber

April 21, 1976

Abstract

This paper is a description of work in progress at the University of California, Irvine on transmission systems and their related clocking and synchronization schemes as applied to distributed computer systems. Several schemes for the transmission and synchronization of messages are discussed relative to a set of evaluation criteria for network communication and the current state of the art in implementation methodologies and technologies. The goal of this research is to implement several of these ideas in a hardware prototype network interface unit to be built for an ARPA ring structured intelligent terminal network.

Introduction

The transmission system of a computer network is that part of the network that is dedicated to performing the transfer of messages or data from one part of the network to another. When a user of the network uses the transmission system, it is more convenient if the user need not get involved in the low-level processes of the communication. The particular architecture of the network should reflect the needs of the users of the network, and the transmission system is in turn influenced by the network architecture. Our goal is to design a network

that is characterized as an inexpensive, reliable, local network of fairly inexpensive, reliable, small-scale computers and peripherals. In the course of our design, we are using a set of criteria that elaborate on the general goals stated above, and design alternatives are measured by these criteria.

Design Criteria

Some of our criteria, by no means ordered or complete, are listed below with a short discussion of their impact.

1. Reliable

   It is important that the elements of the network be able to communicate reliably so that the confidence level of the users is kept high. The amount of hardware in the communication system between any two users is a good measure of the reliability of the system (i.e. the more hardware, the less reliability).

2. Cheap

   If the goal is to connect cheap hardware, then the communication system should not be expensive relative to the cost of the elements connected. In this context, it is illustrative to think of the communication system as "overhead", and a goal of any low-budget enterprise is low overhead.

3. Few Wires

   The fewer the interconnections the better. Wires, light pipes, microwave links, et cetera are all expensive, and should therefore be used sparingly.

4. Distributed Identical Units

   This means that in general the incremental cost to add an element to the network is the same regardless of the number of elements already present. This means that the network is not limited by the number of elements, and to add

"just one more element" is not prohibitively expensive.

5. Easily Reconfigurable

If the transmission system fails, it should be possible to work around the failure until it is fixed. It should also be possible to add elements to the network with a minimum disruption.

6. Self-clocking, Regenerative Transmission

Self-clocked data are essential to prevent the skew problems and timing drift problems of transmission systems. Regenerative transmission means that every relaying node of the network should clean up the data before retransmission, thus avoiding many varieties of impedence matching and noise problems. This simplifies the node design so that it need only be designed to communicate with one other node at a time.

7. Self-checking

It is desirable for the sender of a message to check it after transmission to insure that it was sent properly. Also, it is desirable to include some redundant check information with sent messages to allow the receiver to check messages that are received.

8. Minimum "Essential" Logic At Each Node

In order to provide a reliable transmission system, it is necessary to keep the "hard core" logic to a minimum. This means that the logic necessary to receive, decode, regenerate, and retransmit a message must be limited, as the probability of a failure in a path will be the product of the number of nodes in the path and the probability of failure in a node.

9. Transmission Media Independent

The location of the nodes quite often dictates the most desirable transmission media to reach an addressed node. It is convenient if the selection of a media to communicate to a given node is not

too constrained, and is not necessarily the same for all nodes.

10. Well Defined Load Characteristics

It should be easy to calculate the load handling characteristic of the transmission system so that potential users of the network will be able to plan accordingly. In particular, it is desirable to be able to analyze the interference effect of other users of the network at peak loading times.

11. Easily Usable By Simple Devices

One of the major difficulties in accessing resources on a network is that the resource is not accessable directly, but only through some complex processor on the network that may not be reliable. It is desirable that shared resources be directly accessable, thus eliminating a potentially unreliable bottleneck. For example, common disk files should be directly accessable and present-day disk controllers are certainly adaptable to the network if a simple transmission scheme is employed.

Needless to say, this is an ambitious set of criteria, and any number of network architectures and transmission systems could be designed using the criteria. In order to fix some of the variables and define a more specific example, we will discuss a custom large scale integrated circuit (LSI) implementation, using N-channel metal-oxide semiconductor (MOS) technology. The implementation is a custom design, and although we feel strongly that the custom design is a more effective solution than the use of standard microprocessors, a discussion of these issues is beyond the scope of this paper.

While the choice of technology to implement a design is a direct result of the speed and cost goals of the design, the choice, once made, then impacts the detailed implementation in several ways. For example, the choice of LSI demands regularity of implementation structures in order to allow common "building blocks" to be used in many functional areas of the chip. Also, it is important to design a fair amount of flexibility into the chip in order to make it usable in more than

one application, either by programmabilty or by mask changes or by a mixture of both. In our case, usabililty by several flavors of mini and micro processors is desirable, along with some flexibility in protocol and address recognition functions of the chip. With this in mind, we will now discuss the transmission system itself.

Transmission Tradeoffs

In keeping with our goals of media independence and self-clocking data, the data inputs and outputs of the chip will be TTL levels with the clock integrated into the signal. It is our intent to use TTL current mode differential line drivers and receivers with twisted pair transmission lines in our initial system, but nothing precludes the use of other media such as fiber optics, microwave, etc.

Probabably the most critical factor in the design of a distributed transmission system is the choice of a clocking scheme for the data that allows flexibility and simplicity in the system. We have considered three basic schemes: master clocks, phase-locked loops and individual transmit clocks.

Using a master clock for the system is a very attractive idea from the standpoint of simplicity. A master clock implies a very simple circuit at each node to extract the clock from the data stream, and use that clock to retransmit the data and to clock out data transmitted by the node itself. Unfortunately, the signal tends to deteriorate after a number of retransmissions, and requires a "clock extender" that keeps in step wih the master clock. Also, since the master clock idea violates the goal of identical elements at each node, the obvious solution is to incorporate the repeater clocks at each node. This led us to consider using phase-locked loops.

Phase-locked loops are quite attractive for use in a distributed system. First, they allow for the inevitable variations in transmission speeds that occur in a distributed system. Second, since the system itself is a loop, one can phase-lock the entire system to an integral number of timing pulses. The

phase-locked loop approach also satisfies the identical elements goal, but has several drawbacks. The regenerated signal difficuly cited in the master clock scheme is present in the phase-locked solution also. Since phase-locked loops require several pulses to lock onto a frequency, the protocol would have to allow for extra synchronizing bits and the synchronization of the messages would have to take place for each message, unless clock pulses were on the loop at all times. Additional logic placed on a phase-locked loop allows shorter syncronization times, but this violates the requirement for a minimum of essential logic at each node. This led us to develop our current "best" implementation scheme.

If we use a digital approximation of a phase-locked loop for detection of incoming signals, in combination with indiviual crystal controlled output clocks, we can achieve our goals of identical node elements with a reasonable amount of hardware. Also, if we provide a small amount of buffer at each node, we can correct for the small error in frequency between adjacent nodes in an orderly fashion. The digital realization of a phase-locked loop is similar to the technique used in UART designs to detect bits in a data stream. Since the frequency of the incoming data is known, a multiple of that frequency can be used to detect the presence of a bit in the data stream.

For example, if we are receiving data at 1 Mhz with an imbedded clock signal, the maximum incoming frequency is 2 Mhz, and we can use a clock frequency of 6 Mhz to detect the pulses. When we detect an edge, we count to 2, sample the signal, and look for the next edge. We put the bits into a FIFO register, and when the register is half full, start outputting the data on the internal clock frequency. If the FIFO starts fillng up, we shrink the output pulses to 2 6Mhz clocks, and conversely, when the FIFO starts to go below the half full level, we elongate the pulses to 4 6Mhz clocks. This produces an output at a 2 Mhz fundamental with a fair amount of jitter, but no pulse less than 2 nor more than 4 6Mhz clocks long. Since the crystals are ordinarily within .002%, the FIFO need not be very long, and there will be very few 2 or 4 clock length pulses produced.

Summary

We feel that the design outlined above is a good tradeoff between the conflicting goals of the transmission system for a distributed system. The use of distributed, regenerative clocking allows a very easy expansion of a network with a fixed incremental cost and little of the usual concern for bus fan-out and signal deterioration. In addition, the use of MOS LSI allows a complex interface to be built at low incremental cost to enable low cost devices, such as teletypes, to directly interface to a local network.

ETHERNET: DISTRIBUTED PACKET SWITCHING* FOR LOCAL COMPUTER NETWORKS

BY ROBERT M. METCALFE AND DAVID R. BOGGS

Abstract

Ethernet is a branching broadcast communication system for carrying digital data packets among locally distributed computing stations. The packet transport mechanism provided by Ethernet has been used to build systems which can be viewed as either local computer networks or loosely coupled multiprocessors.

An Ethernet's shared communication facility, its Ether, is a passive broadcast medium with no central control. Coordination of access to the Ether for packet broadcasts is distributed among the contending transmitting stations using controlled statistical arbitration. Switching of packets to their destinations on the Ether is distributed among the receiving stations using packet address recognition.

Design principles and implementation are described based on experience with an operating Ethernet of 100 nodes along a kilometer of coaxial cable. A model for estimating performance under heavy loads and a packet protocol for error-controlled communication are included for completeness.

*(Available as a Xerox report.)

XEROX
PALO ALTO RESEARCH CENTER
3333 COYOTE HILL ROAD / PALO ALTO / CALIFORNIA 94304

PACKET RADIO NETWORK PROTOCOL STRUCTURES*

David L. Retz

Telecommunication Sciences Center
Stanford Research Institute
333 Ravenswood, Menlo Park, CA 94025
May 13, 1976

ABSTRACT

The ARPA Packet Radio network (PRnet) is an experimental packet-switched system currently being tested in the San Francisco area. The PRnet uses broadcast channels to provide a mechanism for data distribution and mobile digital communication. A structure of protocols exists to control the network and to provide a means of reliable end-to-end communication. This paper presents an overview of the PRnet structure and discusses the protocols which are used within the network.

INTRODUCTION

The PRnet [1,2,3,4] consists of an array of packet-switching nodes which are interconnected by radio communication links. As in other packet-switched networks, packets are transported node-by-node from source to destination according to addressing information contained within the packets. Communication resources are shared among the nodes as packets traverse the network.

A primary component at each node in the network is the Packet Radio Unit, or PRU. Each PRU contains a radio unit that transmits or receives packets and a microprocessor-controlled digital unit that controls the routing and flow of packets between nodes. A PRU may operate in a stand-alone fashion (as a repeater) or may be connected to a Host computer. The Host/PRU interface is the portal through which packets enter and leave the network.

---

*This work was supported by the Defense Advanced Research Projects Agency under ARPA order No. 2302-10, Contract No. DAHC15-73-C-0187.

Node connectivity in the PRnet is primarily determined by geographical separation and terrain and may vary with time as nodes move about. Monitoring of connectivity and assignment of routes is performed by processors at certain nodes, called stations. This routing information is distributed to PRU's in the network by transmission of control messages from the station processors.

Random access strategies similar to those used in the ALOHANET [5] and ETHERNET [6] allow the communication channel to be shared among the nodes. An error-detecting code transmitted with each packet identifies packets which are received in error (for example, because of packet collision). Retransmission mechanisms at various levels of protocol within the PRnet compensate for packets which fall by the wayside.

CHANNEL ACCESS PROTOCOL

The Channel Access Protocol, or CAP, provides a basic mechanism for transporting packets from source to destination in the network. This level of protocol utilizes a CAP packet header which is supplied at the source and is passed on through successive nodes in the network until the packet arrives at its destination. The program which runs in each PRU carries out functions of packet routing, hop-by-hop retransmission, radio channel access, and system monitoring.

Routing

A centralized routing algorithm which requires minimal routing table space at each node is currently used in the system. Each node is assigned a logical address by a station and is informed of a route for sending packets to the station. Logical addresses consist of a tuple {Level , Member}, where Level (i.e., hierarchy level) is equal to the number of hops between the node and the station, and Member identifies a particular node at a given level.

Packets are transmitted from one level to another as they travel towards or away from the station; the hierarchy level value is included in the CAP header and is incremented or decremented depending on the packet's direction relative to the station. The routing information contained in each packet thus consists of a list of the Member identifiers at the various levels along the way. Each PRU which receives a packet obtains the Level number and corresponding Member identifier from the

received packet header and compares the result to the logical address which has been assigned to the PRU by the station. Packets which match are delivered to the attached Host or are forwarded by radio to the next node, depending on whether additional entries exist in the packet route list; packets which do not match are discarded.

The station processor maintains a global routing table which maps destination addresses into out-going routes. Packets received by the station are "forwarded" to their destination by setting up the packet route field and transmitting the packet in the out-bound direction.

Figure 1 illustrates a typical network configuration. Logical addresses are shown at each node, with possible routing lists enclosed in brackets.

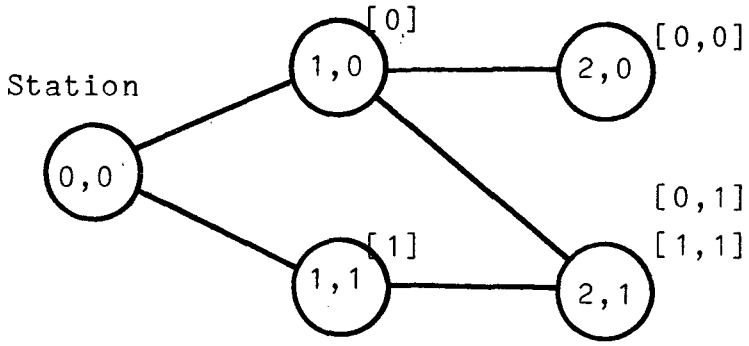


Figure 1. CAP Routing Structure

Hop-by-Hop Retransmission

A packet may be lost as it travels between nodes. This may be caused by its collision with another packet or because of ambient noise at the receiving node. In order to improve a packet's probability of success in traversing many hops, a retransmission/acknowledgement scheme is utilized at each hop. The PRU saves a copy of each packet which it transmits and awaits an acknowledgment from the next PRU along the route. This acknowledgement is implicit if the PRU hears the packet being forwarded at the next hop. The acknowledgement is explicit at the last hop, where only the packet's header is transmitted as a hop-by-hop acknowledgement. If the PRU has not heard the acknowledgement within some time interval it re-transmits the packet. Retransmission may occur a number of times until the PRU gives up and discards the packet, freeing its buffer.

Duplicate packets may be introduced in the system by the Hop-by-Hop retransmission mechanism. Some of these duplicates are detected and discarded by the PRU's (based on a unique packet header which is temporarily stored within the PRU's); others, however, slip through the duplicate filter and must be detected as they arrive at their destination.

Channel Access

There are two approaches which are used in the network for sharing of the communication channel. The ALOHA access mode (which is used in the ALOHANET) enables a PRU to transmit packets spontaneously. The retransmission time-out interval is randomized at each PRU to prevent repetitive collisions of the same packets.

The Carrier-Sense mode causes a PRU to wait until the channel is idle before transmitting by listening for absence of carrier. Carrier-sense mode significantly improves channel utilization when the network is highly connected. [7] The channel access mode used by any specific PRU in the network may be selected by control messages issued by the station processor.

System Monitoring

Each PRU in the network periodically announces its existence to the controlling nodes (the stations) by transmitting special packets, called ROP's (Radio On Packets). This function not only provides a station with information on link performance and connectivity of the known nodes in the network (i.e., the ones in the station's routing table), but also provides a mechanism for detecting that new nodes have joined the network. A ROP packet which is emitted by a new node in the network is forwarded to a station by any node within earshot. A ROP which arrives at the station contains the unique identifier of the PRU which generated it in addition to the route taken. Station routing tables may thus be dynamically updated as the network topology changes.

END-TO-END PROTOCOL

While each node in the network performs a best-effort attempt to transport packets in a hop-by-hop fashion, an end-to-end protocol is used within the PRnet to guarantee reliable delivery and to control the flow of data from source to destination. The end-to-end protocol forms a basis for reliable communication between Host processors

(and processes) which use the network and also supports reliable communication with the PRU's for network control. Packet header fields needed to implement the end-to-end protocol are included within the CAP header.

The guarantee of delivery is made possible by perseverent attempts of a sender to re-transmit a packet until it receives an end-to-end acknowledgement indicating its arrival at the destination. A side-effect of this re-transmission mechanism is the generation of duplicate packets. (Duplicates are also generated by the hop-by-hop re-transmission scheme described earlier.)

Duplicate Detection and Sequencing

Detection of duplicate and out-of-order packets arriving at the destination requires that a unique identifier be transmitted with each packet in order to determine whether a packet has been previously received. A sequence counter is maintained at the destination in order to determine whether an arriving packet is that which is expected. Packets which arrive in duplicate or out-of-order do not match the receiver's expected sequence counter and are discarded. (It is possible to buffer packets which arrive out-of-order but have sequence numbers which fall within a certain range of that which is expected.) Packets which match are treated as new packets by the receiver and cause it to advance its expected sequence number value. The transmitter, meanwhile, assigns a unique sequence number to each new packet sent. The receiver includes its expected sequence number in each acknowledgement packet it sends back to the transmitter; this provides a mechanism for duplicate and out-of-order detection of the acknowledgements.

Initialization of Sequence Numbers

A virtual connection is established between source and destination processes in the network when they agree on sequence numbers to be used. The connection is uniquely identified by a source/destination address pair. A function field included in the PRnet header is used for initialization and termination of connections. A connection is established by sending a packet which contains an initial sequence number and an "Open" function; the connection is terminated when a packet is sent (in sequence) with a "Close" function.

Once a connection is opened, a certain time interval is required to guard against duplicate copies of the "Open" packet. "Open" packets received within this

interval are discarded; after the guard interval, an "Open" packet which is received causes the connection to be resynchronized.

It is assumed that the packets which are introduced to the PRnet are expeditiously either delivered to their destination or lost along the way: the sequencing strategy places a limit on the length of time a packet may lurk in the network. The sequence number space determines the number of packets which may be uniquely identified by a sequence number. Given an upper limit on rate of packet entry to the network, this determines the maximum time a packet may remain in the system. The 16-bit sequence number field used in the end-to-end protocol, for example, assures unique identification of packets which lurk for about 160 seconds, given a packet rate of 200 packets per second.

Flow Control

An end-to-end flow control mechanism aims to control the rate at which a source node introduces packets to the network. Actual data flow from source to destination is governed by contention for resources (e.g., the channel) within the network and by the rate at which packets may be accepted and acknowledged at the destination. This information may be derived from the rate at which end-to-end acknowledgements are received at the source node. The source may adaptively adjust the rate at which it introduces packets to the network in a manner similar to Metcalfe's backoff algorithm [6]. The new-packet rate (and the end-to-end retransmission rate) is decreased with each end-to-end time-out and increased with each acknowledgement received.

SYSTEM CONTROL

The reliable communication facility provided by the end-to-end protocol makes possible the exchange of system control messages between the station processors and the PRU's; in this case the PRU itself acts as a source or destination of packets. Control functions supported by the PRU's make it possible to remotely examine or modify system parameters and to dynamically define the routing structure. Operational performance statistics which are collected within the PRU's may be periodically transmitted to a data collection facility within the network for testing and evaluation of network control algorithms. [8]

REFERENCES

1. Kahn,R.,"Organization of Computer Resources into a Packet Radio Network", Proc. AFIPS NCC May, 1975.

2. Frank, H., I. Gitman, R. Van Slyke, "Packet Radio System: Network Considerations," Proc. AFIPS NCC May, 1975.

3. Fralick, S. C. and J. C. Garrett, "Technological Considerations for Packet Radio Networks," Proc. AFIPS NCC May, 1975.

4. Burchfiel, J., R. Romlinson, and M. Beeler, "Functions and Structure of a Packet Radio Station," Proc. AFIPS NCC May, 1975.

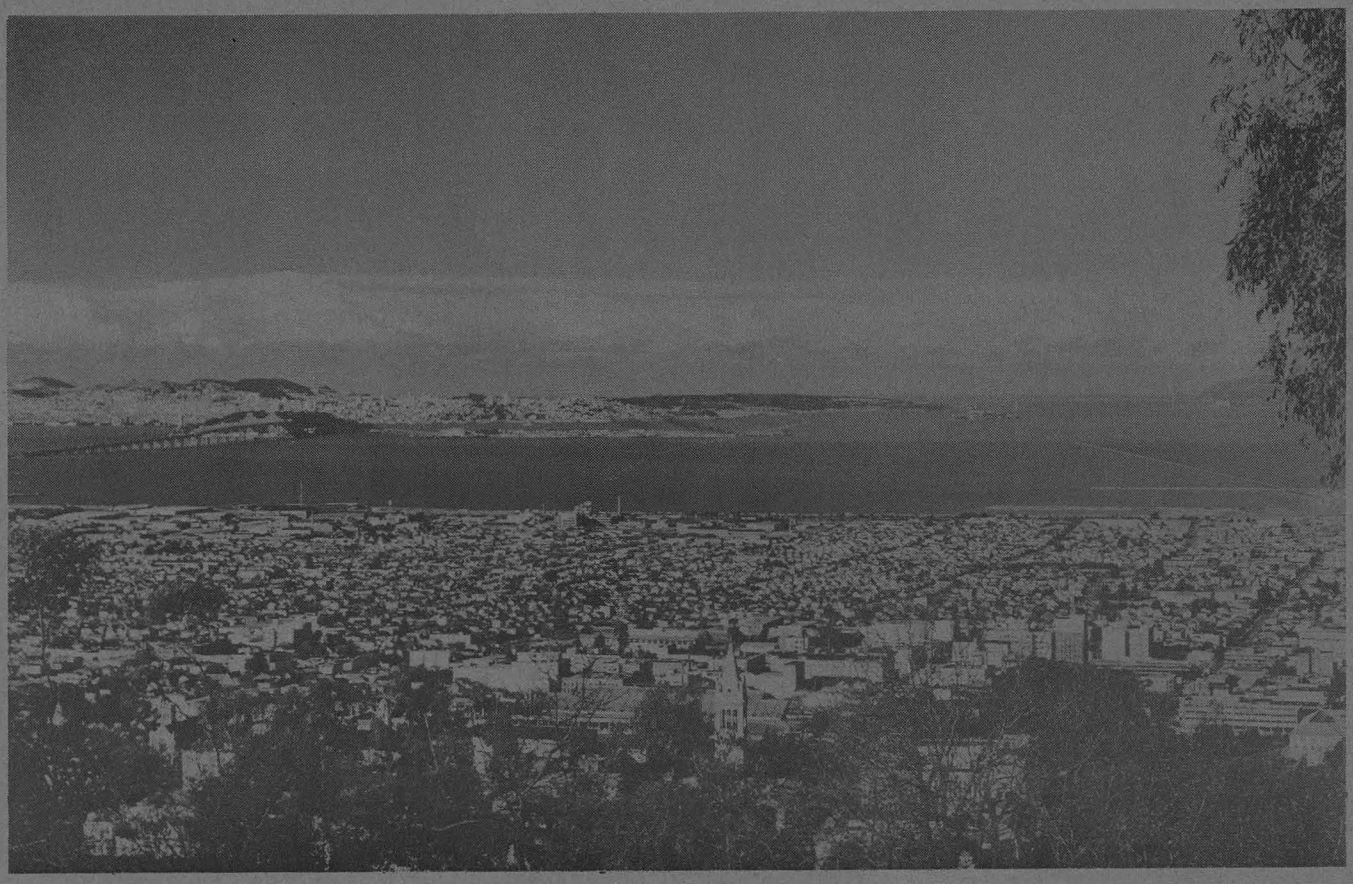
5. Binder, R., N. Abramson, F. Kuo, A. Okinaka, D. Wax, "ALOHA Packet Broadcasting - A Retrospect," Proc. AFIPS NCC May, 1975.

6. Metcalfe, R. M. and D. Boggs, "ETHERNET: Distributed Packet Switching for Local Computer Networks," Xerox Palo Alto Research Center, Report CSL 75-7 Nov, 1975.

7. Kleinrock, L. and F. Tobagi, "Random Access Techniques for Data Transmission over Packet-Switched Radio Channels," Proc. AFIPS NCC May, 1975.

8. Tobagi, F., S. Lieberson and L. Kleinrock, "On Measurement Facilities in Packet Radio Systems," to be presented at 1976 National Computer Conference, New York, June, 1976.

9. Cerf, V. and R. Kahn, "A Protocol for Packet Network Inter-communication," IEEE Transactions on Communications, Vol. COM-22, No. 5, May, 1974, pp. 637-648.



VI-A. GATEWAYS AND LOCAL NETWORKS, PART II

MICROBUS

A multiple microprocessor I/O system

May 5, 1976

Robert L. Fink, William H. Greiman and Arthur D. Turney

Lawrence Berkeley Laboratory
Physics, Computer Science and Mathematics Division
Computer Center

1.1 GENERAL INFORMATION

INTRODUCTION

With the widespread availability of microprocessors a new approach for handling communications and other low to medium speed I/O is now possible for large computer systems. This approach provides for the use of a large number of microprocessors (512 for example) by one or several computer systems in a tightly coupled environment.

GENERAL CHARACTERISTICS

The MICROBUS system will allow up to 8 host computer systems such as the CDC 6000 and 7600 to connect and control the functions of up to 512 SPU-s (Slave Processing Unit).

Each host can read or write any memory location(s) in any SPU. Also each SPU may be started, stopped, and reset to allow easy dumping and loading by the host. The host can sense the type/size/model of each SPU to allow easy allocation of multiple subprocess functions. In addition a unique MICROBUS feature allows a group of neighboring SPU-s to have certain memory locations read or written in one host transfer.

POSSIBLE USES

The MICROBUS system has the potential of handling most all of the medium and low speed I/O for a large computer center. Medium and low speed means single device transfer rates of up to the half million bit per second range. This would include virtually all types of communication lines (except possibly T1 1.5MBPS), line printers in the one to five thousand line per minute range, card readers, plotters etc. This would certainly exclude traditional tape and disk I/O as the rates are much too high (10MBPS).

A new interactive terminal system can be developed via MICROBUS that will not only allow much higher bandwidth terminals but also will support special terminal features on a terminal by terminal basis. Such a system might have up to eight 110 to 1200 BPS lines per SPU and only one higher speed line per SPU. In fact it is possible to interface existing line multiplexors to the SPU providing that the programming, bandwidth and economic limitations are reasonable.

A new batch front end system can be developed via MICROBUS that will support newer protocols and higher line speeds. For a batch

front end system only one line would be placed on an SPU.

Newer locally interfaced peripherals, in the appropriate speed range, could more easily be interfaced to the MICROBUS system. For example existing 1403 line printers could support lower case ASCII character sets by replacing older expensive controllers with more commonly available mini-computer 1403 controllers interfaced thru an SPU.

1.2 MICROPROCESSORS

DEFINITION

The use of microprocessor here refers to the computer on a single chip now widely available from many manufacturers.

TECHNOLOGY

Intel Corporation first introduced a viable microprocessor in the early 1970-s, this was the Intel 8008. The 8008 was an 8-bit computer that allowed one, two or three byte (8 bits) long instructions. The 8008 was not so much expensive ($100 range) as it was slow. For example an add from memory to the accumulator might take 30 micro-seconds. This is about 10 times slower than a PDP 8/E. The newest generation of microprocessors corrects this shortcoming of being too slow by offering speeds in the one to three micro-second range for the same add from memory to a register operation. This places the new micro-s in the speed range of most traditional mini-s.

Another failing of the early micro-s was that they were hard to interface to in that many support circuits (meaning more chips) were required just to interface memory and I/O to them. Again the new micro-s have overcome the old limitations with easy to interface buses being supplied right from the CPU chip itself. Thus many small microprocessor configurations are as simple as just connecting a CPU, memory and I/O chips together with no chips other than the system clock chip(s).

By early 1976 three microprocessor chips stood out as standards of the industry, these were the Intel 8080, the Motorola 6800 and the MOS Technology 6502. The 6502 broke a price barrier by offering a 6800 bus compatible chip for just $25 in single quantity. The 8080 broke the software barrier by initially offering good software support for the CPU, including the innovative PL/M compiler that is now becoming an industry standard. The 6800 has an extremely easy to use bus and has some very capable interfacing chips to go with it.

CHOICE OF MICROPROCESSOR

The choice of microprocessor chip for the MICROBUS system is the Motorola 6800. This choice was made from the three chips mentioned above, the 8080, 6800 and 6502. Sixteen bit processor chips such as the TI were excluded as inappropriate for byte I/O handling. The cost of 16 bit wide memory is not offset by enhanced processing features. Many eight bit processor chips (F8, PACE etc.) were excluded due to high part count or low speed performance. In early 1976 the choice certainly seems to be the three mentioned above.

Between the 8080, 6800 and 6502 the choice is based on much more marginal feature differences. The 8080 has less interrupt processing bandwidth (factor of 4) than the 6800 or the 6502. This is due to the context switch time versus the 6800/6502. Though the 8080 is more powerful in many computational problems it has less power than the 6800/6502 for random byte manipulation such as required for I/O type problems. This is due to its mode of addressing. So the 8080 was excluded.

The 6800 versus 6502 hardware choice is even more marginal. Initially the 6502 won for its simpler clocking requirement and more flexible cycle timing features. However, subsequent evaluation of the 6800 and the 6502 from the software perspective led to the choice of the 6800. In particular there are already existing cross assemblers, compilers and loaders available for the 6800 while there seem to be no plans for a 6502 compiler.

Note that the 16 bit stack pointer and index register capability of the 6800, as well as its simpler addressing modes, make it easier to generate a compiler for than the 6502. Mostly this means that a lot of the neatest instruction features of the 6502 would only be realizable at the assembly language level. This is not the reason the 6800 was chosen though, but because higher level language support will be much more available for the 6800 than the 6502.

All the above seems to be rational in April, 1976. Let us hope that it remains so as things develop in the microprocessor business, because predictable and stable it isnt.

It should be noted that the choice of chip for MICROBUS is not an unchanging one as newer chips may be incorporated in the future.

1.3 MICROBUS DESCRIPTION

MICROBUS SYSTEM

The MICROBUS system consists of host adaptors, a bus switch matrix, internal buses (often called the MICROBUS), local bus controllers and SPU-s.

Up to eight host adaptors connect to the bus switch matrix. The bus switch matrix connects host adaptors to the requested local bus controller(s). The local bus controller actually controls the SPU-s on its bus.

Again note that the terminology MICROBUS is often used to describe the internal MICROBUS system buses.

SPU DESCRIPTION

The MICROBUS system consists of up to 512 Slave Processing Units (SPU-s). Each SPU consists of a fast microprocessor (MOS Technology 6800) at least 4k bytes (8 bits per byte) of RAM (4K by 1 static RAM chips) and the necessary I/O interface for the type of use the SPU is to be put to. Each SPU also has a connection to the MICROBUS system to allow the read/write/interrupt/status operations to be done by a host.

The SPU is on a printed circuit board by itself and enough expansion space is available on the board so that memory expansion and more complex I/O interfacing may be done. The SPU board, and hence the SPU itself, assumes the SPU number the host addresses it with by the slot it is plugged into. Thus an SPU may be plugged anywhere in the system without it being restrapped or otherwise modified.

Each SPU has a status byte that describes its unique configuration so that the host may more easily identify the SPU as to subsystem and capability. The host may read this SPU status at will.

The MICROBUS interfacing of an SPU takes advantage of the two phase nature of the microprocessor. The external clock that drives the 6800 microprocessor chip is considered as two phases for the purpose of knowing when the 6800 is referencing memory. During one phase the 6800 is using the memory and during the other it isn't. Thus another source can control the SPU memory in a very transparent way by alternating phases with the 6800 — one phase for MICROBUS, one phase for the 6800.

Since it was desired to have a one micro-second per byte transfer

capability on the MICROBUS a 400 nano-sec RAM chip was chosen so that the two required memory cycles could be supplied in one micro-second. Thus in each micro-second a 6800 cycle may be done and a MICROBUS cycle may be done, each transparent to the other.

For the above reasons the MICROBUS controlling a given SPU is operated synchronously with the SPU.

MPU DESCRIPTION

A Master Processing Unit (MPU) is an SPU with a host adaptor connection to the MICROBUS system. Thus an MPU may control the SPU-s in the same fashion any host can. This allows a local (to MICROBUS) control capability that could be used for just about any purpose.

LOCAL BUS CONTROLLER DESCRIPTION

The local bus controller (LBC) controls all the SPU-s within its logic bin. There are 32 SPU-s in an SPU bin. The LBC selects the appropriate SPU logic slot (one of 32) and does the specified MICROBUS operation (read/write/interrupt/status). The LBC can sustain a one mega-byte per second transfer rate within its SPU bin.

BUS SWITCH MATRIX DESCRIPTION

The bus switch matrix (BSM) is an n by m switch array made up of multiple 8 by 1 selector logic gates. Up to n host adaptors may dynamically interconnect to m MICROBUS buses (m is likely to be 2).

The BSM will dynamically allocate MICROBUS buses on a cycle by cycle request basis so that all host adaptors will be given equal access to SPU-s and each MICROBUS bus can operate at a 1 M byte per second rate.

HOST ADAPTORS

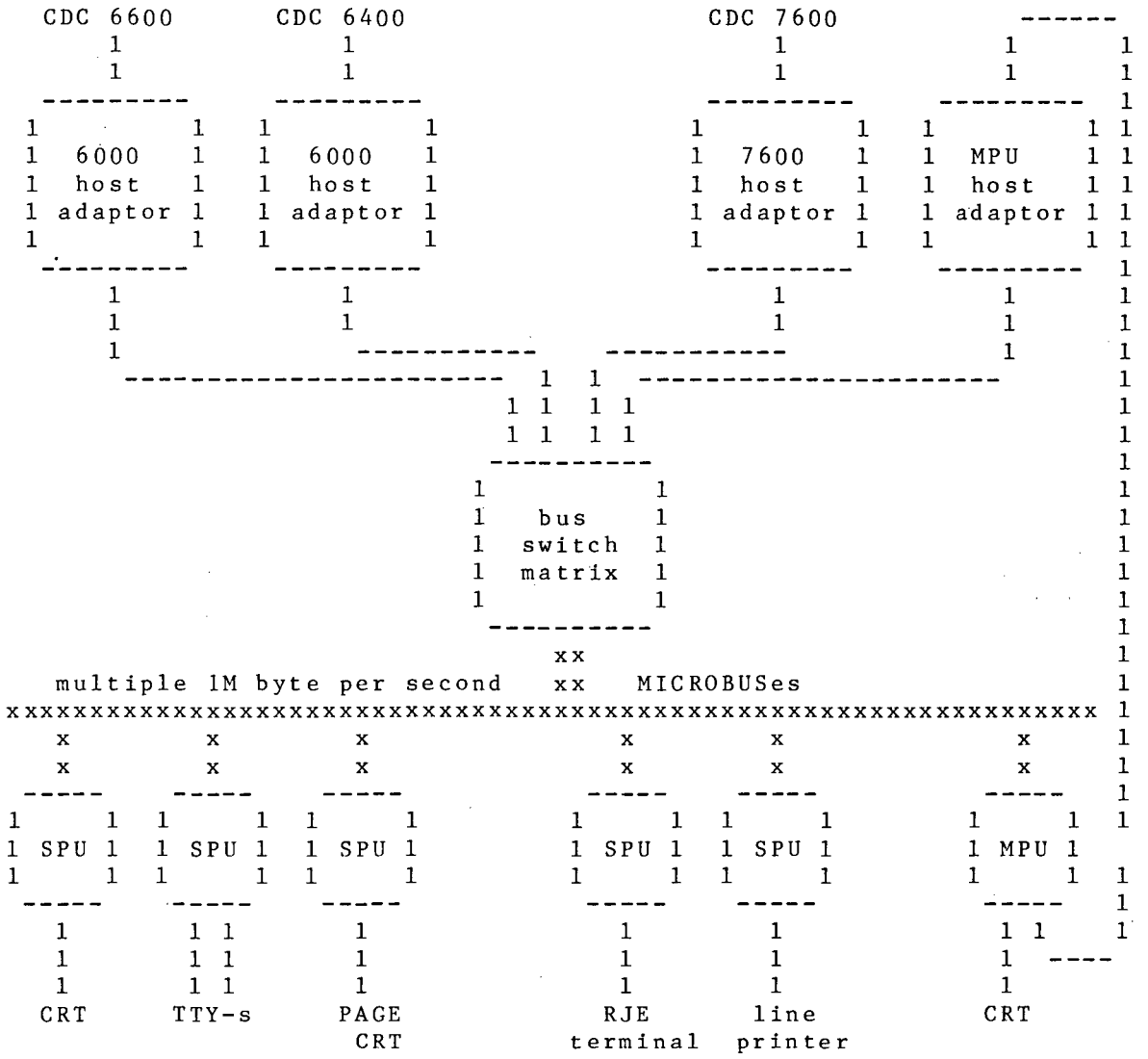
The host adaptor (HA) connects a host system I/O channel to one of the n host ports on the BSM. It is the purpose of the HA to map data and control functions from a host channel to a BSM port.

For a CDC 6000 host adaptor, PPU functions select modes such as- load address, read, write, start, stop, reset, status etc. Also 8 bit bytes are mapped through 12 bit PPU words with high order 4 bit zero fill.

See the chapter on host adaptor specifications for detailed information on how an HA works.

GENERAL MICROBUS DIAGRAM

The diagram below is only intended to give a gross structural view of the MICROBUS system and does not always represent the real structure.



2.1 MICROBUS ADDRESS SPACE

INTERRUPT VECTORS

The upper eight locations of the 6800 address space are reserved for vector interrupt addresses. These are allocated as follows-

FFFF	RESET	low part of vector address
FFFE		high part
FFFD	NMI	low part of vector address
FFFC		high part
FFFB	SWI	low part of vector address
FFFA		high part
FFF9	IRQ	low part of vector address
FFF8		high part

To implement this for the SPU the locations from 0100 thru 017F are mapped to the last 128 locations in the address space the 6800 sees. This means that any 6800 memory references to FF80 thru FFFF are going to reference real addresses 0100 thru 017F. Any 6800 references to locations 0100 thru 017F will not reference real locations 0100 thru 017F but I/O addresses or zeros (see below).

Any MICROBUS references to location 0100 thru 017F will reference real addresses 0100 thru 017F. Any MICROBUS references to location FF80 thru FFFF will get nothing (see below).

I/O ADDRESSING

I/O addressing on the 6800 is done by making control, status and data registers in the I/O device appear as addressable memory on the 6800 bus. Note that the MICROBUS cannot see such addresses, they are really local to the 6800 only. A block of 128 locations, starting at location 0100 hex, has been reserved for i/o address assignment.

The assignment of I/O addresses for asynchronous interfaces (ACIA chips) and for parallel interfaces (PIA chips) has been standardized as follows-

0100	ACIA 2	status and control registers
0101	ACIA 2	transmit and receive data registers
0102	ACIA 1	status and control registers
0103	ACIA 1	transmit and receive data registers
0104	PIA 1	data and direction registers A
0105	PIA 1	control register A
0106	PIA 1	data and direction registers B
0107	PIA 1	control register b
0108	PIA 2	data and direction registers A
0109	PIA 2	control register A
010A	PIA 2	data and direction registers B
010B	PIA 2	control register B
010C	PIA 3	data and direction registers A
010D	PIA 3	control register A
010E	PIA 3	data and direction registers B
010F	PIA 3	control register B

UNASSIGNED ADDRESS SPACE

Unassigned address space on the 6800 bus or the MICROBUS will not map back on assigned space. Thus runaway code referencing blocks of memory will not wraparound into real address space when unassigned address space is reached.

Read references to unassigned address space will yield all zeros, write references will be no operations. This applies to the 6800 and MICROBUS side of the SPU memory.

3.1 GENERAL INFORMATION

GENERAL FEATURES

The MICROBUS host adaptor (HA) connects a host I/O channel to the MICROBUS system. The HA provides a mechanism for the given host to specify the processor and address within the processor that it will operate on. The host will be able to specify the following modes-

read	-read from a given processor/address (one or more words)
read control	-read from one or more processors a given address
write	-to a given processor/address (one or more words)
write control	-write to one or more processors a given address
status	-read SPU status from a given processor(s)
start	-start a given processor
stop	-stop a given processor
nmi	-NMI interrupt a given processor
reset	-RESET interrupt a given processor

The read control and write control modes provide a way to allow a host to scan multiple SPU-s with a single block transfer. In this mode the processor number is incremented after every read/write instead of the address.

The NMI interrupt feature refers to the nonmaskable interrupt of the 6800. Note that this type of interrupt is not acceptable for use to interrupt an SPU in a running environment. This is because multiple hosts could cause interrupts to occur while an SPU is processing an NMI interrupt. The intended use for the NMI interrupt is to aid in dumping/loading. To supply the capability for a normal maskable interrupt would take extra chip hardware and is not the philosophical approach that is even recommended. Long experience with minicomputer front ends has taught that host generated interrupts are more trouble than they are worth. In fact, with the high level ability a host has to prowl around in an SPU memory without SPU interaction, the need is not there for communication to rely on an interrupt at all. Polling by the SPU when convenient is by far the best approach.

4.1 SUPPORT SOFTWARE

GENERAL INFORMATION

At this time only feasibility study application software has been written for MICROBUS, however several non related projects using the 6800 have had support software written and some application code is underway for those. The entire microprocessor software support issue is actively under study by the Real Time Systems Group in the Support Division and the Computer Center in the Physics Division. Whenever common goals can be identified it is hoped that a shared support software effort will result.

Part of the design concept of MICROBUS is that single and/or simple processes required for medium and low speed I/O may be isolated and assigned to a unique processor (an SPU). This then will result in the ability to attain the highest productivity for software people by allowing straightforward processes to be written in a high level language with a minimum of complication.

Process isolation at the processor level also alows simpler development, debug, installation and maintainance of software.

CROSS ASSEMBLER

At this time the only assembler support for the 6800 is the COMPASS assembler thru use of macros and a post processor. For simple small non relocatable programs this is sufficient. One advantage of the COMPASS assembler being used in this mode for MICROBUS is that it is well maintained for BKY systems development and is totally familiar to all prospective MICROBUS programmers.

Disadvantages of the COMPASS cross assembler are the problems in generating 16 bit addresses in 12 bit fields, the classic n pass code generation problem associated with short forms of relative jumps as they go out of range and the lack of relocatable output. It is desired to correct these problems with a better cross assembler when the cross compiler issue is resolved.

ABSOLUTE LOADER

An absolute code module loader has been developed for use by stand alone 6800 systems so that code may be down loaded via a RECC port (TTY port on the 6000/7600 system). This capability is supported thru

the debugger discussed below.

DEBUGGER

The original 6800 ROM resident debugger has been replaced by a BKY version that offers many more features, including the use of high speed CRT-s. Downloading from the Computer Center may be accomplished quickly.

CROSS COMPILER

At this time no cross compiler actually exists at LBL for the 6800. However, it is hoped that the WINTEK PL/W cross compiler will be viable. The RTSG people are evaluating this and BKY Systems people are participating.

One current feeling in RTSG is that even if the WINTEK cross compiler does not work out that the generalized first pass XPL nature of PL/M cross compilers in general may make it very easy to develop one that generates 6800 object code.

RELOCATABLE LOADER

When the cross compiler and assembler issues are resolved it is hoped that they will include a good relocatable loader to handle both the assembler and compiler object output, and have a generalized enough format to include different microprocessors. This loader would really be a cross loader in that it would run in a large machine support environment along with the cross assembler and compiler. A large machine in this context may mean the Computer Center but really means not in the microprocessor.

OK - SESAME

THE ARGONNE INTRA-LABORATORY NETWORK

William P. Lidinsky*

Argonne National Laboratory
Argonne, Illinois

Abstract

A substantial need exists at Argonne National Laboratory for a computer network within the Laboratory which is capable of providing high speed error free information transfer.

Presently there exist at Argonne 60 to 80 computers or computer based systems from a variety of manufacturers and ranging in size from naked minis to the very large machines of the Central Computing Facility (CCF). Many of these machines are used in conjunction with experiments for real time control and data logging and to a lesser extent for data reduction and as local time sharing or software development systems. Almost all of these computer based systems could benefit substantially from the ability to transfer information between each other, the CCF, and/or a central file system. In a number of cases, the need is critical.

To respond to this need the Applied Mathematics Division (AMD) is developing the Argonne Intra-Laboratory Network. As presently conceived, this network will provide a high speed transmission capability between user nodes located throughout the Laboratory. In addition to these user nodes, two server nodes are explicitly planned: one to the CCF via a multi-computer CCF front-end system and one to a central file system.

The network is being implemented as a packet transmission, distributed intelligence system capable of supporting random topology. Primary emphasis will be on task communication between different hosts. Host task to network protocols are being kept simple in order to minimize impact on the user. Links of the network will consist of dedicated Laboratory owned wire, radio, and infra-red carrier facilities. Each node will consist of a multi-microprocessor array connected to a common memory. The reasonably clean separation between network management, network line, and user protocols lends itself to multiprocessing. This fact and the power of today's microprocessors along with the projected price reductions has resulted in an extremely cost effective node architecture.

---

*Work performed under the auspices of the U.S. Energy Research and Development Administration.

INTRODUCTION

Presently there exist at Argonne National Laboratory upwards of 70 computers or computer based systems from a variety of manufacturers and ranging in size from naked minis to the very large machines of the Central Computing Facility (CCF). Many of the small computers and systems are used in conjunction with experiments for real time control and data logging, and to a lesser extent for data reduction and as local time sharing or software development systems. Almost all of these computer based systems could benefit substantially from the ability to transfer information between each other, the CCF, and/or a central file system. In a number of cases, the need is critical.

To respond to this need the Applied Mathematics Division (AMD) is developing the Argonne Intra-Laboratory Network. As presently conceived, this network will provide a high speed transmission capability between nodes located throughout the Laboratory. As a part of the network, two server nodes are explicitly planned: one to the CCF via a multi-computer CCF front-end system and one to a central file system.

The network is being implemented as a packet transmission, distributed control system capable of supporting random topology. Primary emphasis will be on task communication between different hosts. Host task to network protocols are being kept simple in order to minimize impact on the user. Links of the network will consist of dedicated Laboratory owned wire, radio, and infra-red carrier facilities.

BACKGROUND

Argonne National Laboratory is located on a 1700 acre site southwest of Chicago. Organizationally, the Laboratory consists of upwards of 20 discipline, programmatic, and service oriented divisions which are scattered in separate buildings around the site and are as much as a mile apart. Within these divisions

there are presently over 70 computers of various sizes and types. The largest of these exist within the ANL/CCF and consist of an IBM S/370/195 which handles batch computing and an IBM S/360/75 which supports a TSO time sharing capability. The smallest are naked minis and microprocessors. The number and variety of computers grows almost daily.

Many of the micros and minis are associated with experiments or projects where they are used for real time control and data logging. Sometimes these machines also do a limited amount of data reduction. Emerging uses include minicomputer systems for information retrieval and for time sharing local to a particular project or division.

In a large number of cases, additional computing power is needed with the need varying from project to project. Often further reduction or data base updating is required. This is now accomplished by physically carrying magnetic tapes, paper tapes, and/or cards to the CCF and submitting a batch job. The frequency ranges from several times a day to several times a month. There is also a growing need for "quick look" capability. Here an experiment is started, a limited amount of data is taken, data reduction is performed within perhaps one hour, and the results are used to further tune the experiment. This reduction process often needs the computing power of the CCF. Common file capability is also needed as many projects find themselves using similar software and data bases. Presently, a few cooperative efforts exist where experimental data which changes on a daily basis are being used by two groups for different purposes. Program development on one machine for execution on a similar machine at another location is also beginning to occur.

There is then a clear (and in some cases critical) need for the various computers scattered throughout the Laboratory to communicate with each other, the Central Computing Facility, and a central file system. The Argonne Intra-Laboratory Network is being developed in order to fill these needs. The network's initial stimulus was from experimentalists and was to provide a laboratory-wide, high speed, easily used data network which will connect various computers at the laboratory with each other, the CCF, and a central file system. It will also provide a means whereby any location on the Laboratory site can access networks (such as ARPANET) which are outside of Argonne.

EXISTING NETWORKS AND COMMERCIAL OFFERINGS

A large number of existing and proposed networks have been studied and surveys have been made of current or proposed commercial systems and protocols. This was done not only for pedagogical reasons but also to determine the existence of systems which might be brought into Argonne. If one could import or buy a system (especially one compatible with the existing CCF, front

end, or remote user equipment), the effort and overall elapsed time necessary to develop a system would be reduced, and at least a degree of compatibility with any other networks and/or commercial equipment and software could be maintained.

While this study did not yield an off-the-shelf system which would satisfy the laboratory's needs, it did cause us to take a long look at IBM's offerings (SNA, LABS7, DIS) and DEC's DECNET. IBM was considered because Argonne's CCF is almost totally an IBM shop. DEC was considered because of the growing number of PDP-11's in use at Argonne.

Internal to the CCF, memory requirements for DIS are large and dynamic file allocation does not exist. External to the CCF, IBM computers such as System 7's would be needed for every node (which would be limited to a star topology). LABS7 also has disadvantages similar to DIS. In addition, it is not a standard IBM product and consequently is not well supported. SNA was not well defined and appeared to be oriented toward the commercial market. None of IBM's systems were found to be cost effective for our application.

Off-the-shelf DECNET is cost ineffectiveness since DECNET will only run under DEC's disc based operating systems. Another disadvantage involves its possible inefficiency with respect to line utilization. However, DECNET's NSP (Network Services Protocol) has been selected as the protocol basis for network management.

SWIFTE

The initial phase of the Argonne Intra-Laboratory Network is the development of SWIFTE, an experimental system for wideband information transfer. It consists of a single link between a PDP-11/45 in the Physics Division and the communications front-end of the Central Computing Facility located in the Applied Mathematics Division.

As previously mentioned, DECNET's NSP will be used as a basis for SWIFTE's network management protocol. With respect to line handling, original plans for the link specified a synchronous bit-oriented protocol (e.g. SDLC or HDLC) since both national and international standards will likely be of this type. However, recent events within and among CCITT, ISO, and ANSI lead one to believe that meaningful standards will not be soon in coming and that, when they are finally adopted, compatibility will still be a problem due to the latitude within the standards. Consequently, the line protocol question is presently being reconsidered with less emphasis on standards and more on immediate needs.

Figure 1 is a block diagram of SWIFTE. The CCF front-end is an array of three closely coupled Varian V73's which serve as a front end to the CCF. It is described in a separate paper in these proceedings (Amiot, L. W., "Front-Ending at Argonne National Laboratory"). The PDP-11/45 is one of several PDP-11's in the Physics Division. It presently (and for the foreseeable future) runs single user DOS.

The line bandwidth and configuration for both SWIFTE and the full network is full duplex 50 Kb. The SWIFTE line is shielded twisted pair and is about 700 feet long.

Each of the two SWIFTE nodes is implemented with an IPU (Interface Processing Unit) (Refer to Figure 2.). The Western Digital MCP-1600 executes the line handler software. The TI9900 executes both the host interface software and the network management software. The 16K memory is a standard single port dynamic MOS memory.

The goal of SWIFTE is to act as a test bed for the full Intra-Laboratory Network. Initially, the following functions will be implemented:

1) file transfer and update between PDP-11 and front-end disc;

2) remote batch from PDP-11 to the CCF;

3) file transfer between PDP-11 and the CCF (as OS datasets).

INTRA-LABORATORY NETWORK

Figure 3 is a block diagram typical of the full Intra-Laboratory Network. The links will consist primarily of laboratory owned wire but may also include infrared and radio transmission media. The nodes will be implemented with Interface Processing Units. The final design of these nodes and their software will emerge from the work done on SWIFTE.

Three of these nodes and their associated hosts are of particular interest. The file node is necessary for all those user needs associated with storage and common data bases and also for off-loading data transfer operations from the CCF where possible. The CCF and network-network interface nodes actually both go through the front-end system although they are shown here as separate hosts. The need for separate IPU's is unclear at this time.

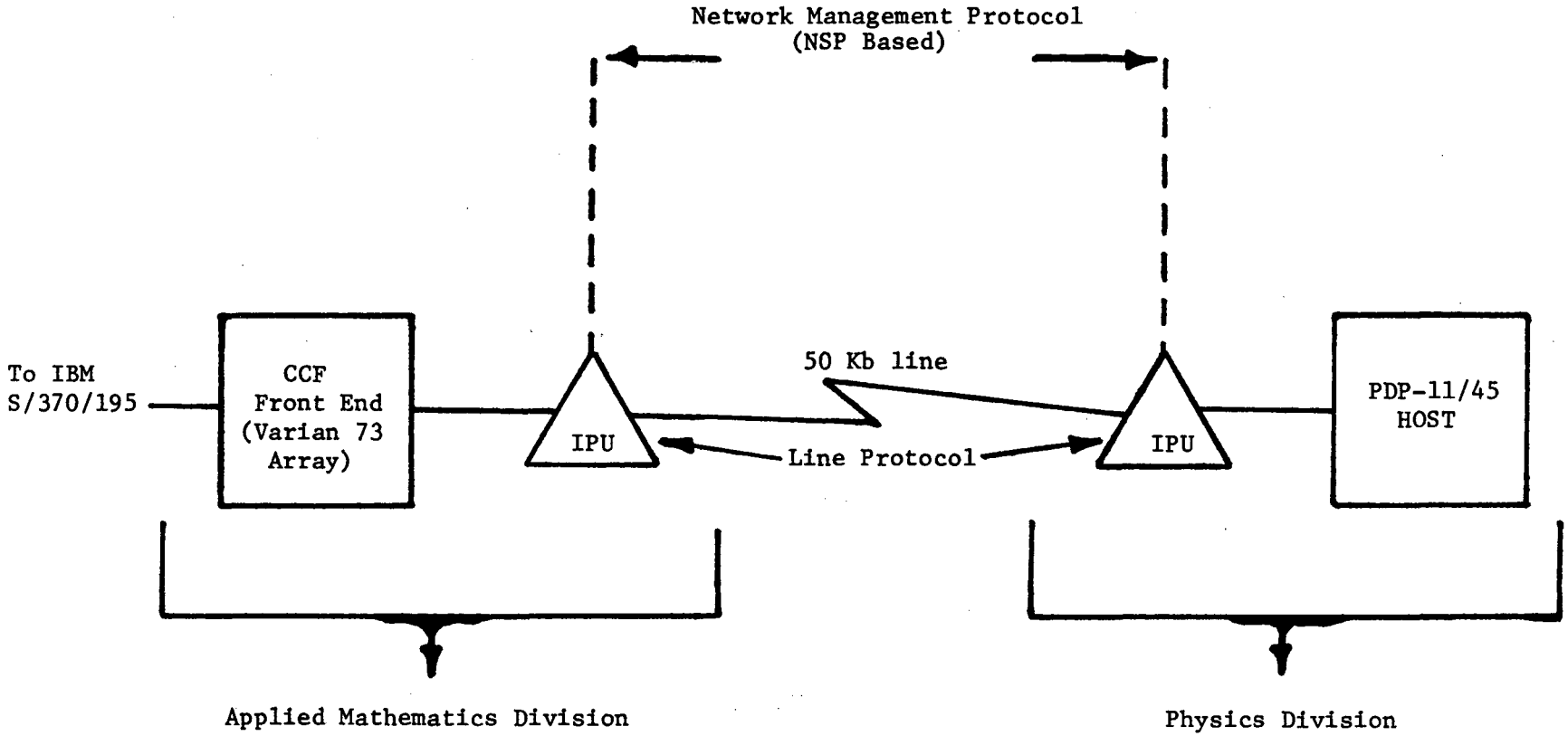


Fig. 1: SWIFTE EXPERIMENTAL LINK

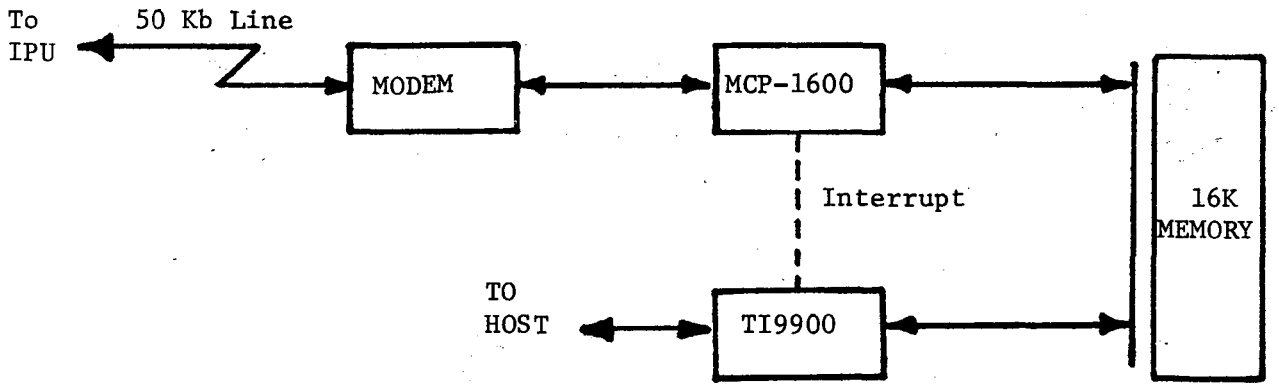


Fig. 2: SWIFTE INTERFACE PROCESSING UNIT (IPU)

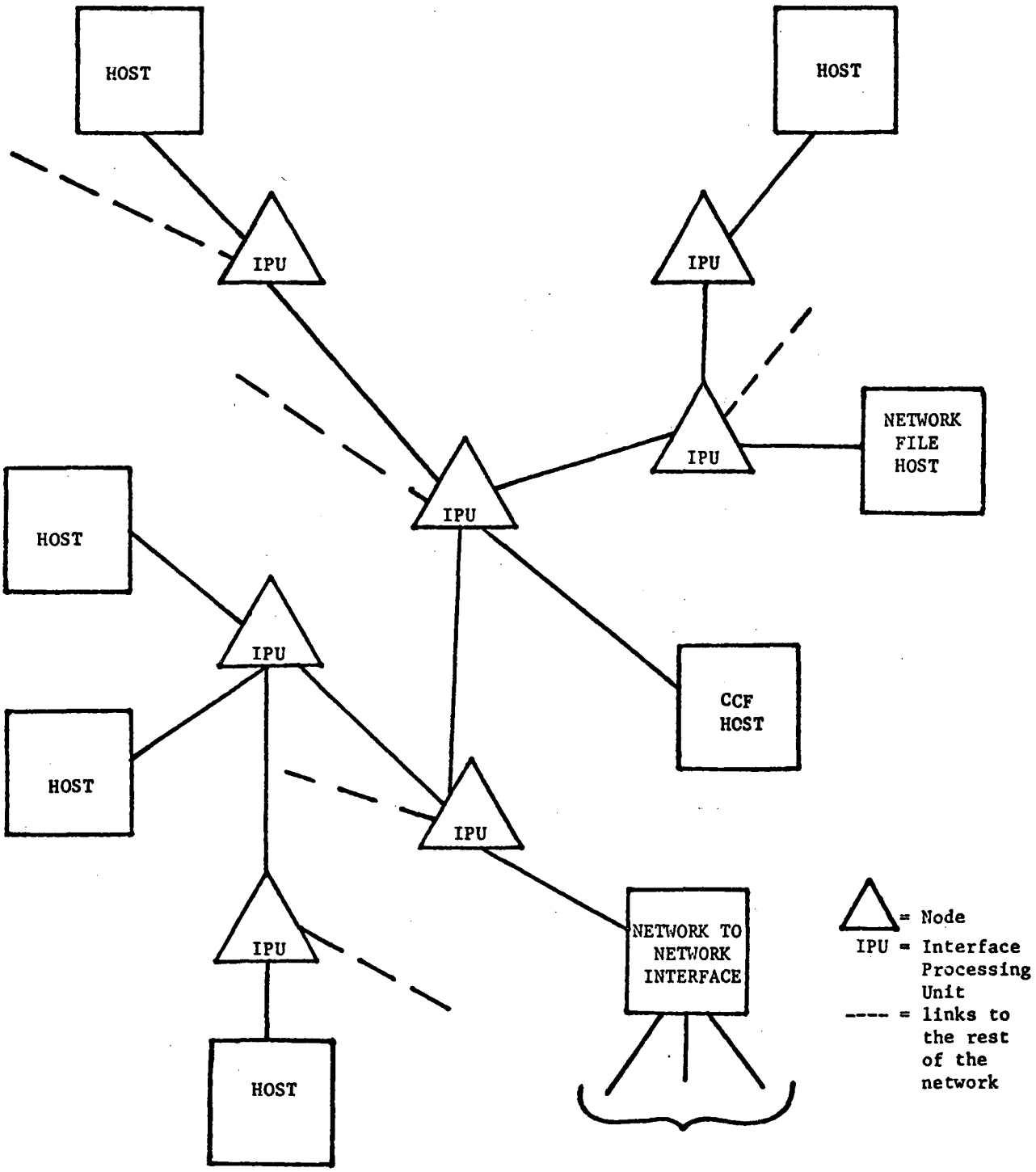


Fig. 3: ARGONNE INTRA-LABORATORY NETWORK

Interface Processing Unit

According to present plans, each node on the Intra-Laboratory Network will consist of an IPU and its corresponding host. The IPUs control the network in a distributed fashion performing three basic functions:

1) host interfacing with the goal of minimizing the network loading of the host;

2) link management;

3) node management.

For any single link of the network the relation between these functions and the tasks which execute them are shown in Figure 4.

Due to the modularity of these tasks, the IPU's are implemented as multi-processors connected via a common high performance multi-port memory and a low bandwidth interrupt mechanism as shown in Figure 5. The line processor executes the line handler software for up to four lines. The network management processor executes the network management software as in SWIFTE. The host-network interface may be handled by a separate processor. Experience with SWIFTE will determine this.

Ideally, all of the processors should be the same for the sake of implementation. However, specialized requirements of the tasks preclude this. The network management processor needs to execute complex logic but not in real time. Therefore, a microprocessor with a rich instruction set is desired; speed is a secondary consideration. The converse is true for the line processor. Here the real time considerations require a fast machine. However, the logic to be performed is less complex. This has led to the choice of the TI9900 and the Western Digital MCP-1600 respectively as discussed in the section on SWIFTE.

The key to the performance of the IPU, however, is in the multi-port memory. SWIFTE will not have such a memory due to the constraints of development time. Instead, a commercial off-the-shelf memory is used. However, with multiple lines and perhaps multiple hosts per IPU, a single port memory would soon become a bottleneck. Ideally, the memory should be multi-ported, interleaved, and/or block structured.

Packet Transmission

Presently, three different point to point transmission technologies exist: circuit switching, message switching, and packet switching. Variants on these are possible. In considering the transmission technology which would best serve the Argonne Intra-Laboratory Network, the following facts emerged:

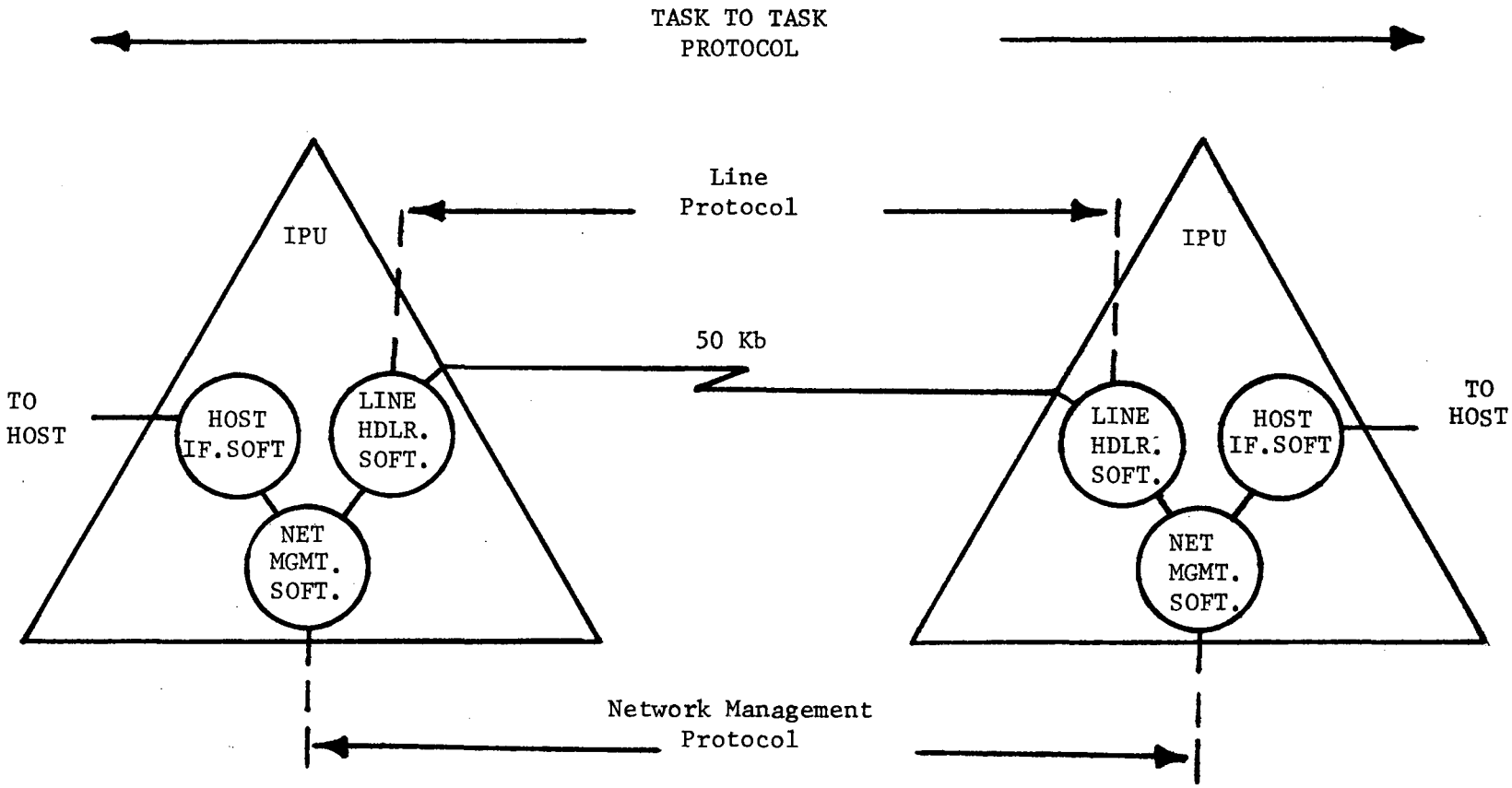


Fig. 4: INTRA-LABORATORY NETWORK - RELATION BETWEEN PROTOCOLS AND SOFTWARE

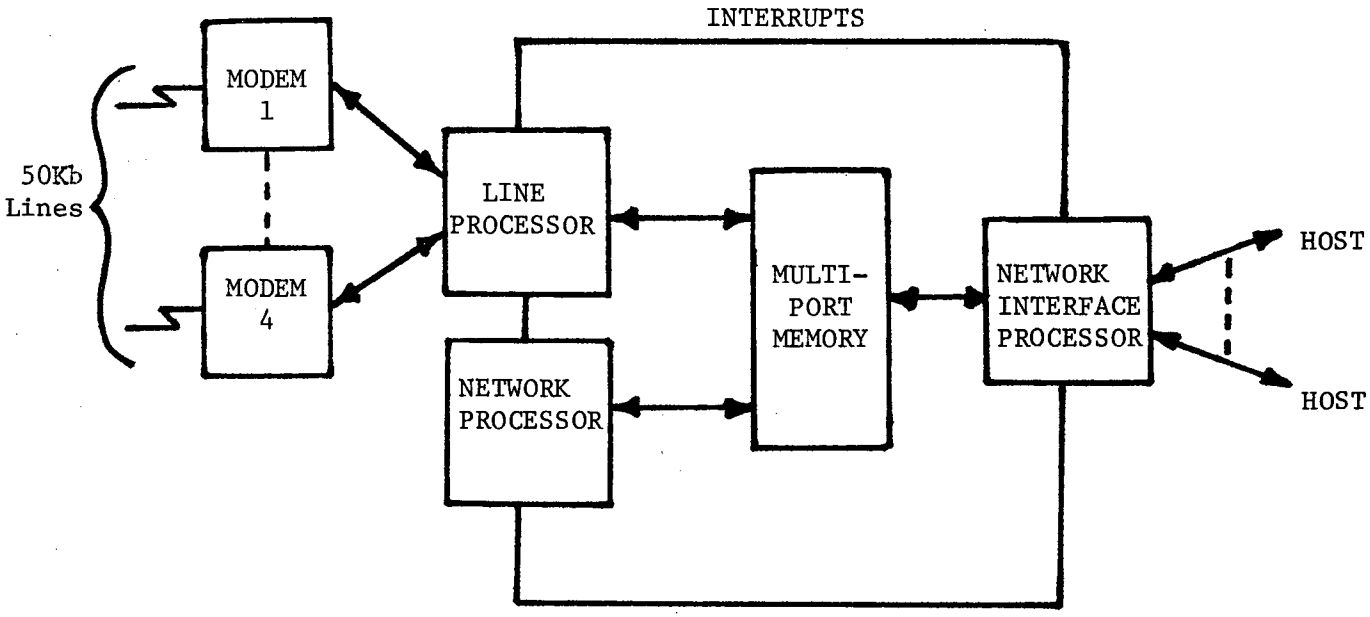


Fig. 5: INTERFACE PROCESSING UNIT

1) The network will have a random topology.

2) The number of intermediate nodes for any transmission will be small (typically zero or one and probably never more than three).

3) Alternate paths will often (but not always) exist. They are highly desirable, but because of the short haul lines two alternatives are sufficient.

4) The amount of data to be transmitted from point to point may range from a few hundred bits to several million bits. The latter case will be somewhat more frequent.

Consequently, what is needed is a network capable of (1) fast response to short transmissions, and (2) high throughput for long transmissions. These are to be implemented within the framework of short haul relatively low error lines with path alternatives limited to, at most, two routes.

Packet switching is the only technology of the above three which can accommodate this range of transmission demands. Full blown packet switching however, implies the implementation of sophisticated information gathering and routing algorithms. In Argonne's case, increasing speed for consistently long transmissions can be solved, for instance, by adding a direct link. This would neither seriously impact our implementation of the network nor be overly costly to effect. Consequently, concurrent packet transmissions or resource reallocation isn't necessary. If a complete packet switching algorithm is determined desirable at a later date because of network evolution, it can be added within the present framework.

Therefore, the Intra-Laboratory Network plans to employ "packet transmission" rather than packet switching in the ARPA sense. Logical links will be established for each transmission and maintained for the duration of that transmission. Error recovery will be at the packet level between nodes and at the message level between sending and receiving hosts.

SUMMARY AND DISCUSSION

The Argonne Intra-Laboratory Network is being developed in two stages. First, an experimental single link called SWIFTE is being developed for overall experimentation and testing. It will also provide a demonstration system where existing users will be able to furnish feedback which can easily effect changes, and where potential users will be able to more accurately evaluate their own needs and plans.

The network is being designed as a packet transmission, distributed control system capable of supporting random topology. Here packet transmission implies fixed routing with only sequential packets for a given transmission.

Primary user emphasis will be on task communication between different hosts with host-network protocols kept as simple as possible in order to achieve network transparency. Links of the network will, for the most part, consist of laboratory owned wire. However, some radio and infrared facilities may also exist.

The nodes are being designed as multi-microprocessor arrays connected to a common high performance memory. The reasonably clean separation between node management, line, and host interface protocols lends itself to multi-processing. This fact and the power of today's microprocessors along with the projected price reductions has led to a cost effective node architecture.

ALTERNATIVES FOR COMPUTER NETWORK INTERCONNECTION*

Carl A. Sunshine
The Rand Corporation

1. INTRODUCTION

As computer networks providing various types of service, geographical coverage, or organizational coverage proliferate, the desirability of interconnecting networks will inevitably increase. While the technology of individual networks is well advanced with many networks already in operation, the problems of network interconnection are just beginning to receive attention and very few actual interconnections exist. This short position paper discusses several important problems involved in interconnection of packet switching networks (PSN) including selecting a local net interface level and service level, maintaining local net autonomy, and Hop-by-Hop versus Endpoint implementation approaches. Further attention to these and other issues such as internet routing and addressing and Gateway functions may be found in [Sunshine76b] along with a more complete list of references.

In considering alternative approaches to network interconnection, a tension often exists between local network independence and global standards. On the one side, each local net's address space, routing algorithms, packet formats, protocols, traffic controls, fees, and other characteristics should be preserved to the greatest extent possible. This minimizes the disruption of normal operations and the cost of interconnection with other nets, while maximizing each network's autonomy, an important political consideration. On the other hand, global agreements on addressing techniques, message formats, and protocols allow improved operations as discussed below. In many cases, such standards can be implemented on top of existing local net functions, achieving most of the benefits of standardization and autonomy at the same time.

The primary function of an interface or "Gateway" between networks is to forward traffic from one net to another, translating between different local net formats and protocols as required. A Gateway may be an actual processor connected to both networks, or may be additional software implemented in existing processors on one or both nets. In either case, a suitable interface between the Gateway and each local net connected must be determined.

Some confusion has resulted from past discussions of this issue [Pouzin73, Pouzin74, Lloyd75, Walden75] because the commonly presented alternatives of "Host" or "packet switch" level interconnec-

*This work was supported in part by the Defense Advanced Research Projects Agency under ARPA Order No. 2494, Contract No. MDA903-76C-0093 with Stanford University. The views and conclusions contained in this document are those of the author and do not necessarily represent the policies of the Defense Advanced Research Projects Agency of the United States Government.

tion have different meanings for different authors. Three distinct concepts are involved in this issue, each of which will be discussed in turn.

Local Net Interface Level

A Gateway can interface to local nets either as a packet switch (employs PS-PS protocol to communicate with local net) or as a Host (employs Host-PS protocol to communicate with local net). This is the most common meaning of interconnection level. In the packet switch case, a Gateway must behave like a normal switching and forwarding node in the local net, while in the Host case, a Gateway behaves like a source/destination of local net packets.

Local Net Service Level

If a Host level local net interface is chosen, a Gateway can make use of various local net service levels for transmitting packets through a local net. Levels include a simple "datagram" or best effort service, and a more reliable "virtual call" service. Other special service levels such as bulk data transfer or interactive terminal handling may also be available.

Endpoint vs. Hop-by-Hop Protocol Implementation

A desired end-end service may be implemented two ways. The Endpoint or "stepwise" [Pouzin75] approach consists of implementing suitable control algorithms at each end of the communication path, while requiring minimal service levels on each hop. The Hop-by-Hop approach provides the desired end-end service without additional end-end protocol by requiring the desired service level on each hop (each local net), and joining the hops together with any necessary translation performed in the Gateways.

2. LOCAL NET INTERFACE LEVEL

Interconnecting networks through a packet switch level interface (PSLI) may appear to be the simplest strategy since the resulting "Catenet" [Pouzin73] appears to be one large network without any complicating hierarchical structure. In fact, PSLI is fraught with difficulties and has not yet been demonstrated to be feasible.

The main difficulty of PSLI stems from widely varying PS-PS protocols in different nets. A Gateway must somehow make the rest of the Catenet look like an extension of the local net it serves. The problem of mapping adjacent net PS-PS protocols can be considered in two parts since a PS-PS protocol involves some functions every node performs, and other functions only performed by source and destination nodes.

Universal node functions include error detection, retransmission, duplicate removal, and routing. Some of these may be easily carried across networks, while others are more difficult. For example, some nets may not perform node-node duplicate filtering, while

others may rely on it to avoid delivering duplicates to end users. In nets that exchange routing data with adjacent nodes or a central authority, the Gateway/packet switch will have to generate appropriate routing data for the rest of the Catenet, or actually translate routing data between local nets. Worse problems arise with specialized node functions such as packet tracing, remote debugging, statistics gathering, and call charging. Gateway/packet switches would have to isolate requests for and responses from these special services to individual nets.

Even if difficulties with universal node functions could be overcome, many local nets provide extensive additional functions at source and destination packet switch nodes. EPSS [Bright75] requires a complicated call set-up, buffer allocation, flow control, and end-end acknowledgement scheme to be enforced by end nodes. ARPANET IMPs [Heart70] perform similar storage reservation, RFNM generation, message reassembly, and message sequencing functions. Many nets also perform accounting or access control functions in end nodes. To interconnect such networks at the packet switch level would require all the appropriate fields and responses to be generated at the final destination, or at the Gateway (in which case the Gateway is behaving like an endpoint of the communication path, or a Host, not a simple packet switch).

The only alternatives for PSLI seem to be requiring identical PS-PS protocols in all nets, implementing all protocols for all nets in every PS, or "masking out" all but the common subset of basic functions as packets traverse a Gateway. None of these seem satisfactory.

Interfacing a Gateway to a local net as a Host, on the other hand, facilitates net independence since local net packet switches need not know other net protocols. Each local net protocol "stops" at the Gateway which serves to cauterize local net idiosyncracies. All internet functions are implemented in Hosts and Gateways on top of the local net transmission services. Local nets have greater control over traffic entering from other nets since internet traffic enters from a Host. Host-PS protocol implementations typically exist for a wide range of machines, providing a headstart and a wide choice for Gateway implementation, while packet switch implementations typically exist only for a single special purpose machine.

3. LOCAL NET SERVICE LEVEL

Given that Gateways are interfaced to local nets as Hosts, several levels of service are typically available for transmitting internet packets through each local net, including datagram, virtual call, bulk data transfer, and interactive terminal services [MacPherson75, Pouzin75, Crocker72]. These services may be provided by the PSN itself, or by additional protocols implemented in the Host computers. In general, more powerful services require more complex protocols in the Host (or PSN), and more control fields in each packet transmitted.

When additional functions are desirable between Gateways, the availablity of higher level services may be an advantage. Where minimal services are required from a local net but only higher level

services are available, internet performance may suffer from the increased overhead [Kleinrock74] as discussed in the following section.

4. ENDPOINT VS. HOP-BY-HOP PROTOCOL IMPLEMENTATION

Ideally, the same spectrum of communication services should be available between users on different nets as those described above for local nets. The basic choice for implementing various end-end services is between Endpoint and Hop-by-Hop approaches. The Endpoint approach requires a common end-end communication control protocol (CCP) such as TCP [Cerf74a, Cerf74b], [Cerf75], or [Zimmerman75] to be used by all participants, but minimal services within each local net. The Hop-by-Hop approach uses existing protocols in each local net to provide the service level desired, but may require a complicated translation between protocols within the Gateway. The following sections compare the implementation of an end-end virtual call level service by both of these approaches.

Endpoint Implementation (see figure 1)

The internet TCP's at each end of a virtual call produce internet packets which must be transmitted through intervening local nets and Gateways. Several local net service levels may be available for this purpose, and one must be selected. Sequencing in each local net is undesirable since it will be performed at the destination, and increases delay in each hop [Sunshine76a] as well as requiring a single exit Gateway from the local net for all packets of a connection. Hop-by-Hop error correction is more efficient than Endpoint [Metcalfe73], so that where error rates are high (as in PRNET [Kahn75]), local retransmission is desirable. For local nets with low error rates, the efficiency difference is small, while the saving in protocol complexity and Gateway buffering from eliminating retransmission on each hop may be substantial. Finally, some form of flow control between Gateways may be necessary, although network capacity may adequately limit traffic levels in some local nets.

In most networks, internet packets will arrive at a Gateway via a relatively simple local net datagram protocol. Packets arriving at a Gateway need not be sequenced before forwarding. The Gateway is free to send different packets from a single connection over different internet routes since the Endpoint TCP will sequence them. If a local route fails during the course of a connection, alternate routes may be used without causing end-end errors.

With an Endpoint approach, the same Gateway implementation and local net services can be used to provide different end-end internet services by using different protocols in the Hosts at each end. For example bulk data transfer or a different virtual call protocol could be implemented in internet Hosts with no change to Gateways.

The portion of a Gateway associated with each local net may be called a Gateway "half", and closely resembles the structure required in an internet Host. Both must contain a local net interface, any Gateway-Gateway functions desired (such as retransmission or flow control), and internet routing functions. In a Gateway, the internet

side of this structure connects with Gateway "halves" for other nets (see figure 2), while in a Host it serves the internet TCP with its additional protocol functions (see figure 3). Hence a "logical" Gateway may be said to exist between each Host engaged in internetworking and the local net [Walden75]. Gateway halves and internet Hosts on a local net can use the same modules to perform these common functions. Gateway "halves" for different nets can also share the same code except for local net interface modules, reducing development costs for interconnecting new networks.

The major disadvantage of an Endpoint approach is the need for agreement on and implementation of a common internet TCP. The implementation problem may be reduced by creation of internet service sites (accessible through existing local net protocols) which would provide all internet TCP functions, with the local net serving essentially as an access line between processes and TCP (see figure 4). The local net access line may degrade performance of the TCP which is no longer fully end-end. This strategy shares some of the disadvantages of a Hop-by-Hop approach, but allows Hosts with limited facilities (intelligent terminals, packet radio units) to make use of internetwork facilities. Another technique for reducing the burden of internetwork communication facilities on Host resources is to place the TCP in a "front-end" processor (discussed at another session of this workshop).

Hop-by-Hop Implementation (see figure 5)

In implementing an end-end virtual call service using the Hop-by-Hop approach, no common end-end protocol is required within different nets. Instead each local net's own virtual call level service is used, with Gateways translating between each local net service. Only bilateral agreement [Kuo74] between connecting networks is required for translation of local protocols. To facilitate this translation, a number of universal virtual call protocol functions can be identified such as call setup and termination, sending and receiving data, signalling interrupts, and obtaining status information [Higginson75, Binder74].

Unfortunately, translation may still be difficult. In some cases similar but incompatible services are provided by different local net protocols, such as letter based, byte based, or line based flow control. The difficulties of interfacing different flow control mechanisms are frequently underestimated [UKP074, Stokes75]. Other services such as status, echo control, or interrupt may not be provided at all by some local nets. In this case the Gateway must simulate compliance locally without being able to obtain the service at the ultimate destination. In general this reduces internet services to the subset of services offered by all local nets, or requires the end user to be aware of what services he is "really" getting depending on the particular local nets traversed. In any case, the desired function is not truly being provided end-end, but only hop-by-hop, which is particularly worrisome for error control.

When the Gateway is unable to perform automatic translation of control functions, the user must explicitly interact with each Gateway along the path to set up a call, or request particular functions such as echo control or status [Binder74]. These functions must be explicitly requested by using "escape" characters interpreted by one of the

Gateways (rather than passed on as data). Thus Gateway software development in improved translation may be traded-off against user involvement in controlling a connection.

Similar difficulties must be expected with the implementation of other special purpose protocols such as bulk data transfer where local net differences are likely to be greater [Stokes75, Kuo75a], since each new service will require a new translation to be implemented in the Gateway. Each addition of a new Gateway or network also presents a unique translation problem between the local net protocols connected, although acceptance of virtual call protocol standards may simplify this problem somewhat.

Another requirement of the Hop-by-Hop approach is that a single internet path (of Gateways) must be used between source and destination. Sequencing is maintained on each hop. When one hop fails or malfunctions, end-end service is affected since there are no corrective end-end control mechanisms.

5. CONCLUSIONS

A Host level interface between Gateways and the local nets they connect offers several advantages including greater local net autonomy, simpler Gateways, and greater generality. This allows implementation of internetwork services in the Hosts and Gateways on top of local net facilities with minimal disruption of existing network operations. A packet switch level interface does not appear feasible unless networks adopt more compatible and simpler PS-PS protocols than are in evidence today.

Endpoint and Hop-by-Hop interconnection strategies appear best suited for different situations. A Hop-by-Hop approach makes use of existing local net protocols, but may require complex translation between protocols at the Gateway, or explicit intervention by the user. The Endpoint approach requires a common end-end protocol at all sites, but minimum local net and Gateway services. Hence a Hop-by-Hop approach appears most suitable for applications where backward compatability and immediate need requirements predominate, or where more user involvement in controlling operations is acceptable. An Endpoint approach offers greater robustness and generality but requires more universal agreement on standards. Overall software development costs may be reduced by the universal applicability of internetworking modules in the Endpoint approach.

REFERENCES

[Binder74] R. Binder, W. S. Lai, and M. Wilson, "The ALOHANET Menehune - Version II," ALOHA System Technical Report B74-6, September 1974.

[Bright75] R. D. Bright, "Experimental Packet Switch Project of the UK Post Office," in Computer Communication Networks, Grimsdale and Kuo, editors, NATO Advanced Studies Institute Series, E-4, Noordhoff Int., Leyden, Netherlands, 1975, pp. 435-444.

[Cerf74a] V. G. Cerf, Y. Dalal, and C. Sunshine, "Specification of Internet Transmission Control Program," INWG Note 72, revised December 1974.

[Cerf74b] V. G. Cerf and R. E. Kahn, "A Protocol for Packet Network Intercommunication," IEEE Trans. on Communications, COM-22, May 1974, pp. 637-648.

[Cerf75] V. G. Cerf, A. McKenzie, R. Scantlebury, and H. Zimmerman, "Proposal for an Internetwork End to End Protocol," INWG Note #96, September 1975. (Submitted to CCITT Study Group VII)

[Crocker72] S. Crocker, J. Heafner, R. Metcalfe, and J. Postel, "Function Oriented Protocols for the ARPA Network," Proc. Spring Joint Computer Conf., 1972, AFIPS Press, pp. 271-280.

[Heart70] F. E. Heart and others, "The Interface Message Processor for the ARPA Computer Network," Proc. Spring Joint Computer Conf., 1970, AFIPS Press, pp. 551-567.

[Higginson75] P. L. Higginson, and A. J. Hinchley, "The Problems of Linking Several Networks with a Gateway Computer," Proc. European Computing Conf. on Communication Networks, September 1975, Online Conferences Ltd., Uxbridge, England, pp. 452-466.

[Kahn75] R. E. Kahn, "The Organization of Computer Resources into a Packet Radio Network," Proc. National Computer Conf., 1975, AFIPS Press, pp. 177-186.

[Kleinrock74] L. Kleinrock, W. E. Naylor, and H. Opderbeck, "A Study of Line Overhead in the ARPANET," INWG Note #71, September 1974. Also in Comm. ACM 19, 1, January 1976, pp. 3-12.

[Kuo74] F. F. Kuo, "Political and Economic Issues for Internetwork Connections," INWG Note 58, May 1974.

[Kuo75a] F. F. Kuo and D. R. Binder, "Computer-Communications by Radio and Satellite: The ALOHA System," in Computer Communication Networks, Grimsdale and Kuo, editors, NATO Advanced Studies Institute Series, E-4, Noordhoff Int., Leyden, Netherlands, 1975, pp. 397-408.

[Lloyd75] D. Lloyd, and P. T. Kirstein, "Alternative Approaches to the Interconnection of Computer Networks," Proc. European Computing Conf. on Communication Networks, September 1975, Online Conferences Ltd., Uxbridge, England, pp. 499-518.

[MacPherson75] S. MacPherson, ed., "Definitions for Packet-Mode Terms Used in Proposed Recommendations," CCITT Study Group VII Paper No. 212-E, Annex 9, June 1975. Also INWG Protocol Note #27.

[Metcalfe73] R. M. Metcalfe, "Packet Communication," MIT Project MAC Report TR-114, December 1973. (PhD Thesis, Harvard University)

[Pouzin73] L. Pouzin, "Interconnection of Packet Switching Networks," INWG Note 42, October 1973. Also SCH 513.1

[Pouzin74] L. Pouzin, "A Proposal for Interconnecting Packet Switching Networks," INWG Note 60, March 1974. Also SCH 527.

[Pouzin75] L. Pouzin, "Virtual Call Issues in Network Architectures," INWG Note #92, June 1975. Also in Proc. European Computing Conf. on Communication Networks, September 1975, Online Conferences Ltd., Uxbridge, England, pp. 603-618.

[Stokes75] A. V. Stokes, and P. L. Higginson, "The Problems of Connecting Hosts into ARPANET," Proc. European Computing Conf. on Communication Networks, September 1975, Online Conferences Ltd., Uxbridge, England, pp. 25-34.

[Sunshine76a] C. A. Sunshine, "Efficiency of Interprocess Communication Protocols for Computer Networks," P-5614, The Rand Corp., March 1976.

[Sunshine76b] C. A. Sunshine, "Interconnection of Computer Networks," to appear in Computer Networks Journal, 1976.

[Walden75] D. C. Walden and R. D. Rettberg, "Gateway Design for Computer Network Interconnection," Proc. European Computing Conf. on Communication Networks, September 1975, Online Conferences Ltd., Uxbridge, England, pp. 113-128.

[UKPO74] United Kingdom Post Office, "Some Considerations on Flow Control for International Packet Transport," INWG Protocol Note #8, June 1974.

[Zimmerman75] H. Zimmerman, "The CYCLADES End-End Protocol," Proc. Fourth Data Communications Symp., Quebec City, Canada, October 1975, IEEE 75 CH1001-7 DATA, pp. 7-21 to 7-26.

Figure 1 Endpoint Network Interconnection

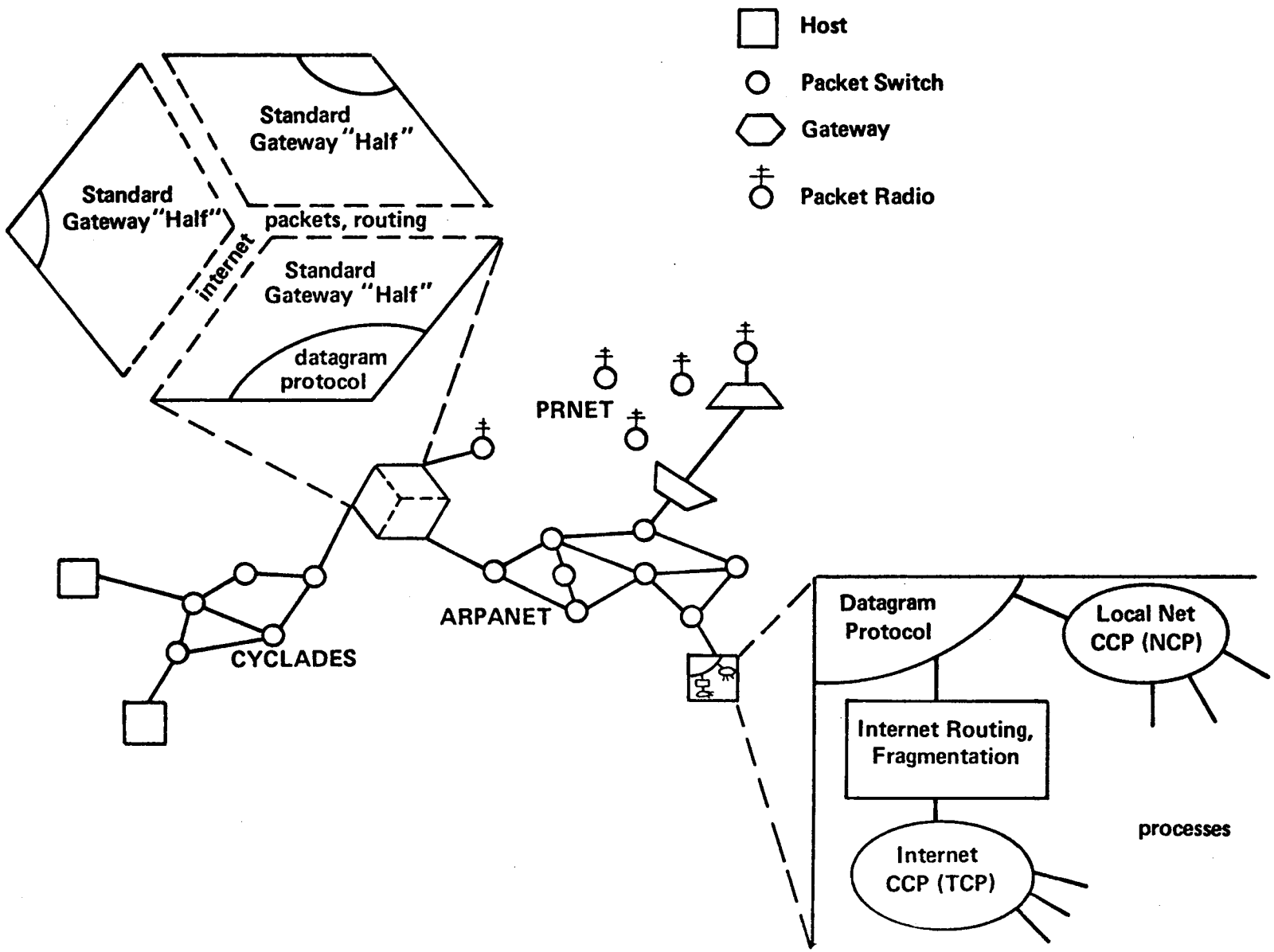


Figure 2 A Gateway "Half"

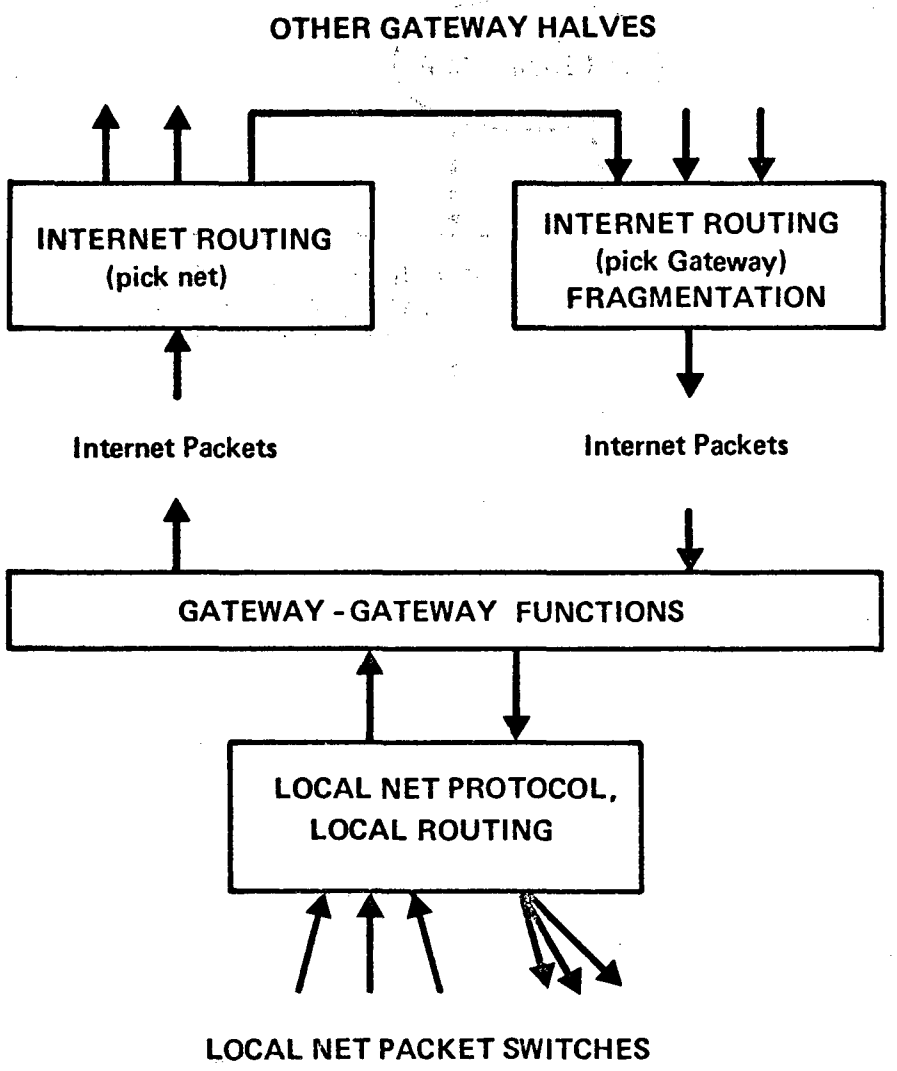


Figure 3 Gateway "Half" In An Internet Host

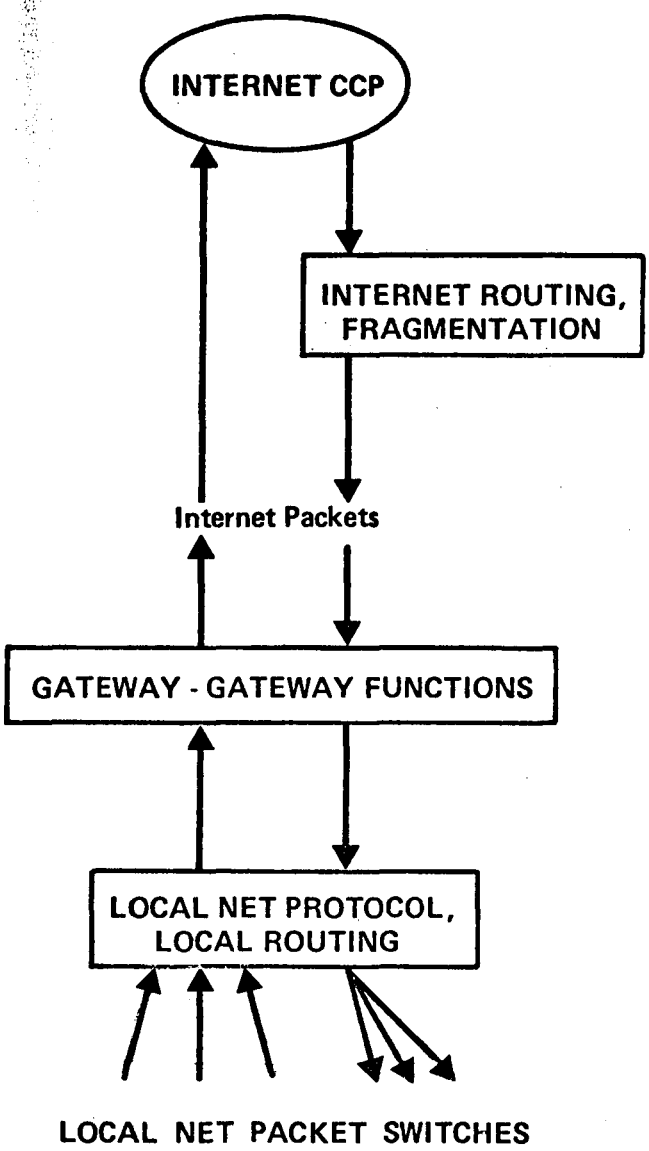


Figure 4 Hop-By-Hop Network Interconnection

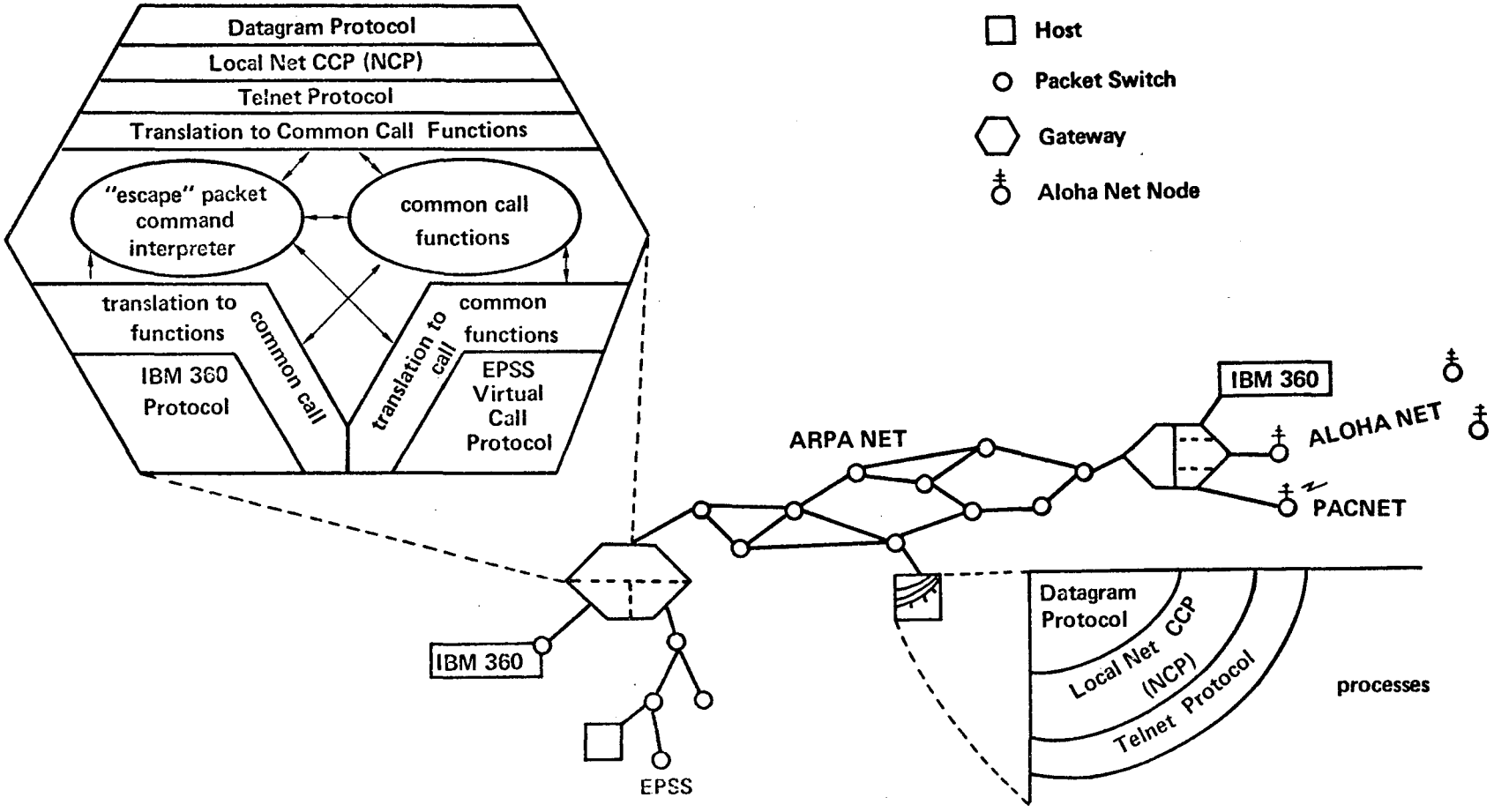
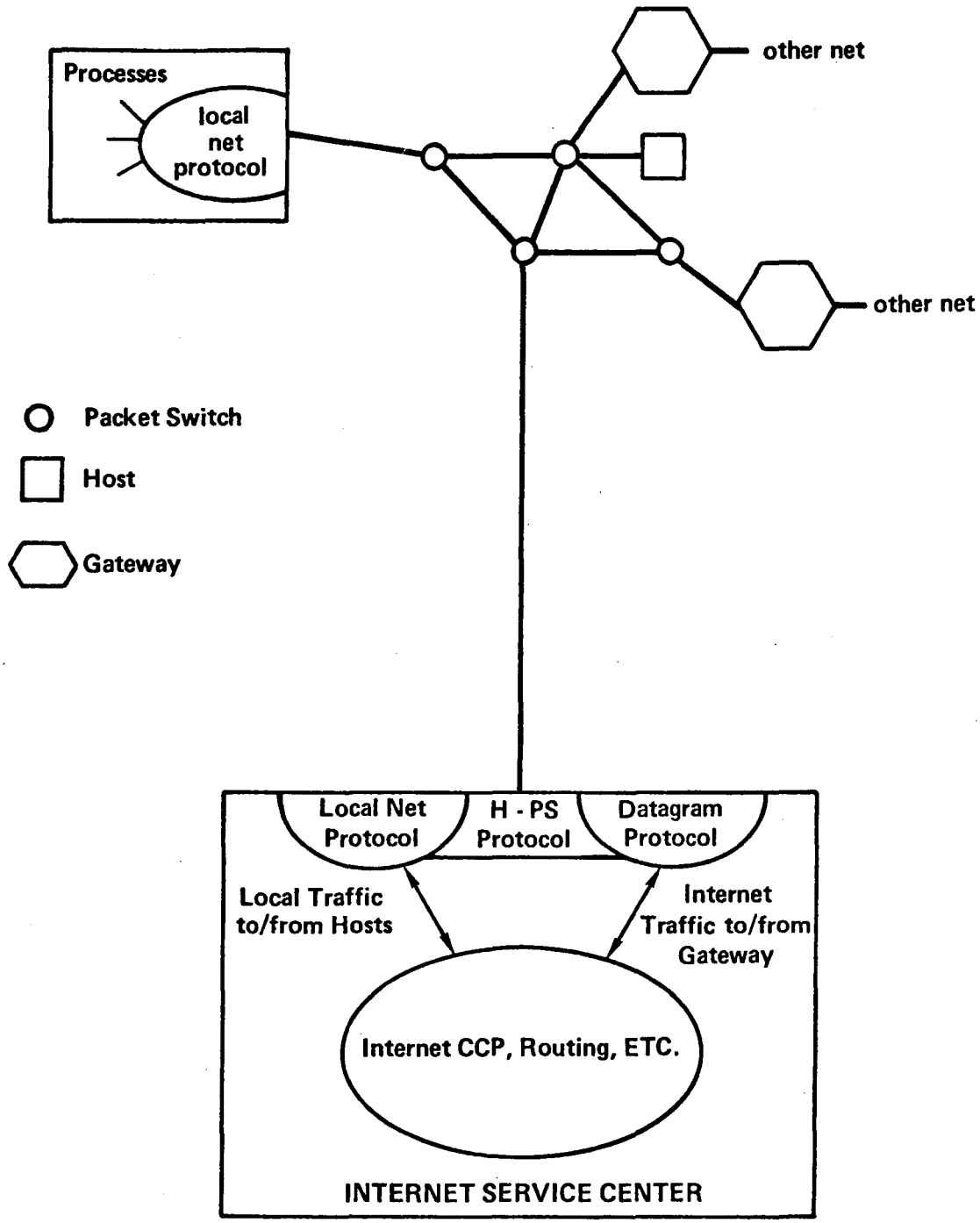


Figure 5 Internet Service Center



VI-B. NETWORK REQUIREMENTS FOR DATA BASE SUPPORT

Synchronization and Resiliency in Network Data Access*

Peter A. Alsberg

Center for Advanced Computation
University of Illinois at Urbana-Champaign

Abstract

The support of multiple data base copies in a network raises important new questions that had only a vague or no counter part at all in single site environments. Two areas of concern are addressed here: synchronization and resiliency.

Synchronization

In single site environments the execution of synchronization primitives to coordinate the access to a shared data base is well understood. A variety of synchronization primitives have been discussed in the literature (e.g., P and V, lock and unlock, block and wakeup, etc.). In a single site environment the execution of a synchronization primitive that does not block a process normally requires 10's or 100's of micro-seconds to execute. The same operation in a network environment will require hundreds of milliseconds. This increased time is due to the unavoidable delay incurred in network message propagation. The synchronization of processes which are not located on the same host must pay the cost of several network message delays to achieve the synchronization. Techniques have been developed to reduce to two or three the number

---

* This work was supported by the Defense Communications Agency Command and Control Technical Center under contract DCA100-75-C-0021

of message delays required to achieve synchronization. The precise number of message delays depends upon the application and the degree of resiliency required.

Even though the number of message delays incurred can be reduced to a very small number, the total lapsed time is still many orders of magnitude greater than was formerly experienced in a single site environment. Synchronizations will still take a significant fraction of a second at a minimum and could be substantially delayed if response time to network messages is slow at a participating host. For many applications this delay is completely unacceptable. Delay is a fundamental property of network service and is heavily influenced by basic physical phenomena like the speed of light. Thus it is not feasible to substantially reduce the time required for network synchronization. The most fruitful avenue for attacking this problem appears to be the development of synchronization avoidance techniques so that the frequency of synchronization in the network environment can be reduced.

Resiliency

The protocols in networks like the ARPA network have some basic flaws when we attempt to apply them to production problems. For example, ARPANET protocols assume that all hosts correctly execute protocol, that no host acts with nefarious intent, that messages are not dropped by the subnet, and that hosts failures occur at a nice place in the execution of a protocol sequence. In fact, all of these assumptions are commonly violated in the ARPANET. Lost messages, hung connections, and hosts that can't talk to each other for a variety of reasons are daily occurrences.

For production operation one needs to support resilient services. Resilient services have four major attributes:

1. they are able to detect and recover from a given maximum number of errors,
2. they are reliable to a sufficiently high degree that a user of a resilient service can ignore the possibility of service failure,
3. if a resilient service provides perfect detection and recovery from n errors, the n + 1st error is not catastrophic. A "best effort" is made to continue service,
4. the abuse of the service by single users should have negligible effect on other users of the service.

This is a careful way of saying that the user of a resilient service should not have to consider the failure of the service in any design which uses the service. He should be able to assume that the system will make a best effort to continue service in the event that perfect service cannot be supported; and that the system will not fall apart when he does something he is not supposed to.

Techniques have been developed for supporting resilient services. In particular, resiliency with respect to communication system and host failures is discussed. Resiliency can never be perfect in a large network environment. However unlikely, it is always possible that all of the dozens of hosts on a large computer network will simultaneously fail and all services will be disrupted. What is important is the establishment of the criteria for acceptable resiliency in such an environment. The concept of n-host resiliency is introduced. In order for service to be disrupted, n hosts must

simultaneously fail in a critical phase of service. Note that it may be possible for n or more hosts to fail outside of the critical phase without disrputing service.

An ARPA-like networks two-host resiliency is all that is ever needed. The meantime between undetected and uncorrected service errors is several centuries. By carefully tuning the sequence of message exchanges it is possible to further extend this by several orders of magnitude. Hence, the consideration of three-host or greater resiliency is of no practical interest.

Resiliency is a question of service integrity. The question of service availability is different. The probability of having at least two-hosts up from among a set of potential server hosts for a service to execute a resiliency algorithm can be low. As a general rule of thumb the magic number seems to be about three or four; using typical large service host availability figures three or four server hosts are required in order to achieve acceptable availability for a resilient network service.

An interesting result is that the addition of resiliency to a previously non resilient service strategy adds no network traffic and negligible processing and storage costs. It does, however, require a somewhat more complex mechanism. Thus, once it is decided to go to a multi-host service strategy (e.g., a multi-copy data base), access to the service may as well be made resilient. It doesn't cost anything more once the initial programming costs are paid. One sour note should be sounded. There are some applications where the integrity of the service (e.g., some data base query and update environments) can be compromised in favor of

achieving higher availabilities with smaller numbers of service hosts. These cases involve two service hosts. Either service host is permitted to operate without the resiliency algorithm if its partner is down. In such an environment the daily occurrence of an undetected and uncorrected error is likely.

DISTRIBUTED FILE SYSTEMS*
(Working Paper)

Yogen K. Dalal
Digital Systems Laboratory
Stanford University, Stanford, Ca. 94305
May 5, 1976.

ABSTRACT

In a computing environment supported by a distributed operating system, it is often necessary to distribute the resources of the file system and its access control among the various host computers of the distributed operating system. This is necessary in order to provide a fail soft file system, and to let files reside at suitable hosts so that the cost for their usage is minimized. This paper examines the logical structure of file systems and shows how they could be distributed, so that any user can reference any file with the same ease, as efficiently as possible. An attempt to do so is complicated by the fact that it is diffciult to maintain consistency between data bases, and that the distributed file system should have provision for the automatic migration of files from one host to another. Adaptive distributed algorithms are proposed for achieving automatic file migration. Their suitability in various network topologies, and under various file usage patterns is under study.

*The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies, either express or implied, of the Defence Advanced Research Projects Agency or the United States Government.

This research was supported by the Defence Advanced Research Projects Agency under ARPA Order No. 2494, Contract No. MDA903-76-0093, and the National Science Foundation under Grant No. NSF MCS 73-07973-A03.

1. Introduction

The file system is one of the most important and basic resources of a distributed operating system that permits resource sharing in a computer network. The concept of integrating memory with the file system as in Multics [Daley68, Bensoussan72], or the dynamic association between memory and file pages as in Tenex [Murphy72], or the fact that many files are executable programs implies homogeneity among the computers in the network if resource sharing is to be successful. However, many files are in standard formats e.g. ASCII or EBCDIC, or can be reconfigured using a data reconfiguration protocol, making them suitable for use even in a heterogeneous computer network. For generality it is assumed that the computer network consists of heterogeneous computers, and that there exist protocols where possible to bridge the heterogeneity. This paper proposes a model for the structure of a distributed file system in such an environment.

No assumptions are made as to whether the computer network provides a single operating system, whose components are distributed, or whether the hosts in the network are independent operating systems designed to treat resources that are remote or local in a similar manner, thereby providing uniform access to all of them. The term distributed operating system will encompass both structures.

For the purposes of this paper, a file is an ordered sequence of elements, which could be words, characters, or bits. The system which controls the mechanisms for access, creation, modification and deletion is called the file system. The collection of directories that control and provide access to the files is the catalog for the file system. Since directories are files, the catalog is a collection of files. At this level a file is formatless and is referenced by a symbolic name. Daley and Neumann provide an excellent discussion on

the structure of a general purpose file system [Daley65]. The logical structure of the file system discussed in this paper is based on their model.

A distributed file system (DFS) is therefore one whose components or resources (the files) are distributed among a number of computer systems, on whose secondary storage systems the resources reside. There are many advantages of having a DFS and for building distributed data bases [Booth72]. One of the primary reasons is to provide a fail soft file system.

There are two aspects of the DFS that we wish to model:

(i) Mechanisms for structuring the directories of the file system, and search algorithms for locating a symbolically referenced file. The file, or subsets of it can then be transfered to the primary memory of the system on which the request was originally generated. The structure of the directories must provide the desired level of access protection.

(ii) Algorithms for permitting files to migrate from one host in the distributed operating system to another so that all the files are (nearly) optimally located, based on an objective function that minimizes the overall cost for accessing and storing the files.

For the purpose of keeping the problems and analysis tractable, certain simplifying assumptions are made about the files in the DFS. In this working paper it is assumed that there exists only one copy of any file in the entire file system, and that the operations permitted on the file will include creation, deletion, modification, examination and execution. It is assumed that mechanisms, similar to ones prevalent in non-distributed file systems [Murphy72, Madnick69], for controlling multiple simultaneous access are also present. It is also assumed that a file is accessed by moving the entire file from the file system storage into the primary memory that simulates a user's virtual memory space.

Resource sharing environments should provide transparency of location to the user, who should be unaware of the distributed nature of the system. Resources which are remote, though referenced identically to locally resident ones, may take a longer time to become available. As a consequence most host operating systems, e.g. those in the ARPANET [Roberts72, McQuillan72, Crocker72], are unaware of the topological structure of the communication network since the hosts are "users" of the communication network. However, when a distributed operating system is attempting to optimize use of its resources, knowledge of the topology is essential. This will be readily apparent when the algorithms for causing files to migrate between the computer systems are considered.

No detailed assumptions are made as to the technology or structure of the distributed environment. The communication subnet could be packet-switched, circuit-switched or a multiaccess channel. The terminology used in this paper will have counterparts in all communication subnets.

One of the primary goals of the DFS presented in this paper is to perform the necessary functions using distributed algorithms, under the assumption that no centralized information source or point of control exists. Such an assumption is necessary in order to preserve the reliability of the DFS. We will investigate the extent to which conventional algorithms used in implementing file systems can be distributed.

2. User Interface and Catalog Structure

The DFS will consist of local file systems at each of the hosts. Their resources are available to all users. Figure 1 shows this model.

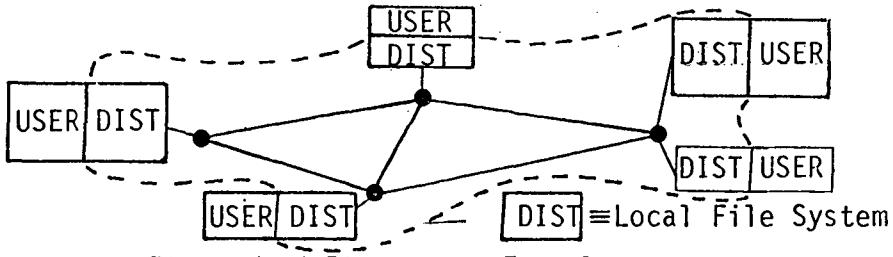


Figure 1. A Distributed File System

The entire file system should have an hierarchical structure, since it seems most appropriate from the users' point of view [Daley65]. Files will be referenced symbolically and shared by a number of users. File names must be unique and must not change even when files migrate to other hosts. The tree name of a file will be its name relative to the root directory. This name does not contain any links. The path name of a file is its name relative to the root directory and may contain links. Hence, in general, a file can have only one tree name but many path names. For the purpose of this working paper assume that the catalog structure does not permit links. The tree name and path name of a file can be specified relative to a working directory other than the root. Since the tree name of the working directory is known the absolute tree name of the file can always be determined.

2.1 Catalog Structure

The logical catalog structure as described above could be centralized at one host, with the non-directory files scattered among various hosts. We rule out this alternative since it contradicts the fundamental requirement of DFS - providing a fail soft file system. The monolithic catalog structure could be duplicated at a few or all hosts. This appears to be the ideal case, since files can be found quickly and file names can follow any convention. Such a scheme is, however, wasteful of space, and introduces a lot of complex problems; namely, how are the various catalogs kept consistent when non-directory files are created, deleted or migrate?

2.1.1 Partitioning based on pure broadcast

Farber and Heinrich propose a technique for structuring the file catalog in the Distributed Computing System (DCS) [Farber72]. In general, the catalog is partitioned into a number of units that contain one or more directories. In order to find a non-directory file, a broadcast is performed as many times as the depth of the path name of that file. Such a technique is simple and appropriate in the DCS, whose subnet is a multiaccess channel and where network addressing is based on process names.

The access protection checks are made while attempting to reference the desired file and not at intermediary directories. Further, all files have their path name associated with them, should it be necessary to find a file without having to go through the directories.

Such a scheme is not suitable, in general, in networks where broadcast is expensive.

2.1.2 Partitioning based on file usage patterns

The monolithic catalog structure can easily be distributed if the root directory at each local file system is the name of that host. Every file created at a local file system will contain the host name as part of its path name. This ensures uniqueness of file names. The forest of trees representing local file systems represents the DFS.

Assume for the moment that files do not migrate. The catalog at each local file system contains the names of files that were created there and also reside there. This is called the Local Catalog (LC). Since there exists a communication subnet, a request for a file not present locally can always be made by broadcasting the tree name for that file. The local file system capable of satisfying the request will probably reply. This is a minor improvement on Farber's scheme.

Now, let us permit files to migrate. For those that do, the LC where they were created does not know where they are, but it must remember that they were created there, so as not to create other files with the same name. The files (directory or non-directory) that have migrated from their parent LC now reside elsewhere. Since the directory files in the tree name of these files will not necessarily be present at the new site of residence, all such files will have an entry in a directory called the Newly Acquired Directory (NAD). It should have a suitable structure to enable a fast search by tree name [TENEX-4].

In addition to the NAD every local file system should have another directory (structured similar to the NAD) that contains what it believes are the current locations of some recently used non-resident files. This will be called the Recently Used Directory (RUD). For a local file system to have an entry in its RUD, the file must have been resident at the host pointed to, though it may have since migrated. The RUD can contain entries for files created at its local host or elsewhere.

2.1.2.1 Search Algorithms

We now briefly indicate how this DFS structure would operate. When a file is created, the LC at that host must be searched to make sure that such a name does not already exist. When a file is referenced (and its tree name is known), the LC is first searched if the tree name contains the local host name in its path. Should the file be present locally, the search is over. If the file referenced was not created at the same host, then the NAD is searched to see if it now resides locally. If so, the search is finished.

If the search fails, the local file system concludes that the file is not present locally, and checks to see if an entry for this tree name exists in its RUD. If not, then the local file system has to resort to a broadcast. If an entry did exist in the RUD, then a request for this file is made to the indicated host, and the search process recurs. The file will either eventually be found, or a point will come where the requested host does not know where the file is, and so broadcast is resorted to. This search will be called the cascade search. We assume all interprocess communication is performed reliably [Cerf74, Cerf74a, Sunshine75].

We have proved elsewhere that, if the system functions correctly, (i.e. no tables have been corrupted) then the cascade search terminates - it either finds the file or has to resort to broadcast. This means that it is not possible for the search to get caught into a loop.

We believe that files will not move around rapidly and so the above technique is appropriate. The size of the RUD at each local file system is dependent on available space and primarily affects local responsiveness. Note that replacement algorithms for the RUD must also be considered.

The catalog structure and search algorithms presented here are an extension of the techniques used by the Resource Sharing Executive (RSEXEC) in the ARPANET [Thomas73]. The RSEXEC does not permit file migration and so the techniques employed there are simpler.

This discussion leads us to conclude that if files are to migrate, then the tree structured catalog (without links) is solely for the convenience of the user, who can name files relative to a given reference point. The file system may have to employ different mechanisms for searching for a file. Further, the complete access protection rights of a file should be stored in a file descriptor block that moves with the file.

3. Optimal Locations for Files

One of the advantages of having a distributed file system is the ability to place files in the network so that the total cost in using them from various hosts in the network is minimized. Techniques for achieving this optimization are examined in this section.

3.1 Related Research

The problem of file allocation in a network is very similar to the plant location problem in Operations Research, and so the models used are very similar. Most of the techniques proposed so far fall into this category [Chu69, Casey72, Eswaran74, Urano74, Levin75, Chang75, Chandy76]. Techniques in estimation theory and time series analyses are used to predict what the file access patterns will be like in the near future, in order to make the optimization adaptive [Segall75, Segall75a].

All these models and techniques model file activity very accurately, but do not consider that the communication cost (usually delay-to-last-bit) may vary with changing load conditions, owing to the presence of other kinds of non-file traffic in the network. Many models make the implicit assumption that the line capacity is large enough so as not to produce queuing delays. In the analysis that assumes non-stationary request patterns for the files it is very difficult to see how the data is gathered or how the results of the optimization are enforced. These models also do not assume the existence of an underlying routing algorithm that is adaptive to changing load conditions, and network failures.

3.2 Distributed Algorithms

If the DFS is considered to be a two dimensional memory hierarchy, the position of a file should move closer to the host that accesses it most often. Files will move around in the DFS, and may even reside at a host where they are not being referenced because hosts nearby access it. Drawing an analogy with force fields, at each requesting site the magnitude of the vector is a function of the request rate and the retrieval delay for the file. The direction of the vector is from the file to the requesting host. The file should be placed in the two dimensional space such that the sum of all the vectors is zero. Hence the file is in "equilibrium".

The algorithm proposed in this paper is based on the "climb" algorithm used as a replacement algorithm in paging systems [Coffman73], except that the file can climb in more than one direction.

Assume that each local file system has knowledge of which hosts are its physical neighbours. It includes its own host in this set. Every time a particular file is accessed, the file system on which it resides calculates the cost to transfer the file to the requestor. The cost is mainly a function of the time taken for this transaction. The file system also records to which physical neighbor the file first went. (In a store and forward network, there may be adaptive routing, and packets of the file may take different paths, and so the physical neighbor recorded is the one to which most packets went for this transaction). Hence, information on the cumulative-cost for a given direction and file is maintained at each local file system. This information is used to determine which files should move and to which physical neighbor. Assume that there is infinite file storage space at each local file system. Hence each file climbs from its present place to another until it reaches what it thinks is the appropriate local file system.

Any dynamic algorithm may lead to instabilities of the kind where the file keeps moving around without any apparent benefit. This could happen if a file kept oscillating between two nodes, or if the movement of one file caused many others to move unnecessarily. The latter situation could arise if the local file systems had finite storage. Therefore a file should be moved to a neighbor file system only if the savings in cost in doing so is larger than a threshold - at least the cost of moving it from one host to another. Each local file system in the DFS may attempt to transfer a file only to one of its immediate physical neighbors. Hence, although requests for a file are coming from all hosts in the network, for purposes of the optimization it appears as though the requests are coming from only the neighboring hosts.

Different strategies for making the decision to cause a file to migrate will now be considered. First some definitions are introduced.

Let U be the Utilization matrix, where an element u(i,n) is the utilization of the file i from neighbor n. U is of dimension NFNxNPN where NFN is the Number of Files at this Node, and NPN is the Number of Physical Neighbors of this node. (The node counts itself as a physical neighbor). An element of U corresponds to the cumulative-cost for a given direction and file. Its value is dependent on the time interval over which the measurement was made, and may also be determined using a predictive algorithm on the measurements.

Let S be the Selection matrix, where an element s(i,n) is the savings made by transfering file i to neighbor n. S is of dimension NFNxNPN. The n that gives the maximum saving for a particular file is the appropriate neighbor to transfer the file.

Let D be the threshold matrix, where an element d(i,n) is the expected cost for transfering file i to neighbor n. D is of dimension NFNxNPN.

The elements of S are determined from the measurements on file utilization over a period of time. At the end of this interval, if there are any files, whose migration would reduce the cost of their usage, then they are transfered to the neighbor that results in maximum cost reduction.

We now show different ways of finding S.

3.2.1 Algorithm 1

The simplest algorithm is one in which

$$S = U - D \qquad --(1)$$

In other words, a file is transfered to the neighbor that makes greatest utilization of it. The algorithm makes no use of the topological relationship of the host on which it is executing with the rest in the network. Hence it is possible that transfer of a file in the direction of greatest utilization may not contribute to lowering the overall cost of usage of the file.

3.2.2 Algorithm II

This algorithm makes some simplifying assumptions regarding the topology and routing algorithms of the network. It assumes that when a file is moved to another neighbor, then the requests and transfer of data from the other neighbors will include the host on which it originally resided on, in its path. In figure 2, if the file was originally at host 1, and is now moved to 2, then traffic from hosts 3, 4, 5 will pass through node 1.

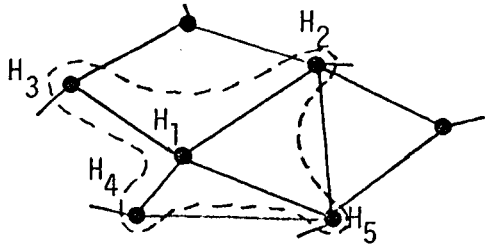


Figure 2. A host and its neighbors in the DFS

Elements of S are calculated as follows:

$$s(i,n) = -\left\{-u(i,n) + \sum_{\substack{g=1 \\ g\neq n}}^{NPN} u(i,g)*L(n)/L(g)\right\} - d(i,n) \qquad ;n=2,\ldots,NPN$$

$$s(i,n) = 0 \qquad ;n=1$$

--(2)

The term in brackets can be understood as follows. The utilization of the file i from its new potential residence n has been reduced by u(i,n) while from the others it has increased by u(i,g)*L(n)/L(g). The L's are the cost of communication over the links that connect the host on which this algorithm is executing to its neighbors. The host on which it is executing is n=1, and the L's are relative to L(1). The savings are relative to the cost of keeping the file at this node.

3.2.3 Algorithm III

This algorithm is an extension of the previous one, and attempts to take into account in greater detail the path taken by file traffic from the neighbors once the file has been moved. If the traffic does not pass through the node corresponding to the original host (as in Algorithm II), then it must pass through some other node such that the cost is less (by virtue of the routing algorithms of the communication subnet). These alternate routes depend on the topology and dynamically changing load conditions. These factors could be taken into account in the following way:

$$s(i,n) = -\left\{-u(i,n) + \sum_{\substack{g=1 \\ g\neq n}}^{NPN} w(g,n)*u(i,g)*L(n)/L(g)\right\} - d(i,n) \quad ;n=2,\ldots,NPN$$

$$s(i,n) = 0 \quad ;n=1 \qquad --(3)$$

where w(g,n) is the topology and routing factor for each neighbor pair. This factor is always less than or equal to one. W is the matrix of these values; it is of dimension NPNxNPN. Further research is required to determine how these factors are calculated, i.e. should they be calculated dynamically or can they be assumed to be heuristic constants?

3.3 Discussion

It is hoped that this distributed optimization will approximate the same optimum as would be predicted by an algorithm that has complete knowledge of the topology and request rates. This is greatly dependent on the rate of variation of the request rates and the speed with which such an algorithm can track the changing optimum. Further, the topology and routing algorithms of the communication network may be such that the stepwise optimization causes a file to get stuck at a local minimum. We have proved elsewhere that this optimization technique works if the network is one-connected (a spanning tree) and that Algorithm II is sufficient to make the correct decision. The effect of other network topologies on the success of this optimization technique is under study.

3.4 An Observation

Traditionally, protocols have been structured in a layered fashion [Crocker72]. This is very appropriate when interfaces between various levels are clean, and state information at lower levels is not required at higher levels. However, a distributed operating system that is trying to optimize the location of its resources in the network must have some knowledge of the topology of the communication subnet. The network of hosts cannot look like a fully connected network for all purposes. The amount of topological information required should be the minimal amount that guarantees some acceptable level of optimization. The algorithms presented in this paper suggest that near neighbor information is sufficient. This is, in addition, helpful when the network is expanded by adding more hosts, since the data structures of only the near neighbors of the new host have to be modified. This means that the host/switching-node interface has to be sensitive to this. The switching node will have to tell the host along which path the packets (say) were sent.

4. Conclusions

This paper has shown the need for a distributed file system, and how one with automatic file migration using adaptive distributed algorithms could be designed. These ideas can be extended to a file system that

(i) has multiple copies of read-only files,

(ii) finite file system storage at each host,

(iii) has appropriate cost measures for permiting transfer of pages of an open file over an extended period of time,

(iv) or permits links in the catalog.

The problem of guaranteeing consistency among duplicate copies of files has not yet been solved. The suitability of the distributed optimization in various network topologies and under various file usage patterns is still under study.

5. References

[Bensoussan72] A. Bensoussan, C. T. Clingen and R. C. Daley, "The Multics Virtual Memory: Concept and Design," Comm. ACM, May 1972, Vol. 15, No. 5, pp. 308-318.

[Booth72] G. M. Booth, "The use of distributed data bases in Information Networks," Proc. First International Conference on Computer Communication, Oct. 1972, pp. 371-375.

[Casey72] R. G. Casey, "Allocation of copies of a file in an information network," Proc. Spring Joint Computer Conf., 1972, AFIPS Press, pp. 617-625.

[Cerf74] V. G. Cerf and R. E. Kahn, "A Protocol for Packet Network Intercommunication," IEEE Trans. on Communications, COM-22, May 1974, pp. 637-648.

[Cerf74a] V. Cerf, Y. Dalal and C. Sunshine, "Specification of Internet Transmission Control Program," INWG Note #72, revised Dec. 1974.

[Chandy76] K. M. Chandy and J. E. Hewes, "File allocation in Distributed Systems," Proc. International Symposium on Computer Performance Modelling, Measurement and Evaluation, March 1976, pp. 10-13.

[Chang75] S. K. Chang, "Data Base Decomposition in a Hierarchical Computer System," Research Report RC 5345, IBM Watson Research Center, Yorktown Heights, New York, March 1975.

[Chu69] W. W. Chu, "Optimal File Allocation in a Multi-Computer Information System," IEEE Trans. on Computers, Vol. C-18, No. 10, Oct. 1969, pp. 885-889.

[Coffman73] E. G. Coffman and P. J. Denning, "Operating Systems Theory", Prentice-Hall, 1973.

[Crocker72] S. Crocker, J. Heafner, R. Metcalfe and J. Postel, "Function Oriented Protocols for the ARPA network," Proc. Spring Joint Computer Conf., 1972, AFIPS Press, pp. 271-280.

[Daley65] R. C. Daley and P. G. Neumann, "A general-Purpose file system for secondary storage," Proc. Fall Joint Computer Conf., 1965, AFIPS Press, pp.213-228.

[Daley68] R. C. Daley and J. B. Dennis, "Virtual Memory, Processes, and Sharing in MULTICS," Comm. ACM, Vol. 11, No. 5, May 1968, pp. 306-312.

[Eswaran74] K. P. Eswaran, "Placement of records in a file and file allocation in a Computer Network," IFIP-74, pp. 304-307.

[Farber72] D. J. Farber and F. R. Heinrich, "The Structure of a distributed Computer System - The distributed File System," Proc. of ICCC 1972, pp. 364-370.

[Levin75] K. D. Levin and H. L. Morgan, "Optimizing distributed data bases - A framework for research," Proc. NCC, 1975, AFIPS Press, pp. 473-478.

[Madnick69] S. E. Madnick and J. W. Alsop II, "A modular approach to file system design," Proc. Spring Joint Computer Conf., 1969, AFIPS Press, pp. 1-13.

[McQuillan72] J. M. McQuillan, W. R. Crowther, B. P. Cosell, D. Walden and F. E. Heart, "Improvements in the Design and Performance of the ARPA Network," Proc. Fall Joint Computer Conf., 1972, AFIPS Press, pp. 741-754.

[Murphy72] D. L. Murphy, "Storage organization and management in TENEX," Proc. Fall Joint Computer Conf., 1972, AFIPS Press, pp. 23-31.

[Roberts72] L. G. Roberts and B. D. Wessler, "Computer Network Development to Achieve Resource Sharing," Proc. Spring Joint Computer Conf., 1970, AFIPS Press, pp. 543-549.

[Segall75] A. Segall, "Dynamic File Assignment in a Computer Network. Part I: Deterministic Parameters," MIT, 1975 (submitted to IEEE Trans. on Automatic Control).

[Segall75a] A. Segall, "Dynamic File Assignment in a Computer Network. Part II: Random Rates of Demand," MIT, Feb. 1975 (submitted to IEEE Trans. on Automatic Control).

[Sunshine75] C. A. Sunshine, "Interprocess Communication Protocols for Computer Networks," Ph.D. Thesis Stanford University, 1975, DSL Technical Report #105.

[TENEX-4] TENEX-4 File System, 14 Jan 1970.

[Thomas73] R. H. Thomas, "A Resource Sharing Executive for the Arpanet," Proc. NCC, 1973, AFIPS Press, pp. 155-163.

[Urano74] Y. Urano, K. Ono and S. Inoue, "Optimal Design of Distributed Networks," Proc. ICCC, August 1974, pp. 413-420.

VI-C. DATA ACCESS AND MANIPULATION LANGUAGES

NIC/QUERY, A Novice User Interface Program

E. J. Feinler and D. S. Maynard

SRI, Augmentation Research Center, Menlo Park, Calif.

ABSTRACT

This paper describes the NIC/QUERY program and the data bases that run under it. NIC/QUERY permits novice or sometime users to access a data base made up of structured files. It uses a few simple commands and is designed to be self-explanatory. The query program is a subsystem of the SRI Augmentation Research Center Online System (NLS). NLS utilizes heirarchically structured files. It also supports a linking capability which allows special strings of text, called links, to reference any node in any NLS file. NIC/QUERY takes advantage of these two NLS features. Users do not need any knowledge of NLS to use the query program, even though the program draws upon sophisticated NLS procedures.

NIC/QUERY is being used by the ARPA Network Information Center at SRI to enable network users to browse through information about the ARPANET. This application of the program is described in detail. It shows how an NLS user, without programing experience, can construct a data base which can in turn be easily browsed by a novice user through use of the NIC/QUERY subsystem.

INTRODUCTION

In 1972 the Network Information Center (NIC), a part of the Augmentation Research Center at SRI, was asked to participate in a demonstration of the ARPANET at the International Computer Communication Conference (ICCC). A large amount of information had already been gathered describing the ARPANET, but a method was needed for presenting this information to the conference attendees, many of whom were not computer oriented. The information was to be accessed through the network, since that was the emphasis of the meeting. We also wanted to draw heavily upon useful features of the SRI Online System (NLS), particularly the heirarchically structured file system and linking capability. However, we felt that the

NLS command language and subsystem access were too sophisticated for random users. It was this need for a simple subset of the full NLS capability that would allow novice users to enter NLS and view data files with little or no knowledge of the system, that triggered the concept and design of NIC/QUERY.

NLS FILE SYSTEM

To understand how the Query program accesses data it is necessary to have an overview of the NLS file structure (4)(6).

NLS operates on a hierarchical, random file system with several unique features that provide advantages for entering, organizing, and accessing data. Storage of information within separate structure and data blocks permits rapid movement within and between NLS files.

Conceptually, an NLS file is a tree. Each node has a pointer to its first subnode and a pointer to its successor. If it has no subnode, the subpointer points to the node itself. If the node has no successor, the successor pointer points to the node's parent. Each node is currently represented by a ring element. These ring elements point in turn to the associated data block.

The basic unit of an NLS file is the 'statement' which is a string of text made up of from 1 to 2000 characters. A statement may be a character, line, sentence, heading, paragraph, table, or graphic. A file is composed of one or more statements that can be arranged at different levels in an outline or tree form. The text in an NLS file can be clipped and re-expanded, and the user's view of it may be altered by single-letter codes known as viewspecs.

For a more detailed description of the NLS file structure see (4). For the purpose of this discussion it is sufficient to know that NLS has a hierarchical or tree-like file system made up of multilevel statements. Statements, or text within statements, may be easily referenced, and level and view clipping are available.

STRUCTURE OF THE DATA BASES

In order to discuss NIC/QUERY in specific terms - what it does and how it works - an NLS file, which happens to be a short description of one of the Hosts on the ARPANET

(LLL-RISOS, dec 21), has been included as a sample data base (See Appendix 1). This is a typical file used to produce the hardcopy ARPANET Resource Handbook, a document published by the NIC under the sponsorship of the Defense Communications Agency (DCA) for distribution to ARPANET users (2). The same files used to produce the hardcopy handbook can with minor modifications serve as the data base for Query. This eliminates the need to recapture or reorganize the data extensively for hardcopy production.

The basic structure of this sample file is that of an outline. A heading or data element is followed, down a level, by the data pertaining to it. Note that some of the descriptive text is presented in paragraph form and some is presented as tables. Either is acceptable.

HOW NIC/QUERY WORKS

Essentially NIC/QUERY does the following: it creates a numbered menu from any headings or data element names at a given level and presents this menu to the user. The user then selects one of the menu items or specifies a keyword. Upon selection of an item or data element by the user, the program displays the text one level beneath that data element. The user can 'walk' down through the outlined file in a logical sequence opening up successive levels of information. He can also jump back to a previous menu or go back to a higher node in the tree structure and go down a different information path. This path can be a branch within a file or an entirely new file; thus the data base can be a single file or multiple files, whichever the builder chooses.

Although this is a very simple concept, its application creates a powerful data access tool which any NLS user can drop over a properly structured NLS file.

The simplicity of the program does mean that it has several limitations. These are important to note:

- NIC/QUERY is not primarily a data searching tool, it is rather a data displaying tool. It will not find all occurrences of a given subject or keyword, although it will find the first occurrence of a keyword within a given file. It does not have Boolean searching capability at present.

- The current NIC/QUERY is designed for use on a

teletype terminal rather than a CRT display, although it could easily be adapted for either.

- NIC/QUERY is not suitable for very large data bases. The file size is not open-ended. NLS was not designed to be a data base management system in the traditional sense, and does not effectively support multiple parallel organizations on one data file. There is also a limit to the number of initial data paths that can be introduced to the user and still be meaningful.

- Although the user can go down many successive levels within a file, there is a limit to how many levels he wants to go down before the number of menu selections becomes impractical.

As presently implemented, NIC/QUERY is a useful tool for displaying moderate amounts of data, but it is not suitable for searching or for accessing large data bases. (The ARPANET Resource Data Base contains approximately 150 files and represents about 1500 pages of written text. This is what is meant by 'moderate', as opposed to, for example, large bibliographic data bases with many thousands of records.)

Another limitation that should be mentioned is that of insufficient online access. A computer query system is useful only if the user has enough access, and the response time is adequate for his needs. An inherent problem in any online query system is the time required for finding and displaying information compared with looking the same information up in a well-organized book. On the other hand, online data is easier to keep current than that contained in a book, and it is at the user's finger tips when needed. All other factors being equal, the amount of online access and speed of data display will probably have a strong influence on user acceptance of a fully automated query system.

THE COMMAND LANGUAGE

The original ICCC version of the query program had two commands: 'show' and 'quit' (8). It was obviously easy to learn, but severely limited movement within files. Also, the original version required the user to type in the name of the data element that he wished to 'show'. This was time consuming and introduced typing errors. Consequently, a few more commands and a numbered menu selection feature were added. It should be emphasized,

however, that the point is NOT to have many commands, and indeed, one of the hardest design problems has been to keep new commands from creeping in when the present ones, or lack thereof, will do.

The command language for NIC/QUERY is very simple and consists of the following commands:

[@]NIC <CR>

where 'NIC' is a single program-access word, typed by the user immediately after login. It automatically loads NLS and the Query subsystem and presents the user with his first set of instructions. (This word can easily be changed for access to data bases other than those of the NIC.)

TYPEIN <CR>

where TYPEIN is usually a menu number corresponding to a chosen data element followed by a carriage return. In addition the user may at any time type in a keyword. Query will search the data tree below the current node for a corresponding data element, and it will display the first data element it finds which matches the search criteria. The user may also type in a phrase as a series of words separated by spaces. Query will perform the indicated word searches sequentially. This provides a user who is familiar with the data base, or a branch therein, with direct access to the data element he seeks. In essence it allows a more experienced user to 'come in from the side' of a file rather than down from the top.

As an optional feature the data base builder may also define additional "index" files which contain keywords and pointers to relevant data nodes. These index files will be searched automatically whenever the given keyword cannot be found within the user's current context.

↑

takes the user up to the next highest level and displays the menu for that level.

←

takes the user back through previously displayed nodes, one at a time. The user can accept the choice displayed or continue jumping back. This facility might be thought of as a list of 'book markers' marking nodes where the user has already been and to which he may want to return.

?

displays a list of legal commands

CONTROL-O

aborts TYPEIN back to the prompt

CONTROL-A

backspaces a letter

CONTROL-W

backspaces a word

DEL (CONTROL-X)

aborts session but gives user the option of leaving the system, continuing or, beginning again, in case the abort was accidental.

QUIT <CR>

aborts session and returns the user to EXEC level.

BUILDING THE INSTRUCTION SET AND LINK TABLE

Before discussing the format and content of the instruction set and link tables, the concept of NLS links will be introduced. Links are strings of characters in a statement that name the address of any location in any NLS file along with a specified view. Links have the syntax <ADDRESS : VIEWSPECS> and may have several kinds of delimiters. Links may be automatic (that is, during a pointing command, if the program finds the link syntax within the specified text, it will automatically jump to the address specified by the link), or they may be manual (the user must issue a command 'Jump to link' and bug or address the link string.)

Query makes extensive use of automatic links which the user never sees. Thus a query data base may be simply a table of links off to multiple files or addresses within files. Appendix 3 gives a condensed version of the link table for the ARPANET Resource Data Files. This particular data base is composed of over 100 small files, and was chosen to demonstrate how easily multiple files can be included under one query 'umbrella'.

A Query data base can also consist of only one file, if the builder chooses. In fact, this would be the simplest to build. With only a single data file, links would be missing altogether or would point to addresses within the file.

When the user types the recognition word to enter Query, e.g, NIC <CR>, he enters NLS and the query subsystem automatically, and causes a file containing the initial menu of topics and the self-explanatory instruction set to be loaded and displayed. The contents of this file can be tailored to fit the data being viewed, and it is written by the data base builder.

The instruction set generally explains how to make a menu selection and states the purpose for which the data is intended. It then gives a beginning menu from which the user can begin branching downward. One item on the initial menu can be a verbose HELP file corresponding to a HELP document when printed out. Any node may have an instruction set of its own pertaining only to that node.

A LOOK AT THE ARPANET RESOURCE DATA THROUGH QUERY

Appendix 2 gives a brief scenario of a typical ARPANET user querying the ARPANET Resource Data Files.

QUERY AS AN NLS SUBSYSTEM

Originally NIC/QUERY was a separate user program, or what is commonly known as a 'hack'. It was designed for a demo and was thought of as transitory. However, as often happens with such programs, it was useful and refused to go away. In the first version the command language and instruction set were specified by the program and were specific to the NIC data base. The instructions and introductory menus could not be altered by the data base builder without an understanding of the code and without being able to modify and recompile the program.

As other applications for the same type of novice access arose, several similar NLS tools began to evolve. The NLS online HELP system was one of these (5). Implementation of this system provided many of the design features now incorporated into NIC/QUERY. To accomodate other NLS users who wanted to adapt similar programming concepts to their own data bases, NIC/QUERY was rewritten as a generalized data access tool. Now it is a subsystem of NLS available to all NLS users. The instruction set and introductory files can be written and edited by the data base builder. The command language is written in a high level language, so that commands as well as prompts associated with commands can be easily altered with a minimum of programming effort. This means that the Query program has achieved a high degree of flexibility, and can be used by NLS users to display a variety of data bases of their own design and choosing.

FUTURE POSSIBILITIES

As stated above NIC/QUERY is not a searching tool at present. We would like to add searching capability including Boolean logic. Since NLS is not a data base management system, we would like to interface NLS with an already existing database management system in such a way that information could be captured, edited, and viewed in NLS, but stored and searched in the data base management system. Data files could be queried through the query program and the results displayed at the user's console. The user would not need to know that more than one system was invoked to produce an answer, nor what the hand shakes were among them.

It would be useful to be able to store data elements sequentially within statements as well as in separate statements. This is possible now but needs to be simplified for the data base builder. We would like to have an easily constructed data input prompting system. This would allow the data base builder to specify data input elements for which the system would construct suitable input prompts, e.g., for a data base consisting of personnel data, the builder might specify that NAME, ADDRESS, and TELEPHONE NUMBER are data elements. The input system would then prompt a clerk for these data elements while the question-mark facility would supply added instructions.

NLS is a sophisticated, extensible, modular information processing system. A growing online user community uses

NLS to maintain a wide variety of information. To the systems builder NLS provides a unique environment for modular programming. A Command Meta Language (CML), which is a high level language for specifying the user interface of a modular extension or subsystem, is available (1). A Command Language Interpreter (CLI) parses the user input according to the grammar specified by the CML (3). The CLI performs the functions of command recognition, command completion, command feedback, prompting, response to ?, and limited editing and validation of user entered parameters. NLS also provides a rich set of primitives for manipulating information within NLS files (7).

Many application subsystems for managing particular data bases have been built using these NLS tools (9). Typically these subsystems perform the tasks of data entry, data verification, data update, interactive query, and report generation. Implementation time for producing such a subsystem within the NLS environment is on the order of a few man-months from design to completion.

A general methodology of data manipulation and display is evolving out of the work already done on these various subsystems, of which NIC/QUERY is only one piece. The goal is to provide a system which allows any NLS user to effectively become a systems builder and data base administrator. In other words, the average NLS user will be able to define, create, and maintain a data base easily, using only his existing NLS capabilities. Query will provide an accessing mechanism for his data base suitable for a wide range of users.

SUMMARY

NIC/QUERY is a program that permits nonprogrammers to present data to a novice user population. Combined with the text editing and data handling features of the SRI Online System (NLS), it constitutes a useful approach to providing data access in a networking environment. The interesting features of NIC/QUERY are: 1) the data base and the instruction set can be written by a nonprogrammer, 2) the program is self explanatory, 3) files used for the data base can be readable, outlined files usable for other purposes, 4) the command language and prompts can be changed with simple programming modifications, and 5) access to NLS, Query, and the data files is automatic. Query is one of a collection of

data-handling tools provided by the SRI Augmentation Research Center NLS system.

REFERENCES

1. Dornbush, C. F., Victor, K. E. and Irby, C. H. A Command Meta Language for NLS, ARC 22130, Augmentation Res. Center, Stanford Res. Inst., Menlo Park, Calif., Jan. 10, 1975.

2. Feinler, E. J. ARPANET Resource Handbook, NIC 23200, Augmentation Res. Center, Stanford Res. Inst., Menlo Park, Calif., Sept. 1975.

3. Irby, C. H. The Command Meta Language System, ARC 27266, Augmentation Res. Center, Stanford Res. Inst., Menlo Park, Calif., Jan. 6, 1976.

4. Knowledge Workshop Development, ARC 22133, Final Rept. on SRI Proj. 1868, Augmentation Res. Center, Stanford Res. Inst., Menlo Park, Calif., Jan. 30, 1976, p. 17.

5. Ibid, p. 15.

6. Lehtman, H. G. NLS File Structure: Documentation, ARC 17069, Augmentation Res. Center, Stanford Res. Inst., Menlo Park, Calif., June 6, 1973.

7. Smith, D. C., Maynard, D. S., and Cornish, J. A. Information Retrieval Primitives for NLS, ARC 27356, Augmentation Res. Center, Stanford Res. Inst., Menlo Park, Calif., May 13, 1976. (An SRI-ARC Internal Working Paper)

8. Vallee, J. F. Q1: A Simple Retrieval Tool for TNLS Structured Files, NIC 11841, Augmentation Res. Center, Stanford Res. Inst., Menlo Park, Calif., Sept. 19, 1972.

9. White, J. E. QUERY2, L10 Source Code, NIC 28001, Augmentation Res. Center, Stanford Res. Inst., Menlo Park, Calif., May, 7, 1976. (Not available for distribution)

APPENDIX 1

SAMPLE OF AN NLS FILE

(LLL-RISOS) UNIVERSITY OF CALIFORNIA
LAWRENCE LIVERMORE LABORATORY

(FUNCTION)

SERVER COMPUTER: PDP-11/45 HOST ADDR. 21

(ADDRESS)

University of California
Lawrence Livermore Laboratory
RISOS Project, L-307
Box 808
Livermore, California 94550

(PERSONNEL)

PRINCIPAL-INVESTIGATOR
Robert P. Abbott (ABBOTT@ISI) (415) 447-1100

LIAISON
James E. Donnelley (JED@BBN) (415) 447-1100

SOFTWARE-CONTACT
Charles R. Landau (RISOS@ISI) (415) 447-1100

HARDWARE-CONTACT
Doyle R. Hopkins (RISOS@ISI) (415) 447-1100

(HARDWARE)

(COMPUTER)

TYPE	CORE AMOUNT	CORE SPEED	WORD LENGTH
PDP-11/45	56K	1.0 microsec	16 bits

(PERIPHERALS)

HOW MANY	TYPE	MAKE	MODEL
DISKS			
1	1.2M word platter	DEC	RK11
1	20M word pack	DEC	RP11

DRUMS None

TAPES

1	9 track, 1600 bpi	DEC	TU66F-9/45
1	dual Dectape	DEC	TU56

PRINTERS

1	96chr, 7x9 matrix	Versatec	LP1150

CARD-READERS

1	300 cpm	DEC	CR11

DATA-COMMUNICATIONS

5	300 bd auto ans	Vadic	VA305D
3	Auto dial	Vadic	VA801AD
1	2000 bd synch	ICC	2200/20

(TERMINALS)

HOW MANY	TYPE	MAKE	MODEL
7	96 chr 5x7 matrix	TI	733 KSR
3	96 chr 5x7 matrix	TI	725
1	96 chr CRT w/edit	Hazeltine	2000
6	128 chr CRT w/edit	Beehive	SB2

(OPERATING-SYSTEM)

The RATS (RISOS Arpanet Terminal System) is a capability list, virtual memory, multiprogramming system. RATS allows controlled communication between mutually suspicious processes in separate environments and permits naturally extended use of all programs through a user definable capability mechanism.

(USER-PROGRAMS)

RATS currently has very few user programs. Most of the existing programs are utility programs or programs for using the communication equipment on the system. All available programs are kept in the RATS public directory and the available program documentation is kept online (see HELP below). Anyone that is interested in writing programs for a C-list system are encouraged to try their hand at RATS. Contact the Liaison for more information.

(SYSTEM-USE)

Currently undergoing rapid change. Users interested in writing software are welcome. Look for online documentation or contact the Liaison.

(HELP)

The command language processor and most subsystems respond to "?" with a list of available commands and to "help <CR>" with more complete information.

(PROTOCOLS)

(SERVER)

The ARPA Network server protocols currently implemented are:

1. TELNET (new server only. Socket 23)

(USER)

The ARPA Network user protocols currently implemented are:

1. TELNET

2. FTP

(INTERESTS)

The RISOS (Research In Secured Operating Systems) project at the Lawrence Livermore Laboratory is interested in the ARPA Network largely to connect to remote computers on the network for testing operating system integrity and for transporting files (e.g., operating system listings) and general communication (e.g., network mail). The heavy communications orientation of the RATS system has demanded a continued active interest in communication protocols.

The LLL IMP has been eyed by several other groups at the Laboratory for applications from file transport of data at LLL for specialized processors on the ARPANET to possible connection of the Laboratory's unclassified CDC-6600 computer. Such applications are currently only speculation.

The Laboratory is also currently designing and building a computer network for use by the ERDA laboratories doing research on controlled thermonuclear fusion. (This has been dubbed the CTR subnet).

(DOCUMENTATION)

For a description of RATS as of March 1, 1974, see:

1. An Introduction to RATS (RISOS/ARPA Terminal System): An Operating System for the DEC PDP-11/45, UCRL 51582, Lawrence Livermore Laboratory, Livermore, Calif. (Mar. 1, 1974)

Available from the Technical Information Department of the Lawrence Livermore Laboratory, Box 808, Livermore, Calif. 94550.

The current RATS Handbook is available online as <RISOS>RATS-HANDBOOK.TXT from BBN using the Anonymous login convention under BBN's FTP server.

APPENDIX 2

SCENARIO OF ACCESS TO A DATA BASE THROUGH NIC/QUERY

The following is a brief scenario of a typical ARPANET user querying the ARPANET Resource Data Files. ("USER: xxx" = what the user types; "QUERY: xxx" = what the program replies; <CR> = a carriage return typed by the user)

USER: nic <CR>

QUERY:

The NIC/QUERY system lets ARPANET users view files that describe the ARPANET. Each list of topics is presented to the user as a numbered menu of selections.

To see more detail on one of the topics below, type its corresponding number followed by a carriage return <CR>.

1. HELP and additional Query commands
2. GENERAL-INFORMATION about the ARPANET
3. PERSONNEL-INFORMATION
4. HOSTS
5. ACRONYMS
5. PROGRAMS
6. PROTOCOLS
7. DOCUMENTATION

< for back, ↑ for up, or TYPEIN:

USER: 1 <CR>

QUERY:

(HELP) and additional query commands

?	Gives a list of legal commands.
↑	Takes you up to the next highest level and redisplays the menu for that level.
←	Takes you back to previous views. For each previous view the system will show you the first line of the view

and ask "Y/N?" If the view shown is the one you want, type: y <CR>. If it is not, type: n <CR> to see another view.

TYPEIN — You can type a keyword followed by a <CR> instead of typing a menu number. The system will search for this keyword.

DEL — Aborts session but gives you the option of continuing, beginning again, or leaving the system in case the abort was accidental.

CONTROL-O — Aborts TYPEIN back to the last prompt

CONTROL-A — Backspaces a character

CONTROL-W — Backspaces a word

quit <CR> — Aborts session and returns you to "@". (NOTE: You must type out the word 'quit'.)

< for back, ↑ for up, or TYPEIN:

USER: 4 <CR>

QUERY:

(HOSTS)
1. SERVERS
2. USERS
3. TIPS

< for back, ↑ for up, or TYPEIN:

USER: 1 <CR>

QUERY:

(SERVERS)

1. AMES-67	2. ANL	3.ARPA-DMS
4. BBN-11X	5. BBN-TENEX	6. BBN-TENEXA
7. BBN-TENEXB	8. BBN-TENEXD	9. CCA-TENEX
10. CHII	11. CMU-10A	12. CMU-10B
13. HARV-10	14. HAWAII-500	15. I4-TENEX
16. I4B-TENEX	17. LBL	18. LL

---- ETC ----

< for back, ↑ for up, or TYPEIN:

USER: 17 <CR>

QUERY:

(LBL) UNIVERSITY OF CALIFORNIA
LAWRENCE BERKELEY LABORATORY
MATH AND COMPUTING DEPARTMENT

1. FUNCTION 2. ADDRESS 3. PERSONNEL
4. ACCOUNTING 5. SERVICE-SCHEDULE 6. LOGIN
7. LOGOUT 8. CONTROL-CHARACTERS 9. HELP
10. NETWORK-COMMANDS 11. PROTOCOLS 12. HARDWARE
13. OPERATING SYSTEM 14. USER-PROGRAMS 15. INTERESTS
16. DOCUMENTATION

< for back, ↑ for up, or TYPEIN:

USER: 3 <CR>

QUERY:

(PERSONNEL)
MANAGERS
James A. Baker (415) 843-2740
David F. Stevens (415) 843-2740
LIAISON
Dennis E. Hall (DEHall@SRI-AI)(415) 843-2740
ACCOUNTS
Thomas P. Hitchcock (415) 843-2740

---- ETC ----

< for back, ↑ for up, or TYPEIN:

USER: 2 <CR>

QUERY:

(ADDRESS)
University of California
Lawrence Berkeley Laboratory
Math and Computing Department
Building 50B, Room 1143
Berkeley, California 94720

USER: lll-risos <CR>

(NOTE: Here the user 'came in from the side' by typing a keyword, since he knew the name of the host whose write-up he wished to view.

QUERY:

(LLL-RISOS) UNIVERSITY OF CALIFORNIA
LAWRENCE LIVERMORE LABORATORY

1. FUNCTION	2. ADDRESS	3. PERSONNEL
4. HARDWARE	5. OPERATING-SYSTEM	6. USER-PROGRAMS
7. SYSTEM-USE	8. HELP	9. PROTOCOLS
10. INTERESTS	11. DOCUMENTATION	

< for back, ↑ for up, or TYPEIN:

USER: 1 <CR>

QUERY:

(FUNCTION)
SERVER COMPUTER: PDP-11/45 HOST ADDR 21

< for back, ↑ for up, or TYPEIN:

USER: ↑ <CR>

QUERY:

(LLL-RISOS) UNIVERSITY OF CALIFORNIA
LAWRENCE LIVERMORE LABORATORY

1. FUNCTION	2. ADDRESS	3. PERSONNEL
4. HARDWARE	5. OPERATING-SYSTEM	6. USER-PROGRAMS
7. SYSTEM-USE	8. HELP	9. PROTOCOLS
10. INTERESTS	11. DOCUMENTATION	

< for back, ↑ for up, or TYPEIN:

USER: 5 <CR>

QUERY:

(OPERATING-SYSTEM)
The RATS (RISOS ARPANET Terminal System) is a capability list, virtual memory, multiprogramming system. RATS allows controlled communication between mutually suspicious processes in separate environments

and permits naturally extended use of all programs through a user definable capability mechanism.

< for back, ↑ for up, or TYPEIN:

USER: <DEL> <CR>

QUERY:

Type C to continue, Q to quit, or B to begin again:

USER: b <CR>

QUERY:

(HELP-Q)

The NIC/QUERY system lets ARPANET users view files that describe the ARPANET. Each list of topics is presented to the user as a numbered menu of selections.

To see more detail on one of the topics below, type its corresponding number followed by a carriage return <CR>.

1. HELP and additional Query commands
2. GENERAL-INFORMATION about the ARPANET
3. PERSONNEL-INFORMATION
4. HOSTS
5. ACRONYMS
6. PROGRAMS
7. PROTOCOLS
8. DOCUMENTATION

< for back, ↑ for up, or TYPEIN:

USER: programs <CR>

QUERY:

(PROGRAM)
1. BY-NAME
2. BY-TYPE

< for back, ↑ for up, or TYPEIN:

USER: 1 <CR>

```
QUERY:

   (BY-NAME)
     Typein a program name followed by a <CR>, e.g. ALGOL
   <CR>

     This will give a definition of the program, and
     list the hosts on which it is available.

   < for back, ↑ for up, or TYPEIN:

USER: zog <CR>

QUERY:

   ZOG
     CMU query language.  An interactive guide for
     novice  users which provides an introduction to
     many  available systems.
       CMU-10A
       CMU-10B

   < for back, ↑ for up, or TYPEIN:

USER: <<< <CR>  (NOTE:  User has backed up through 3
nodes)

QUERY:

   "(BY-NAME)" Y/N: "(PROGRAMS)" Y/N: "(HELP-Q)" Y/N

USER: n <CR> [to first two]; y <CR> [to final node]

QUERY:

   (HELP-Q)
     [displays beginning menu again.]

   < for back, ↑ for up, or TYPEIN:

   ---- Here user decides to stop ----

USER: quit <CR>

QUERY:  Leaving Query
```

APPENDIX 3

NIC/QUERY BEGINNING INSTRUCTION FILE AND TABLE OF LINKS

The Scenario above shows what text the user sees. This sample file shows everything in a beginning instruction file and link table, much of which the user does not see. Hidden links are recognizable by the ## <DIRECTORY,FILENAME,STATEMENT-NUMBER:VIEW> ## syntax. Also, the NOTEd explanations are not part of the actual file.

(HELP-Q)

The NIC/QUERY system lets ARPANET users view files that describe the ARPANET. Each list of topics is presented to the user as a numbered menu of selections.

To see more detail on one of the topics below, type its corresponding number followed by a carriage return <CR>.

(HELP) and additional query commands

(NOTE: This menu item has no link since the information to be displayed to the user occurs one level below the element name 'HELP')

?	Gives a list of legal commands.
↑	Takes you up to the next highest level and redisplays the menu for that level.
←	Takes you back to previous views. For each previous view the system will show you the first line of the view and ask "Y/N?" If the view shown is the one you want, type: y <CR>. If it is not, type: n <CR> to see another view.
TYPEIN	You can type a keyword followed by a <CR> instead of typing a menu number. The system will search for this term.
DEL	Aborts session but gives you the option of continuing, beginning again, or

```
                leaving the system in case the abort
                was accidental.

   CONTROL-O  Aborts TYPEIN back to the last prompt

   CONTROL-A  Backspaces a character

   CONTROL-W  Backspaces a word

   quit <CR>  Aborts session and returns you to "@".
           (NOTE:  You must type out the word 'quit'.)

(GENERAL-INFORMATION) about the ARPANET
## <NETINFO,HELP-GEN,1:g> ##

   (NOTE:  This is a hidden link off to another file
   that has its own instructions)

(PERSONNEL-INFORMATION)
## <*,IDENTS,1:g> ##

(HOSTS)

   (SERVERS)

      (AMES-67)
      ## <netinfo,ames-67,1> ##

      (ANL)
      ## <netinfo,anl,1> ##

      (ARPA-DMS)
      ## <netinfo,arpa-dms,1> ##

         ---- ETC ----

            (NOTE:  ---- ETC ---- indicates that the
            list of links is actually much longer.
            Only enough examples are included here to
            show the structure of the link table.)

   (USERS)

      (ADR)
      ## <netinfo,adr,1> ##

      (AMES-11)
      ## <netinfo,ames-11,1> ##
```

```
      (ARC-RD)
      ## <netinfo,arc-rd,1> ##

         ---- ETC ----

   (TIPS)

      (AFWL-TIP)
      ## <netinfo,afwl-tip,1> ##

      (ALOHA-TIP)
      ## <netinfo,aloha-tip,1> ##

      (AMES-TIP)
      ## <netinfo,ames-tip,1> ##

         ---- ETC ----

(ACRONYMS)
## <pi,bigboy,1a:xrryr> ##

(PROGRAMS)
## <netinfo,PROGRAMS,1:g> ##

(PROTOCOLS)
## <NETINFO,HELP-PROTOCOLS,1> ##

(DOCUMENTATION)
## <nic,rfc-index,1> ##
```

SYSTEM R: A RELATIONAL APPROACH TO DATA BASE MANAGEMENT*

M. M. Astrahan, M. W. Blasgen, D. D. Chamberlin,
K. P. Eswaran, J. N. Gray, P. P. Griffiths, W. F. King,
R. A. Lorie, P. R. McJones, J. W. Mehl, G. R. Putzolu,
I. L. Traiger, B. W. Wade and V. Watson

IBM Research Laboratory
San Jose, California 95193

ABSTRACT: System R is a data base management system which provides a high-level relational data interface. The system provides a high level of data independence by isolating the end user as much as possible from underlying storage structures. The system permits definition of a variety of relational views on common underlying data. Data control features are provided, including authorization, integrity assertions, triggered transactions, a logging and recovery subsystem, and facilities for maintaining data consistency in a shared-update environment.

This paper contains a description of the overall architecture and design of the system. At the present time the system is being implemented and the design evaluated. We emphasize that System R is a vehicle for research in data base architecture, and is not planned as a product.

*RJ 1738 (#25356)
February 27, 1976
Computer Science

THE BERKELEY DATA-BASE MANAGEMENT SYSTEM

David R. Richards

Computer Science and Applied
Mathematics Department

Lawrence Berkeley Laboratory
University of California
Berkeley, California 94720

June 1976

ABSTRACT

The Berkeley Data-Base Management System (BDMS) is a transportable, general-purpose, hierarchical data base management and information retrieval system that has been developed at LBL. It may be used procedurally by application programs or as a stand-alone system (batch or interactive). Facilities of the latter include a simple data-base definition language and a sophisticated hierarchical data editor. The query language currently being implemented will provide multikey retrieval with boolean operators, nested parentheses, truncated and range searching, and the ability to save an intermediate search result and use it in a subsequent query.

The system is designed to support numerical as well as character data and so is especially well adapted to scientific applications. Hooks are provided for the attachment of user processors for input data validation, input and output editing, key transformation, and composite key generation. The user controls the degree of data base inversion (indexing), and all indices are mentioned dynamically, i.e., any change to the data base is reflected immediately in the indices.

The Data Base Management System, INGRES

by

Michael Stonebraker

Electronics Research Laboratory

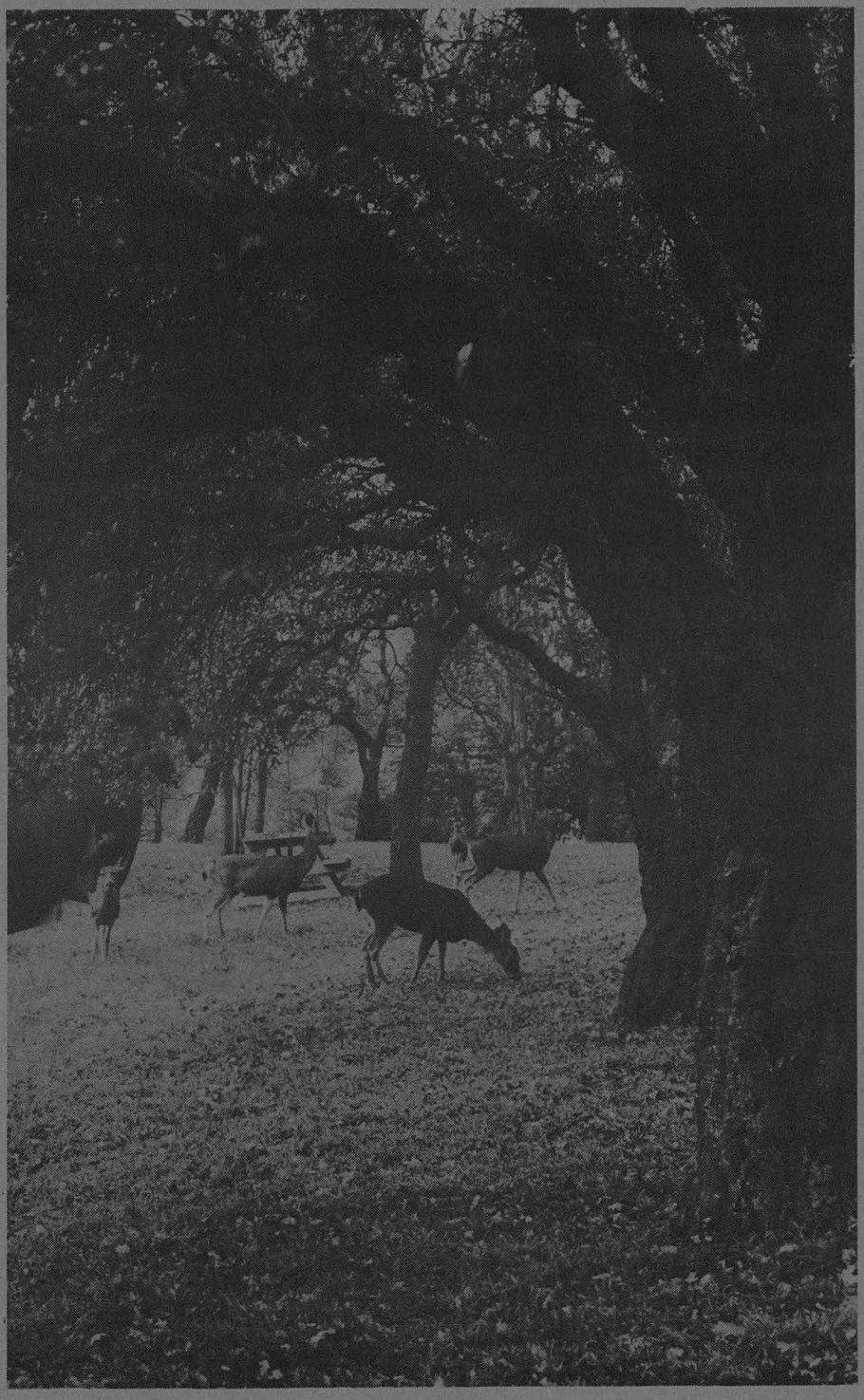
and

Lawrence Berkeley Laboratory

University of California

ABSTRACT

This paper briefly presents the capabilities of the data base system, INGRES. This multi-user system gives a relational view of data, supports two high level non-procedural languages and runs as a collection of user processes on top of the UNIX operating system for PDP 11/40, 11/45 and 11/70 computers. Besides current facilities, further directions and desired features are suggested.



VII. DATA MANAGEMENT MACHINES

ARCHITECTURE OF DATA BASE ORIENTED SYSTEMS

H. Baskin, H. Holmes, R. Freitas, P. Stoffel

Lawrence Berkeley Laboratory
University of California
Berkeley, California 94720

June, 1976

ABSTRACT

We shall present several new concepts which we envision will become the dominant pattern for the organization and design of data base oriented systems.

Current developments and trends in technology have brought us to a point where the majority of the processing related to typical data base applications can be performed adequately and rapidly using inexpensive processors. This being the case, we expect that the use of such a processor imbedded in a terminal will become the prevalent situation. For data base applications, such an intelligent terminal could also have a small secondary storage unit which is quite inexpensive. The primary function of the central system would, in this case, become that of the data base manager for a multitude of such semi-autonomous terminals. This data base management function would be quite different from that which current systems employ. The principle aspects of the new DBM function would include:

- supervising and organizing mass storage
- extracting subsets from the mass data base as needed
- providing access control and protection functions
- providing assured continuity of availability of the data
- assuring the integrity of the data.

In this talk we shall discuss some new problem areas and techniques related to the extraction of subsets from the full data base with emphasis on minimizing the cost of communication between the terminal and the central system. A tentative outline of some ideas for utilizing intelligent terminals in conjunction with centralized data management and mass storage facilities is described. This project is being undertaken jointly by the University of California at Berkeley and the Lawrence Berkeley Laboratory.

APPROACH

Current developments and trends in the technology have brought us to the point where terminals can be used for the majority of processing required in typical data base applications. Such a terminal would be used in conjunction with a centralized data management and mass storage facility. For these applications, such a terminal would also have a small secondary storage unit, which would be quite inexpensive.

The use of such intelligent terminals allows the user to work effectively with a small amount of data locally. The user can get highly interactive performance without a large amount of high speed processing in a central computer.

Programming for such a terminal would be considerably simpler, since each terminal would have only a single user.

Such a terminal might include an alphanumeric CRT, a PDP-11 or some microprocessor with 32K bytes of memory and, say, dual floppy disks for the secondary storage.

The primary function of the central system would, in this case, be data-base management for a multitude of such semi-autonomous terminals. This data-base management function would be quite different from that

which current systems employ. For example, it would no longer have to emphasize highly interactive response.

The central system would be primarily responsible for data base storage, while the intelligent terminals would be primarily responsible for the user interface and query decomposition and analysis.

Issues of concern for the central system would include such things as:

- supervising and organizing mass storage
- extracting subsets from the mass data base as needed
- providing access control and protection functions
- providing assured continuity of availability of the data
- assuring the integrity of data.

Of course, these issues are also of concern in existing systems and we intend to borrow as much as possible of the relevant techniques from existing systems. Issues which pertain to this use of intelligent terminals are many; we will discuss the choice of a suitable division of labor and the reduction of communication bandwidth required between the central system and the intelligent terminal.

We believe that the most effective division of labor is one which allows the intelligent processor to be autonomous; that is, it should have sufficient processing power and local secondary storage to answer most queries without explicit attention from the central system.

The next question, then, is, given this capability, how do we exploit it to reduce the communication bandwidth required between the terminal and the central system.

We plan to investigate systems in which information about the queries

is gathered and sent to the central system for the purpose of predicting what material from the central mass storage will be required to answer future queries.

The system will then operate in a mode in which the terminals contain subsets of the central mass data base.

Then, as the user makes queries, the central system sends the requested data and tries to anticipate the user's future queries.

Figure 5 illustrates this process. As the user makes a series of queries, the central system will extract a slice from the mass store and send it to the intelligent terminal. Then, if the user starts up a new set of queries, the system will send data for the new area. Eventually, since the secondary storage at the terminal is limited, the terminal will have to discard parts of its working set.

The benefits to the central system are that the logic is potentially simpler and that the requirement for rapid response is considerably eased.

For example, the user might make from one to four requests per minute, while the time to transmit a subset of the data base to the terminal may be a few mintues to tens of minutes.

The central facility no longer needs to be an interactive facility. Each request from a terminal can be processed sequentially - simply set up the subset of the data base which is to be transmitted.

There are a multitude of opportunities and the technical problems in this scheme which we are just beginning to comprehend.

- how do you monitor transactions?
- how do you extract some "essence" from the transactions?
- how do you use this information to predict the next useful "layer" of data?

- where should the predictive facility be located?

CURRENT STATUS

For a pilot version of the system, we are using a collection of equipment which includes a PDP-11, some memory, 7 disk pack drives (2314 equivalent) and 3 META IV processors, (Figure 6). This equipment was originally built to investigate modular, redundant, highly available computer systems. Eventually, we want to exploit these charateristics to provide a highly available central facility with modular expansion capability.

The system is designed so that any module can fail or be powered down without "crashing" the system.

Not shown on this diagram is the reconfiguration logic which provides this capability.

In order to add capacity, one can simply add on more of whatever you need; for example, just attach more disk drives to the switch.

The advantage of the switch (see Figure 6) is that any processor can access any disk directly. Since there is a controller for each drive, there are no controller conflicts as there are in most systems.

The terminal we are using is a PDP-11/05 with 32K bytes of memory, a CRT display, and a disk cartridge with 2.5M bytes of storage (Figure 7).

We are using asynchronous lines for communcations. We plan to run them at 9600 baud.

For the initial software, we plan to use BDMS - Berkeley Database Management System - a locally written system which also runs on the Computer Center hardware.

We can capitalize on the local experience with this system and perhaps even use the same data. This would allow us to make precise

comparisons of the distributed system versus the timesharing approach.

We will first run BDMS as a stand-alone system in the central facility and simply use the terminal in a non-intelligent way.

As we gain experience, we will move the user interface of BDMS from the central system into the terminal.

After this is done, we will instrument the systems appropriately and start trying to manage subsets of the data in the terminal.

REFERENCES:

(1) H. B. Baskin, B. R. Borgenson, and R. Roberts, "PRIME - A Modular Architecture for Terminal Oriented Systems", SJCC, vol. 40, 1972 AFIPS Press, Montvale, New Jersey, P.431.

(2) D. Richards, BDMS: Berkeley Data-Base Management System Users' Manual (Version 1.2), LBL-4683, Lawrence Berkeley Laboratory, University of California, Berkeley, California (April 1976).

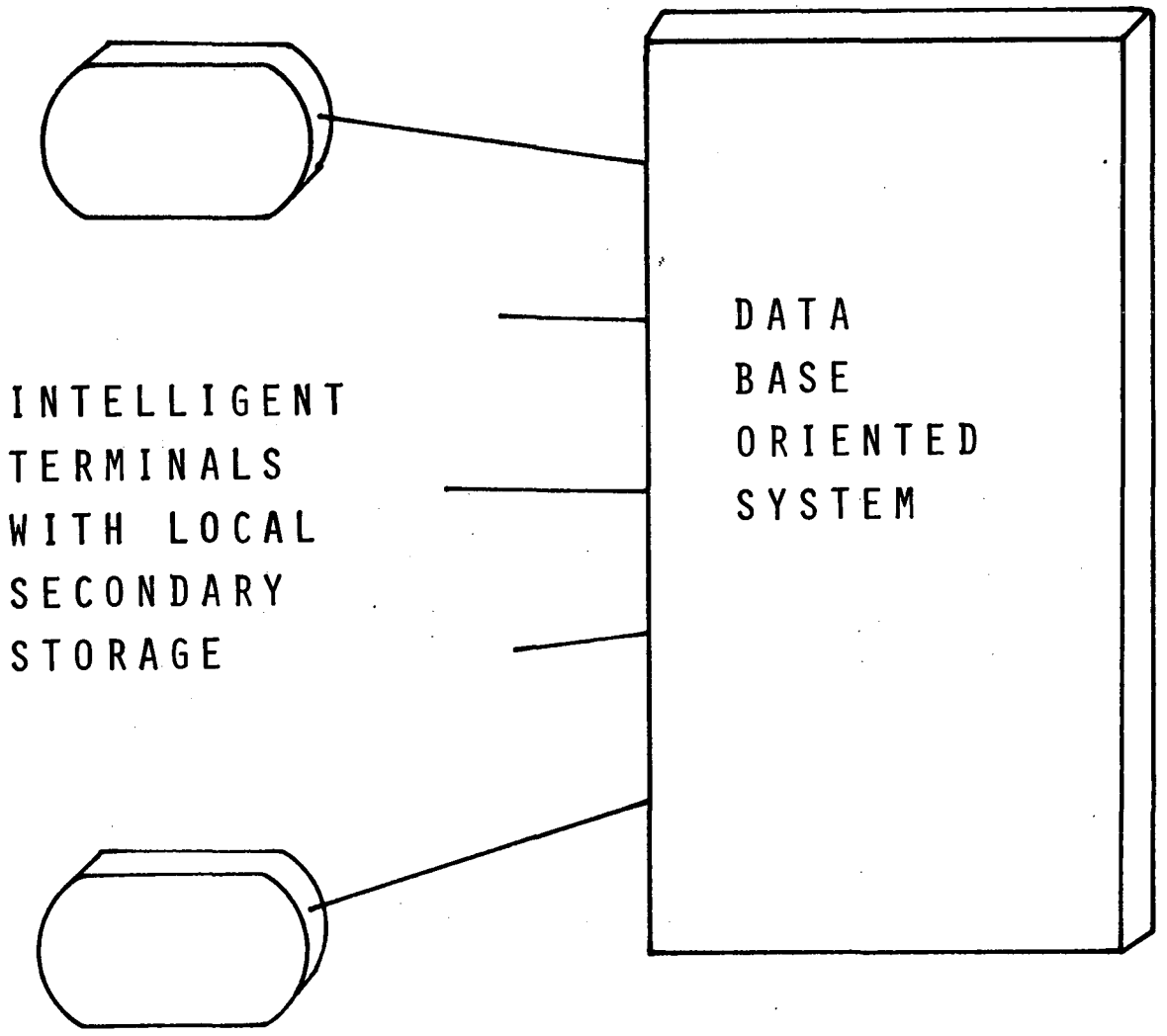


FIGURE 1. SYSTEM CONFIGURATION

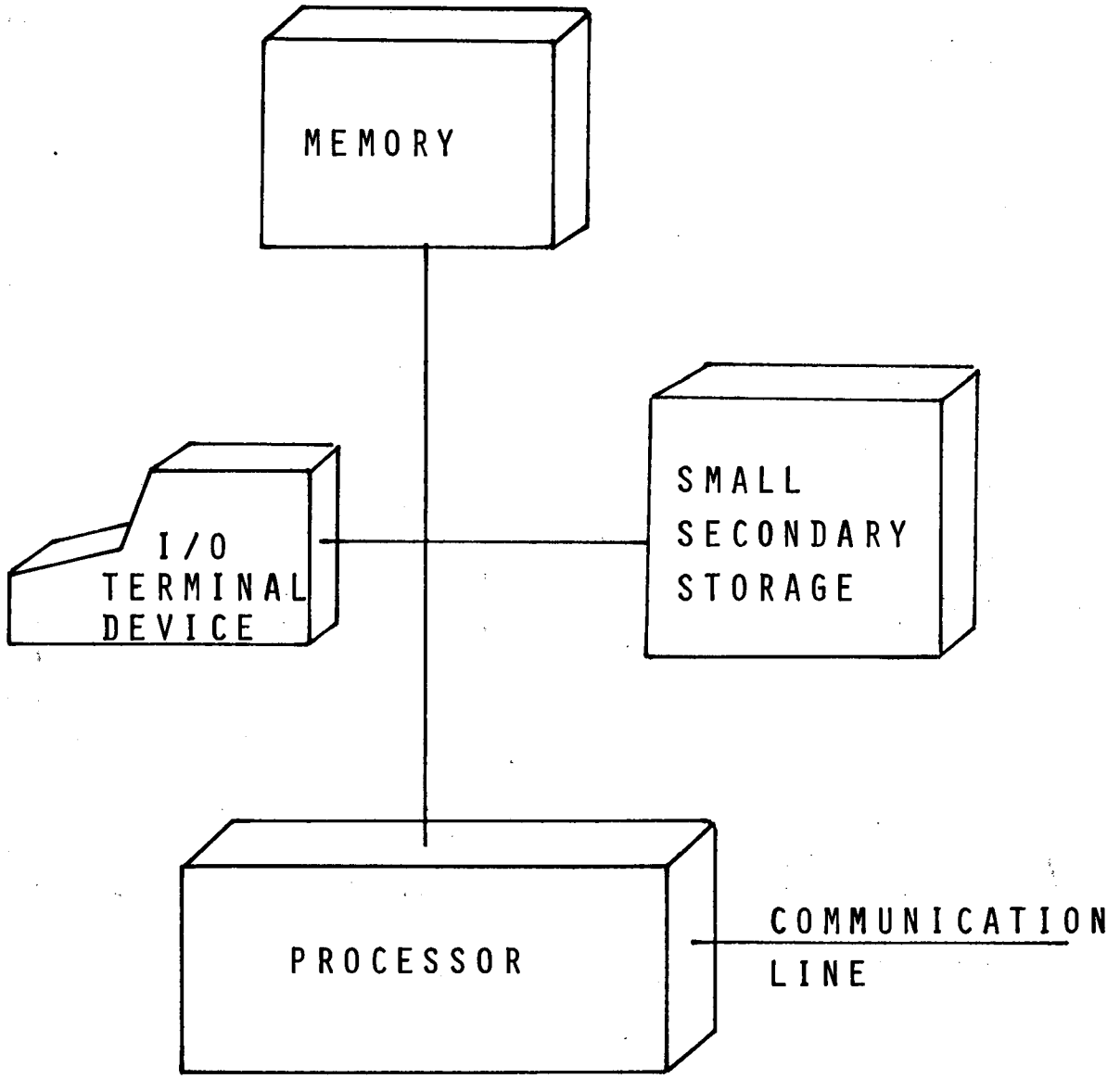


FIGURE 2. INTELLIGENT TERMINAL

CENTRALIZED FUNCTIONS

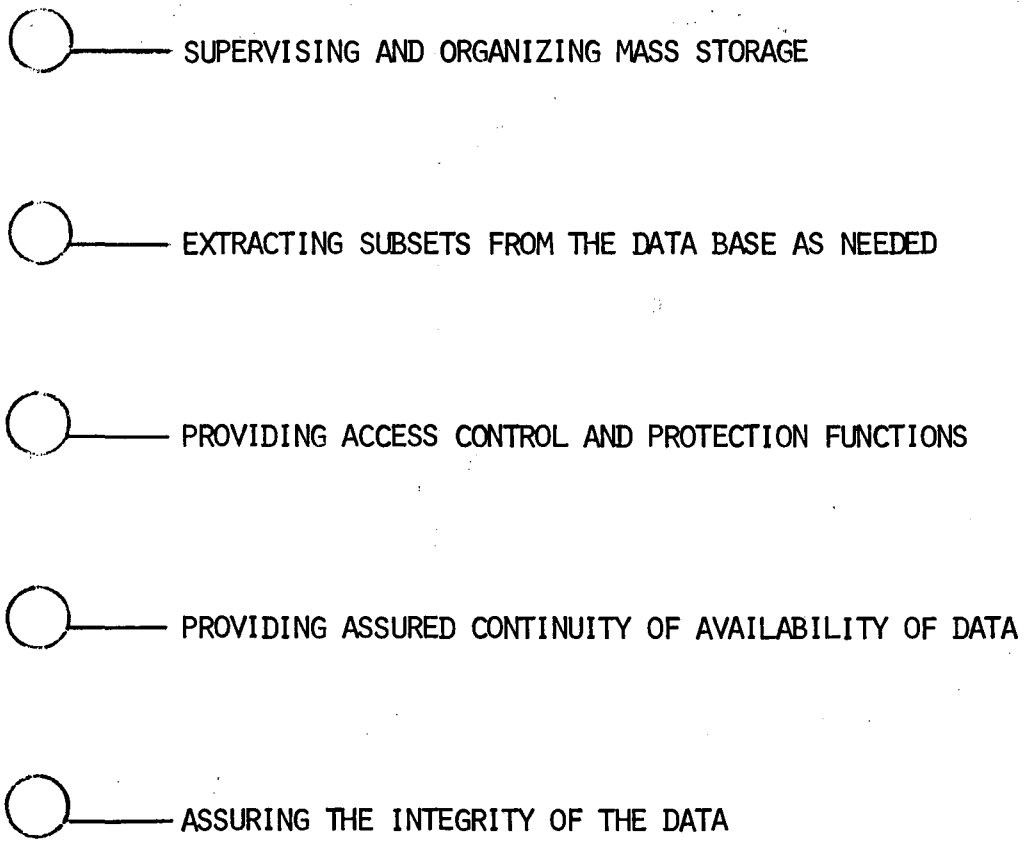


FIGURE 3.

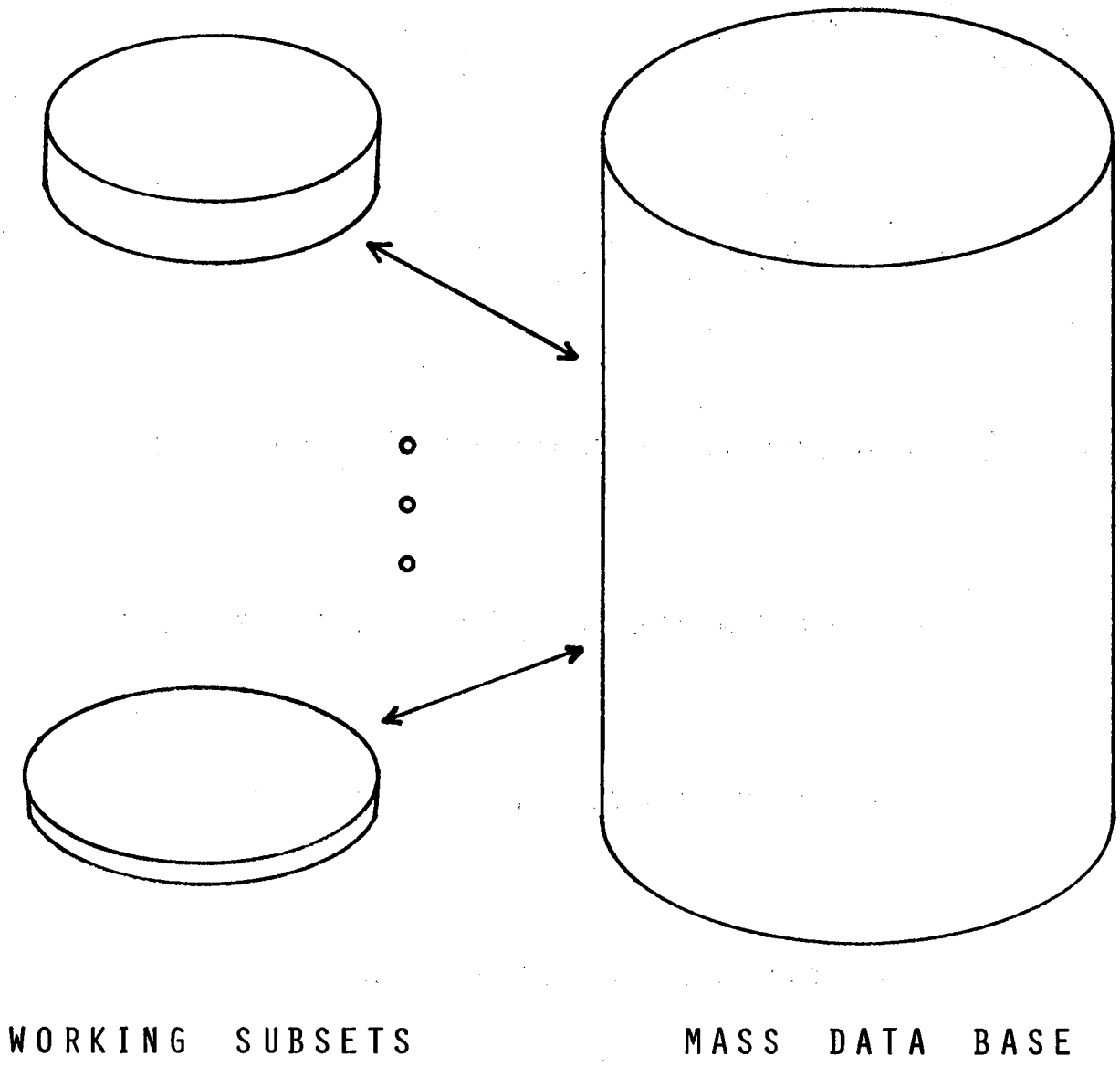


FIGURE 4. WORKING SUBSETS ARE PROVIDED BY THE CENTRAL SYSTEM

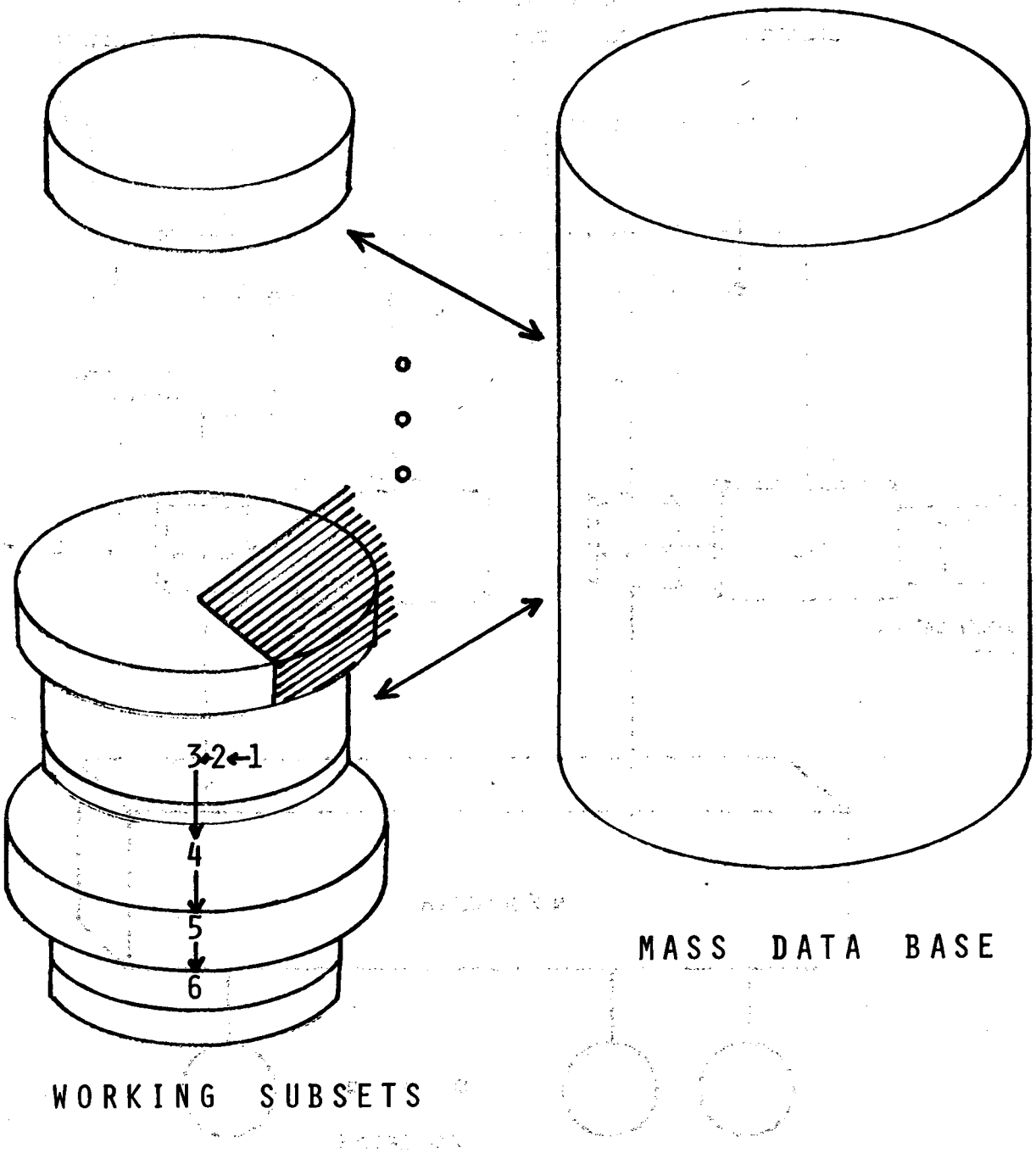


FIGURE 5. WORKING SUBSETS ARE DYNAMICALLY EXTENDED

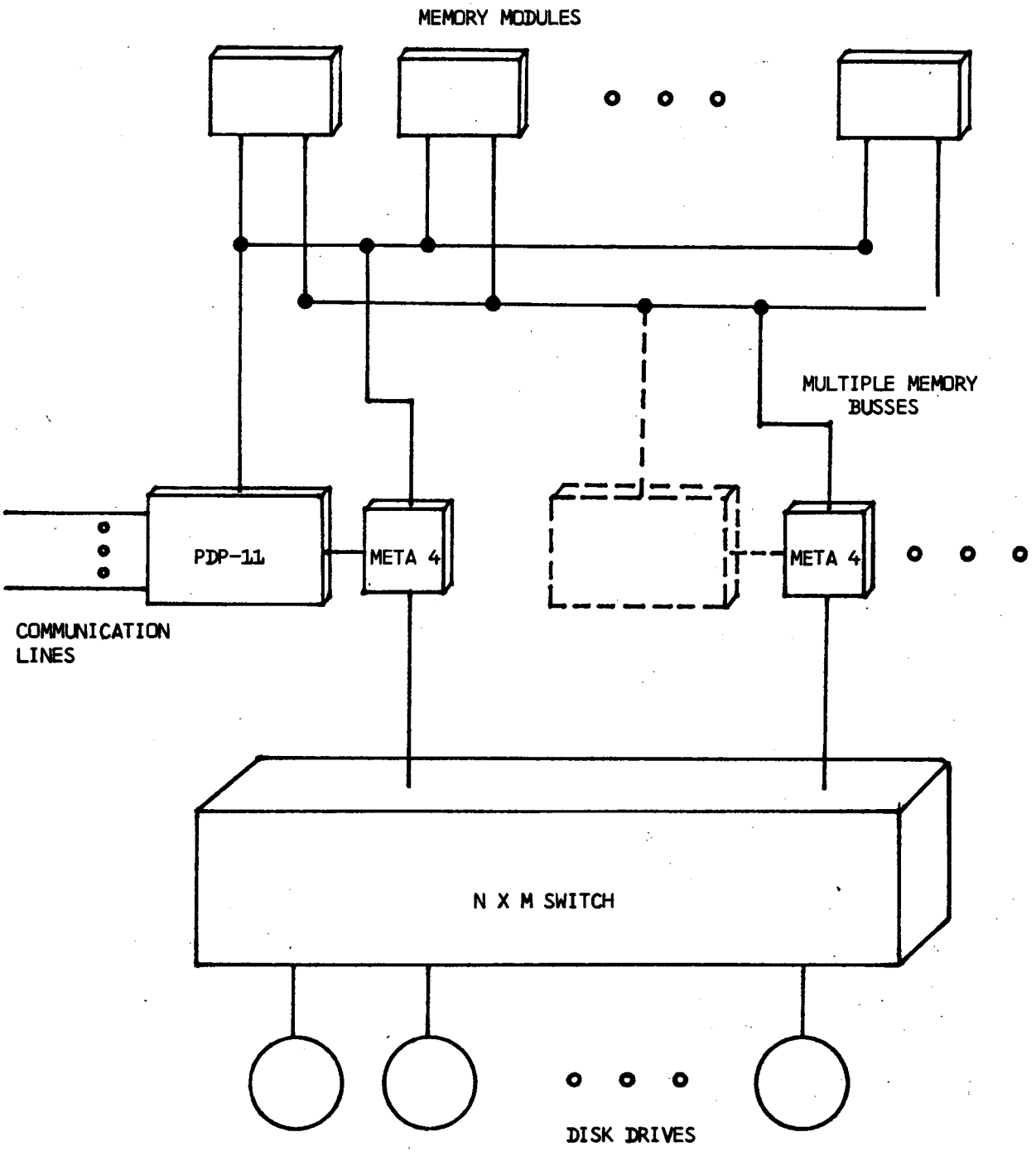


FIGURE 6. CENTRAL SYSTEM HARDWARE

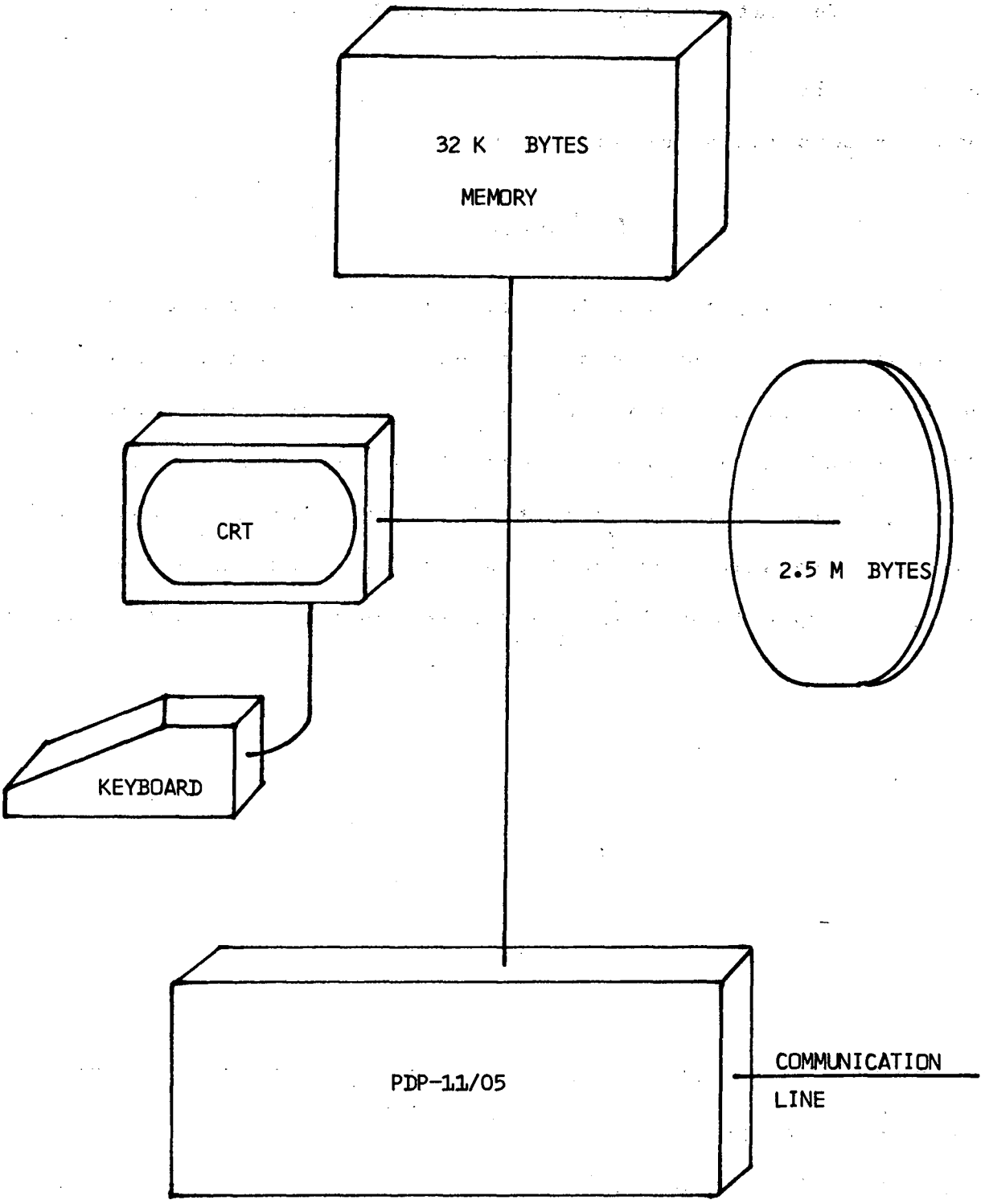


FIGURE 7. TERMINAL HARDWARE

The Datacomputer -- a Network Data Utility

Jerry Farrell

Computer Corporation of America (1)

Abstract

The Datacomputer provides data storage and management to a community of users on a computer network. It offers the capacity and economy of a trillion bit mass store, and the conversion facilities and flexible access controls necessary to support data sharing among processes on multiple, dissimilar machines. It has been in operation on the Arpanet for the past three years, and currently supports well over a hundred users on a dozen machines.

---

(1) Research reported in this paper was supported by the Advanced Research Projects Agency of the Department of Defense under Contract No. MDA 903-74-C-0225.

author's address: Computer Corporation of America, 575 Technology Square, Cambridge, MA 02139; SNDMSG JF@CCA

Datacomputer Characteristics

The Datacomputer can be seen as a data management box attached to a network: User programs run at other hosts on the network; the Datacomputer runs at its own host; and they cooperate through the network -- the interface is a high-level language called Datalanguage.

The first characteristic of this box is that it contains a mass store. The Datacomputer supports a device whose on-line capacity can expand to over three trillion bits. A Terabit Memory System with a capacity near 200 billion bits is about to start acceptance testing at CCA. In addition to its enormous capacity, this TBM offers significant economy: capital cost for the device is on the order of one dollar per million bits. When we add in various overhead figures, we estimate a storage charge to users on the order of one cent per month per 512 36-bit words. This compares to a charge of approximately twenty-five cents per month for the same amount of data on a well-known current time-sharing system.

Beyond the TBM, the Datacomputer provides a data management system, designed especially for its function as a generalized network utility. It is accessible through the Arpanet, using a standard Host-Host protocol. Data can be shared among machines at any point on the net. New information in the data base is immediately available to all the authorized users of that data. Conversion facilities are provided to translate among different formats and machine representations, so that dissimilar machines can use the same information. Flexible access controls allow a database administrator to specify the bounds of legitimate usage conveniently, and enforce safeguards against improper accesses.

Data management functions are performed in response to commands and requests expressed in a high-level language -- Datalanguage. That is, the control interface consists of passing character strings back and forth. This interface is the same, regardless of the user's host machine. Coupled with the conversion facilities mentioned above, it allows the Datacomputer to service users on many different kinds of machines -- different locations, different sizes, different manufacturers and architectures.

The network environment has a strong influence on the nature of Datalanguage. Communication between processes in the the network must be at a much lower bandwidth than the standard DBMS pattern of procedure calls at the record level in a single machine. Consequently, we are forced to define Datalanguage at a high semantic level: users must specify much more what their

desired result is, and leave the Datacomputer to produce that result autonomously. This has advantages: it allows the user to treat the Datacomputer's processing as primitive, ignoring many details of how a desired result is produced. It also allows the Datacomputer to perform global optimizations on the user's request, being very selective about what data are accessed, what are transferred, and what processing is done internally to the system.

The isolation of the Datacomputer provides another advantage: as I mentioned, access is restricted to messages passed to the Datacomputer over the network interface. This means there are no user processes running on the Datacomputer; as a trivial corollary, there are no hostile user programs. User processing must be specified at a high level relative to the security implementation, and is restricted to operations expressible in Datalanguage. All this gives a significant advantage in providing secure access controls.

Datalanguage Example

I think that's enough to give an idea of what the Datacomputer's about; now I'd like to turn to how it's used. As I've just explained, this means essentially explaining the uses of Datalanguage. To do this, I'll follow one example through a number of kinds of processing.

Datalanguage serves three broad purposes: administration, description, and manipulation.

Administrative Datalanguage

Administrative uses of Datalanguage are illustrated in the first figure. They include creating a directory structure, defining privileges and allocating resources, and monitoring usage. Most of this activity is connected in one way or another to the Datacomputer directory. The directory is tree-structured. Access control and allocation information is attached to nodes at any level in the tree. User identification is at least in part tied to the directory, although users are also identified by their Host and ICP socket in the network. Note that, in general, logging in to a node does not automatically grant any special privileges with respect to data stored under that node.

Administrative Datalanguage

```
LOGIN CCA.JF; CREATE LIBRARY,M=25;
CREATEP LIBRARY,H=31,S=13303816,G=CLRWA;
CREATEP LIBRARY,D=RWA;

LOGIN LIBRARY; CREATE LOCAL; CREATE FOREIGN; CREATE DATA;
CREATEP LOCAL,H=31,G=L; CREATEP FOREIGN,P='READMORE',G=L;
CREATEP DATA,U=CCA.JF.LIBRARY,G=RWA;
CREATEP DATA,U=CCA.JF.LIBRARY.LOCAL,G=RA;
CREATEP DATA,U=CCA.JF.LIBRARY.FOREIGN,G=R;
CREATEP DATA,D=RWA;
```

Figure 1

Here, I have logged in to an existing node in the Datacomputer, and now build myself a library system under it. The first step creates a new subtree in the directory, allocating it 25 megabits of storage. I then specify that I may login to this node (I am identified by a particular socket number which corresponds to my directory at CCA's TENEX), and have full privileges; everyone else is excluded at this level. Next I create a pair of nodes via which my friends may also access the library, and one under which I will actually store the data.

On this structure, I attach privilege blocks which specify how much sharing I want to do. Note that I have all privileges: Control, Login, Read, Write, and Append. Other people are allowed to add information if they are coming from my own host, but not to change existing information. Foreign freinds are restricted to viewing what is there, and only if they know the password. These restrictions could have been applied with a less complicated directory structure, but this structure allows me to monitor resource utilization according to my categories.

Descriptive Datalanguage

The next function of Datalanguage is description. This covers both logical structure and physical format, and it occurs at two separate levels.

The first is the file description; this is the description of the data as it is stored in the Datacomputer. The second is the port description; this refers to the data as it is trans-

ferred across the network interface, into or out of the Datacomputer. There is no necessary correspondence between port and file descriptions; one port may be used for many files, or may require data from several files at once. Different port descriptions may be used to select different subsets from a file or set of files. Machines with different characteristics, such as word size or character sets, may use corresponding ports to access the same data in a uniform manner; the Datacomputer uses the physical aspects of the descriptions to perform necessary conversions as data is transferred.

The logical structure of data in the Datacomputer is fairly simple. All data is hierarchical. There are six data types: LIST, STRUCTURE, STRING, INTEGER, and BYTE. LISTs and STRUCTUREs are aggregate containers; they hold one or more subcontainers, which may in turn be LISTs or STRUCTUREs, or one of the elementary containers. The elementary containers, STRING, INTEGER, and BYTE, hold data which cannot be further decomposed. LISTs contain multiple occurrences of a single subcontainer; STRUCTUREs collect several subcontainers, one occurrence of each, into a single container.

Figure 2 illustrates the use of Datalanguage to describe data. The first step is to describe a file which will hold the catalog for the library. This is a list of up to 10,000 books. Each book is described as a structure containing a title, possible subtitles, authors, publisher, publication date, length and subjects. Fields not yet mentioned have special purposes which

Data Descriptions

```
CREATE DATA.BOOKS FILE LIST (,500,10000)
    BOOK STRUCTURE
        TC INTEGER
        TITLE STRING (,50,500),C=TC,I=D
        SUBTITLES LIST (,1,10),C=1
            SUBTITLE STRING (,100),C=1
        AUTHORS LIST (,1,10),C=1
            AUTHOR STRING (,60),C=1
        PUBLISHER STRING (,30,100),C=1
        DATEPUBLISHED INTEGER(4)
        PAGES INTEGER (,3,5),C=1
        SUBJECTS LIST (,1,10),C=1
            SUBJECT STRING (,25),C=1,I=I
        KEYAUTHORS LIST (,1,5),C=1
            AUTHOR STRING (,25),C=1,I=I
    END;

CREATE DATA.IMAGEPORT PORT LIKE BOOKS;

CREATE DATA.ASCIIPORT PORT LIST,P=EOF
    BOOK STRUCTURE
        TITLE STRING (,132),P=EOR
        SUBTITLES LIST (,10),P=EOB
            SUBTITLE STRING (,100),P=EOR
        AUTHORS LIST (,10),P=EOB
            AUTHOR STRING (,60),P=EOR
        PUBLISHER STRING (,50),D=' '
        DATEPUBLISHED INTEGER (4),D=','
        PAGES INTEGER (,5),D=' '
        FILL STRING (2),F='P',P=EOR
        SUBJECTS LIST (,10),P=EOB
            SUBJECT STRING (,25),P=EOR
    END;

CREATE DATA.TITLEPORT PORT LIST, P=EOF
    BOOK STRUCTURE
        TITLE STRING (,132),P=EOR
        SUBTITLES LIST (,10),P=EOB
            SUBTITLE STRING (,100),P=EOR
    END;

CREATE DATA.AUTHORPORT PORT LIST,P=EOF
    BOOK STRUCTURE
        AUTHOR STRING (,25),P=EOR
    END;
```

Figure 2

will be described shortly. Items that have a characteristic length, such as lists, strings, and integers expressed as strings of digits, must be dimensioned. If that length is not fixed, there must also be some termination specified, either a count or a recognizable end marker. Here, TITLE is declared to be a string of 0 to 500 characters; its average length is declared to be 50 for space allocation purposes. The length of the title of each book is stored in a separate variable, TC, as a binary integer. The number of authors may range from 1 to 10, with a 1-byte count kept at the head of the data, as indicated by the "C=1".

Several fields are considered likely candidates for selection criteria, and so are marked for inversion: TITLE, SUBJECT, and KEYAUTHORs. (This latter field allows books with many authors to be indexed only under principal writers, rather than every member of the AUTHORs list.) As data is stored in the file, the Datacomputer will generate an auxiliary database, called an inversion. For each value of each inverted field, this inversion will contain the location of every record in the file which has that particular field/value combination.

Next I create a port with exactly the same description, to be used for efficient transfers of large volumes of data. Features of the description which are not meaningful for ports, such as inverted fields, are ignored.

Next I create another port which will be used for viewing selected book descriptions at a terminal. All the fields are described as ASCII strings here, and the various counts have been replaced by punctuation. The Datacomputer recognizes three levels of punctuation in ASCII data: EOR, which means carriage-return / linefeed or other new line; EOB which means formfeed; and EOF, or End-of-File.

Other ports are created to transfer only certain fields for selected records; one to get (or give) the authors, and one for titles.

Datalanguage for Data Manipulation

Figure 3 illustrates the third function of Datalanguage, Data manipulation. There are two preliminary statements: the first specifies that inversion information is to be collected and processed in a more efficient batch manner; the second specifies a network connection which the Datacomputer should use for transfers involving IMAGEPORT, the port named in the statement.

Then, data is assigned to the file through IMAGEPORT. The Datacomputer attempts to open the network connection specified earlier; the user process should cooperate (or cause another process to cooperate) by opening its side. The Datacomputer then reads data through that connection until it is closed, parsing it according to the port description; once again, there must be a cooperating process on the other side of the net which feeds the data into that connection, and closes it when it is finished.

Data Manipulation

```
MODE BOOKS WRITE DEFER;

CONNECT IMAGEPORT TO 13303820;

BOOKS=IMAGEPORT;

FOR QUERY IN AUTHORPORT
    BEGIN DECLARE I INTEGER  I=0
        FOR TITLEPORT, BOOKS WITH ANY KEYAUTHORS
                                  WITH AUTHOR = QUERY.AUTHOR
            BEGIN
                BOOK=BOOK
                I=I+1
            END
        COMMENT
 QUERY.AUTHOR ! ' HAS ' ! I ! ' BOOK(S) IN THE LIBRARY.'
    END;
```

Figure 3

The second request is more complicated. A FOR statement is used to apply another statement to selected members of a list, possibly generating matching members in a second, output list. In this case, a list of author's names is fed to the Datacomputer through AUTHORPORT. The context variable QUERY is used instead of the actual field name in the port (BOOK), since that would be ambiguous later in request. Then, for every author's name received, all members of the BOOKS file are selected which have at least one key author the same as this query author. The book is counted, and its title transferred out of the Datacomputer through TITLEPORT. When all the books with a given author have been processed, a message with the count for that author is generated. The process is repeated for the next author found in AUTHORPORT, and so until the list of authors is exhausted. Since neither of the ports used in this request have been the object of

a CONNECT statement, they each transfer their data over the network connections which carry Datalanguage and the responding Datacomputer messages.

Status

The Datacomputer has been available on the Arpanet for about three years using disk storage only; a TBM mass store was delivered in February and is undergoing acceptance testing now. We have been through several preliminary releases of the Datacomputer, and the first full release, Version 1, came out last August. A second version incorporating the TBM and other enhancements should come out this summer.

There are a number of applications currently using the Datacomputer; I would like to mention two of them briefly, plus another which is under intensive development right now.

A system called DFTP, which runs under the TOPS-10, TENEX, and ITS operating systems, available at 11 PDP-10 sites on the Arpanet, providing file archival services to more than a hundred users.

A survey of the status of all nodes and hosts on the Arpanet, conducted by a program at MIT's Laboratory for Computer Science, is stored in the Datacomputer. A program at MIT attempts to connect to each site surveyed every 20 minutes, and records the result and response time. The surveys for a single day (about 7200) are transferred to the Datacomputer every night.

Several systems, at MIT and elsewhere, access this data to provide historical surveys of service on the net.

Finally, in an application currently under development, the Datacomputer will act as the repository for a large database of seismic information collected from sensors around the world, and fed through the Arpanet in real time to the Datacomputer site at rates up to 20 kilobaud around the clock. This data will be processed into various files in the Datacomputer, and made available to authorized researchers around the net.

Summary

The Datacomputer is a large-scale network data utility. It offers the capacity and economy of a trillion bit mass store. It is designed to service a community of users on a computer network, working from distant and dissimilar machines, with a uniform, high-level interface. It supports controlled data sharing via extensive selection and conversion facilities, network-wide accessibility, flexible access controls, and the improved security and high semantic level of a dedicated data management machine.

THE CASE FOR A PARALLEL-ASSOCIATIVE APPROACH TO DATA BASE MACHINE ARCHITECTURES

S.A. Schuster, E.A. Ozkarahan, and K.C. Smith
Computer Systems Research Group
University of Toronto
May 1976

ABSTRACT

Various performance requirements for generalized data base management systems (GDBMS) are reviewed. The conventional approach of using software and serial computer to implement GDBMS is outlined. Several problems are discussed which arise in using the conventional approach to meet GDBMS requirements. An overview of the parallel-associative processor design philosophy for data base machines is presented. The "Relational Associative Processor (RAP)" data base machine is surveyed. It is argued that RAP-like machines are a viable approach to meeting GDBMS requirements especially in light of the demands anticipated by distributed data base systems.

DATA BASE MANAGEMENT SYSTEMS

Current requirements for data bases necessitate the implementation of data base management systems. Data base management systems aim at creating data bases that have an existence separate from the potential applications that will use it. These systems are complex integrations of computer software and hardware which attempt to provide their users (1) a "logical view" of a data base distinct from the details of its computer storage and manipulation and providing a query language by which users can specify the retrievals and updates of the data stored according to the view.

The full potential of data bases will be realized only if three important requirements are met by data base management system implementations. First, the languages provided to users must be sufficiently user-oriented and powerful to permit simple specifications of the desired data manipulation. That is, a user must only be required to write few statements to cause the execution of complex queries. Such languages allow users to specify manipulations in a set-oriented fashion (e.g., retrieve or update all...) and to associatively address the data to manipulated by its content (e.g.,... all employees who work in the future department) rather than the data's hardware location.

Secondly, queries must be satisfied within fast response time limits. Data base systems will be required to operate within on-line concurrent user environments which support users at terminals, batch application programs running within multi-tasking systems, and communication systems through distributed computer networks. It is important to note that query (update and retrieval) execution time will have an significant impact on the feasibility of distributed data base systems.

The third requirement has to do with the technical administration of data base systems. The separation of the physical data from its users causes the responsibility for efficient performance to be transfered from the user to the administrator (called the Data Base Administrator) of the system. The responsibility for system tuning was originally distributed over several users and file processing programming systems. Today, the data base administrator must make decisions, often conflicting with individual user requirements, for all the users. Present techniques used in data base systems makes this job extremely complex and contributes greatly to the high cost of using data bases. It is imperative that this problem be alleviated so that the tuning of data bases systems can be accomplished more effectively and easily.

Several efficiency and reliability problems arise when using conventional computers to implement modern data base management systems. We will discuss some of these problems shortly. The University of Toronto's project RAP (a Relational Associative Processor) is aimed at providing new computer architectures for solving these problems.

USING CONVENTIONAL COMPUTERS FOR DATA MANAGEMENT

The problems of implementing data base systems on conventional Von Nueman computers arise because of the processing and addressing structure of these computers. Today's machines are designed to process data sequentially (i.e., the execution of one instruction on a single data item at a time) and that accessing of the data to be processed is accomplished by specifying its location or hardware address. Conventional machines require users to indicate an operation repetatively or sequentially for each item separately.

Conventional machines lack the parallel processing and associative addressing architecture required by modern data bases. This causes implementors to simulate this architecture through software. Accordingly, to implement efficient access to a modern data base, large complex programs must be written to (1) map the users view into the physical reality of the machine and (2) to provide access paths to permit fast location of arbitrary portions of the data base. Access paths are extra data and/or sorting strategies which "index" the original data base by providing a more direct access to a specific item's storage location. Additional software must be provided to utilize and maintain the access paths.

Today's machine architectures cause the need for access paths and logical to physical mappings. These can only be provided by extra software which places a tremendous overhead on computer systems. The result is that general purpose data base management systems, which must meet the requirements of many applications

and broad user environments, perform poorly in practice. A few of the problems caused by the use of access paths and need for mappings in conventional computer data base management systems follow:

1) Unbalanced Performance

   Access paths provide fast retrieval of data at the expense of slower updates. This happens because updates to the data base must also be reflected in the access paths. The data of the index to the actual data base must be retrieved, changed, and rewritten and/or data must be resorted as well as the actual data base modification that is required.

2) Poor Generalized Performance

   The state-of-the-art in access path techniques does not permit sufficiently tunable generalized systems that respond to widely varying applications and user environments. Generalized systems relying on software may always exhibit poor performance when compared with specially designed single application systems. On the other hand, single application systems are expensive, time consuming to implement, and do not meet the requirements of modern data bases.

3) Unmastered Complexity

   The programs and data structures associated with the processing and concurrent user synchronization of access paths and the mapping of the users logical view to a conventional machine architecture are among the most complex in computer science. This has caused the creation and administration of data base systems to be extremely expensive.

The above problems must be solved if the full potentials of data bases are to be realized. A major consequence is that only those questions whose formulation were preconceived can be conveniently answered. All other "novel"

queries become dramatically more difficult and expensive to answer. Recent developments in micro-electronic technology can be utilized to provide such a solution.

THE RAP APPROACH TO DATA MANAGEMENT

The solution to the problems of providing data base management systems is to eliminate the need for access paths and mappings. This can be done by developing new computers whose architectures utilize many processors and memories in parallel and which address data associatively. This approach enables a large data base to be represented directly in the memory and processing architecture of the computer. The data is then divided into smaller, more processable segments and distributed across many processors to be processed in parallel. Associativity helps eliminate access paths and logical to physical mappings. Parallelism causes associative addressing and set-oriented processing to be performed at high speeds.

The associative processor RAP being designed and implemented at the University of Toronto is based on the above principles. RAP is designed to augment a conventional computer in order to implement efficient data base management systems. The design employs a set of parallel processors called cells which address data associatively. A statistical set function unit is provided to calculate summary statistics. The cells and set function unit are driven by a central controller. This organization is shown in figure 1. Each cell is composed of a microprocessor specially designed for data management operations and a sequential circulating memory (e.g., a track of a drum or disk, CCD, bubble memory, etc.). Each data base operation is executed in parallel within the cells which operate directly on the data as it circulates through the cell processors.

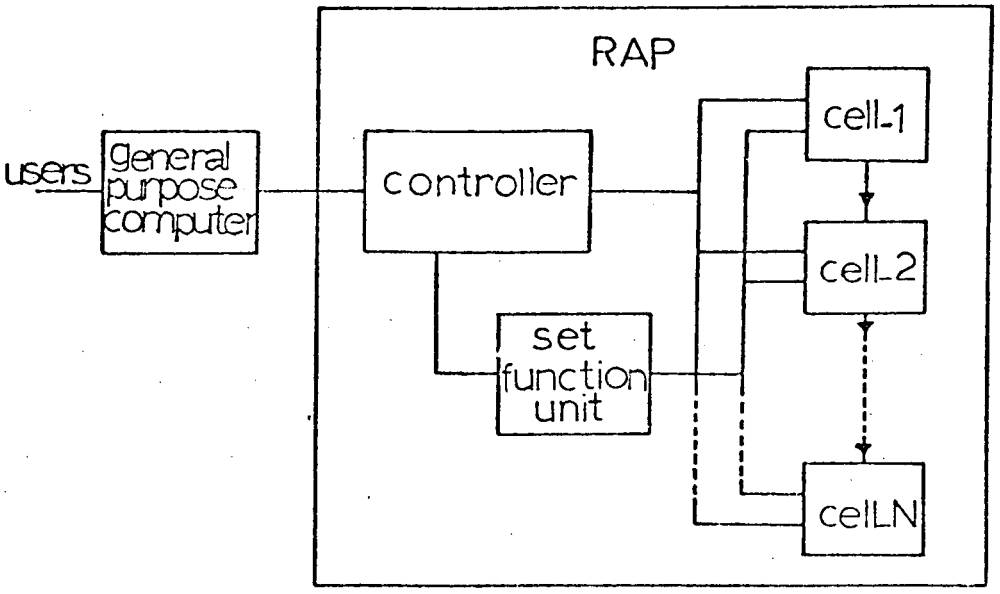


Figure 1. Overview of RAP Architecture

RAP provides an intermediate-level view of data and a collection of set-oriented instructions implemented entirely by hardware. Their design is sufficiently general to support set-oriented operations on the commonly desired high-level user views of data: hierarchical, network, and relational.

RAP stores data as unordered occurrences of records defined by a record-type. A record-type is a "template" which formats the set of record occurrences. The format of the record occurrences is defined by naming the data items whose values occur in each record. The occurrences of a record-type stores data about a set of similar entities. The name of a record-type indentifies the set entities. The names of data items identify the attributes which characterize the entity. The length of each item is fixed according to a users choice of one of three sizes. Each occurrence of a record type stores data which describes a particular entity by assigning a value to each of the items according to the format of the record-type.

Each record-type and occurrence is augmented by four special one-bit items called mark-bits. These items can be set to 0 or 1 under user control of various marking instructions or by the intermediate operations of other hardware instructions. The bits are mostly used to temporarily indicate subsets of record occurrences so that the results of one instruction can be used in subsequent instructions by treating the mark bits as data to be tested for associative addressing.

A RAP instruction reflects the general structure of a data base query. The format for each instruction is:

<OPCODE>(<MARK-BITS>)[<OBJECT>:<QUALIFICATION>]

The OPCODE specifies the operation to be performed, MARK-BIT specifies which mark-bits are to be set, OBJECT is the record type name and list of items to be manipulated, and the QUALIFICATION is a Boolean predicate of conditions on

mark-bits and item contents that select by associatively addressing which record occurrences are to be manipulated. The instructions fall into the following categories according to the type of data manipulation the opcode performs:

a) Selection:

used in process of locating and marking subsets of record occurrences. Several opcodes allow mark bits to be set in the occurrences of one record-type based on values that occur in records of a different record-type by associating different records to each other through items with identical value types.

b) Retrieval:

used in the transferring of selected items and occurrences of a record-type or values computed by the statistical processor.

c) Update:

used to modify values of the items in selected record occurrences by direct replacement or through numeric computation.

d) Statistics:

used to compute common statistical quantities such as the COUNT, SUM, AVERAGE, MAXIMUM, or MINIMUM of the values occurring in a item of selected record occurrences.

e) Schema:

used to create and destroy the definitions of record-types.

f) Insertion and Deletion:

used to insert one or more occurrences of a record-type or to delete selected occurrences.

g) Branching:

used to test various conditions and alter accordingly the sequence of instructions to be executed.

The processor has been designed to close the gap between the user's logical view of data and the way it is represented on the storage of the computer which supports the processing of the data. This will allow data base management system applications to be implemented more quickly because the implementor will no longer be concerned with the details of representing and searching a data base. Because the device more closely represents a data base management system, data base queries can often be formulated from just a few instructions and often only a single instruction is sufficient. The RAP system is designed to execute the most important instructions within one parallel rotation of the cell memories (estimated to be 50 milliseconds for a cell capacity of 0.5×10^6 bits). Another important feature is that the RAP instruction set is sufficiently complete so that queries can be implemented entirely within the RAP processor.

Studies have been conducted to compare the hypothetical performance of using RAP relative to using a conventional computer system for implementing a relational data base. Both approaches were modelled analytically. The models considered resident data bases for RAP and fast access paths in the form of inverted lists for the conventional system. The results show that significant gains in query execution speed can be achieved by the RAP architecture over the conventional system. This study indicates that, under many circumstances, on-line retrievals and updates of large data bases may only be possible with the use of RAP-like systems.

PROJECT RAP

Project RAP is primarily being funded by the Federal Departments of Communication and Supply and Services. It is also supported in part by the National Research Council. The initial grant for the first phase is $140,000.00 for a period of one year. The project is being conducted by

members of the University of Toronto's Computer Systems Research Group which is jointly administered by the Departments of Computer Science and Electrical Engineering.

During the first phase of the project an experimental version of the RAP computer will be built to prove the feasibility of the concept. Software implications are also being studied. Another goal is to identify and interest potential users and manufacturers of computers to pursue a full scale prototype of the system. This could ultimately lead to a marketable product.

The experimental version of RAP will consist of two cells with CCD memories and a controller capable of executing a third to half of the instruction set. A system is expected to be demonstratable by the fall of 1976. This will include software support for both a high-level query language and RAP assembler language interfaces.

REFERENCES

1) Ozkarahan, E.A., Schuster, S.A., and Smith, K.C., "A Data Base Processor", Technical Report CSRG-43, Computer Systems Research Group, University of Toronto, November 1974.

2) Ozkarahan, E.A., Schuster, S.A., and Smith, K.C., "RAP-An Associative Processor for Data Base Management", Proc. AFIPS 1975 NCC, Vol. 44, pp. 379-388, also to appear in 1975: Best Computer Papers, Auerbach, 1976.

3) Schuster, S.A., Ozkarahan, E.A., and Smith, K.C., "A Virtual Memory System for a Relational Associative Processor", Technical Report CSRG-64, Computer System Research Group, University of Toronto, December 1975, also to appear in Proc. AFIPS 1976 NCC, Vol. 45.

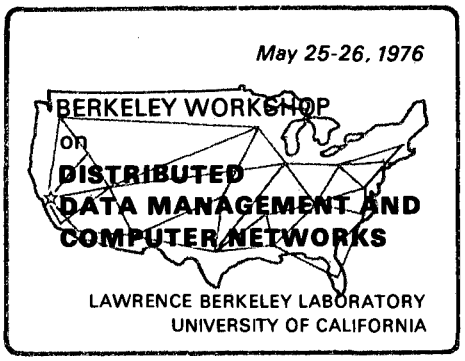
4) Ozkarahan, E.A., Schuster, S.A., and Sevcik, K.C., "Performance Evaluation of Relational Associative Processor", Technical Report CSRG-65, Computer Systems Research Group, University of Toronto, January 1976. qualifying items.

5) Ozkarahan, E.A., "An Associative Processor for Relational Data Bases-RAP", Ph.D. Thesis, Department of Computer Science, University of Toronto, January 1976.

6) Schuster, S.A., Ozkarahan, E.A., and Smith, K.C., "The RAP Associative Processor Approach to Large Data Bases", Technical Report (to appear) Computer Systems Research Group, University of Toronto, April 1976.

7) Kerschberg, L., Ozkarahan, E.A., and Pacheco, J.E.S., "A Synthetic English Query Language for a Relational Associative Processor", Technical Report (to appear) Computer Systems Research Group, University of Toronto, April 1976.

☆GPO 691-786-1976



MAY 24, 1976 MONDAY

1700 EVENING REGISTRATION

1800 RECEPTION, MARRIOTT INN HOSPITALITY ROOM

MAY 25, 1976 TUESDAY

0900 OPENING REMARKS: *Donald Austin, LBL*

WELCOMING ADDRESS: *Robert Birge, LBL*

KEYNOTE ADDRESS: *Milton Rose, ERDA/DPR*

0930 COMBINED SESSION, MARRIOTT INN BELVEDERE ROOM

AN OVERVIEW OF EXISTING DISTRIBUTED NETWORKS

The opening session will present brief overviews of computer networks in operation today. These include a general purpose distributed network, a star network for controlled thermonuclear reactor research, an educational network, a commercial timesharing network, and a commercial communications network - all of which are national or international in scope.

Gene Franklin, Stanford: EDUCOM

Peggy Karp, TELENET, Inc.: TELENET

John Killeen, LLL: The CTR Computer Network

LaRoy Tymes, TYMSHARE, Inc.: TYMNET

Stephen Walker, ARPA: ARPANET (paper read by Don Austin)

1100 BREAK

MAY 25, 1976

1115 PARALLEL SESSIONS, MARRIOTT INN

I. NETWORK GRAPHICS PROTOCOL — TREASURE ROOM

The advantages to be gained by sharing computer graphics techniques and hardware over distributed networks led to the formulation of a general graphics protocol. The problems involved, strategies evolved, and future plans for network graphics will be discussed.

James Michener, Intermetrics: The Design of the ARPA Network Graphics Protocol

Harvard Holmes, LBL: A Simple Implementation Strategy for the ARPA Network Graphics Protocol

Robert Sproull, XEROX-PARC: Network Graphics Isn't Networking

II. MINI-HOST FRONT ENDS TO THE ARPANET — ANGEL ROOM

The pioneering efforts required to implement connections to the ARPANET will be discussed from a variety of viewpoints. The insight gained from the experiences will be of interest to future network protocol design and implementation strategies.

Lawrence Amiot, ANL: Front-Ending at Argonne National Laboratory

Ed Franceschini, NYU: An ARPANET Front End for Large Computers

Gary Grossman, UI-CAC: An Alternative Front End Architecture

III. VALIDATING SOFTWARE ON DISTRIBUTED SYSTEMS — BELVEDERE ROOM

The problems inherent in producing validated software, such as ANL's classic EISPACK, will be described. A general approach will be discussed based on the National Software Works concept, and methods for implementing a universal command language for a distributed network environment will be presented.

Terry Gray, UCLA: Job Control in a Network Computing Environment

James Pool, ANL: Mathematical Software in the Network Environment

Jonathan Postel, SRI-ARC: Consistent Access to Programs

1215 LUNCHEON, MARRIOTT INN

BUSES LEAVE FOR LBL STARTING AT 1330

MAY 25, 1976

1400 COMBINED SESSION AT LBL

HIGH LEVEL PROTOCOLS — LBL AUDITORIUM

High level protocols are designed to render the network environment transparent to the user. Several approaches are being investigated to provide a more useful and useable distributed network environment by careful design of sophisticated process-to-process protocols.

James Donnelley, LLL: Extendable Communication Protocols

Gary Grossman, UI-CAC: A Host-to-Front End Protocol

Jonathan Postel, SRI-ARC: Frontend-Backend Split Programs

James White, USC-ISI: A Network Wide Virtual Programming Environment

1530 BREAK. TOURS OF THE LBL COMPUTER CENTER AND THE COMPUTER GRAPHICS RESEARCH LAB WILL BE CONDUCTED FOR INTERESTED PARTIES. SEVERAL GRAPHICS AND ALPHANUMERIC TERMINALS WILL BE AVAILABLE FOR DEMONSTRATIONS.

1600 COMBINED SESSION AT LBL

DISTRIBUTED DATA BASES — LBL AUDITORIUM

The ideas behind distributed data bases have found applications ranging from commercial banking systems to large socio-economic data bases used by the ERDA national laboratories. The difficult problems in maintaining and accessing these data bases, and the considerable advantages to be gained from distributed data bases will be discussed.

Peter Alsberg, UI-CAC: Data Distribution Strategies

John McCarthy, Stanford: Describing Other People's Files

Arie Shoshani, SDC: Common Standards for Distributed Data Bases

Michael Stonebraker, UCB/LBL: Proposal for a Network INGRES

1730 WINE AND CHEESE TASTING EVENT, LBL CAFETERIA.

BUSES FROM LBL TO THE MARRIOTT INN WILL BE AVAILABLE FROM 1900-2000

MAY 26, 1976 WEDNESDAY

0900 PARALLEL SESSIONS, MARRIOTT INN

I. TELECONFERENCING AND NETWORK MAIL — BELVEDERE ROOM

The use of computer networks for mail and teleconferencing is rapidly becoming practical for large-scale implementation. The ability to easily maintain specialized distribution lists and effect immediate transmission of messages is far beyond the usual communication facilities available today. Teleconferencing adds another dimension to group interaction by using networked computer in the role of moderator, secretary, and participant.

Thomas Marill, CCA: TDA Message System

Doug Dodds, BBN: Problems in the Technology of Network Message Processing

Rod Stotz, USC-ISI: Military Message Handling Experiments

Jacques Vallee, IFTF: Computer Conferencing and Data Base Utilization

II. DATA TRANSLATION AND EXCHANGE STANDARDS — TREASURE ROOM

The translation of data formats for exchange between different operating systems has long been a major problem. Techniques for exchange of large, structured numeric data bases are being investigated by an ERDA group, and other groups are investigating high level languages for this purpose.

Al Brooks, UCC-ORNL: The IWGDE Extension of the ANSI Z39.2 Standards

Joseph Nardi, BNL: A Machinge Interpretable Design for Physical and Logical Description of Sequentially Archived Data

Arie Shoshani, SDC: On the Importance of Common Standards for Data Base Conversion

Nan Shu, IBM: An Approach to Data Migration in Computer Networks

David Richards, LBL: Incorporation of Hierarchically Structured Data in the ERDA Exchange Standard

III. GATEWAYS AND LOCAL NETWORKS, PART I — ANGEL ROOM

Gateways are loosely defined as nodes common to two or more networks; the most common type provides intercommunication between local networks and distributed national networks. Topics to be discussed include both the implementation schemes for several local networks and facilities for interfacing these to other networks.

Michael Lyle, UCI: Transmission System Tradeoffs in Ring-Structured Digital Systems

Robert Metcalfe, TTI: ETHERNET: Distributed Packet Switching for Local Computer Networks

David Retz, USC-ISI: Packet Radio Network - Protocol Structures

1030 BREAK

MAY 26, 1976

1100 PARALLEL SESSIONS, MARRIOTT INN

I. GATEWAYS AND LOCAL NETWORKS, PART II — ANGEL ROOM

Several new local networks are being designed at ERDA laboratories. Methods for providing gateways between networks will be discussed.

James Donnelley, LLL: Incorporating Gateways into Existing Network Protocols

William Greiman, LBL: MICROBUS - A Multiple Microprocessor Communications System

William Lidinsky, ANL: The Argonne Intra-Laboratory Network

Carl Sunshine, RAND: Network Interconnection Strategies

II. NETWORK REQUIREMENTS FOR DATA BASE SUPPORT — TREASURE ROOM

The standard operations on data bases require new concepts when applied in a distributed newwork environment. Inter-host communication protocols are being developed to solve the problems engendered by heterogenous operating systems.

Peter Alsberg, UI-CAC: Synchronization and Resiliency in Network Data Access

Yogen Dalal, Stanford: Distributed File Systems

Paul Mockapetris, UCI: Future Plans for a New Ring

Steve Kimbleton, NBS: Data Base Support Requirements for Network Operating Systems

III. DATA ACCESS AND MANIPULATION LANGUAGES — BELVEDERE ROOM

The problems involved in providing effective human-machine interfaces will be discussed. Primary emphasis is given to relational and hierarchical data structure models which provide the user a capability for accessing data bases in a manner independent of physical storage structures.

Elizabeth Feinler, SRI-ARC: NIC/QUERY - A Novice User Interface Program

Irv Traiger, IBM: System R: A Relational Approach to Data Base Management

David Richards, LBL: BDMS - The Berkeley Data-Base Management System

Michael Stonebraker, UCB/LBL: The Data Base Management System INGRES

1230 LUNCHEON, MARRIOTT INN

MAY 26, 1976

1400 COMBINED SESSION, MARRIOTT INN

DATA MANAGEMENT MACHINES — BELVEDERE ROOM

The proliferation of complex data management systems for various problem classes has lead to the investigation of systems architectures for providing fast and fail-soft hardware support. Several experimental implementations of these ideas will be discussed.

Herbert Baskin, UCB/LBL: The Architecture of Data Base Oriented Systems

Jerry Farrell, CCA: The Datacomputer- A Network Data Utility

Stewart Schuster, Univ. Toronto: .The Case for a Parallel-Associative Approach to Data Base Machine Architectures

QUESTIONS REGARDING ACCOMMODATIONS, TRANSPORTATION, ETC. SHOULD BE DIRECTED TO:

Justine J. Lynch
Workshop Coordinator
(415) 647-1668

OR

Lee Handeland
(415) 843-2740 ext. 6203
(FTS) 451-6203

COMMENTS AND SUGGESTIONS REGARDING THE PROGRAMS SHOULD BE DIRECTED TO:

Dennis Hall DEHall@MULTICS
Program Chairman
(415) 843-2740 ext. 6053
(FTS) 451-6053

OR

Don Austin DAustin@MULTICS
Workshop Chairman
(415) 843-2740 ext. 5313
(FTS) 451-5313

ACRONYM DEFINITIONS

ANL	*Argonne National Laboratory*
ARPA	*Advanced Research Projects Agency*
BBN	*Bolt Beranek and Newman*
BNL	*Brookhaven National Laboratory*
CCA	*Computer Corporation of America*
CSC	*Computer Sciences Corporation*
ERDA/DPR	*Energy Research & Development Admin./Div. of Physical Research*
IBM	*International Business Machines*
IFTF	*Institute For The Future*
LBL	*Lawrence Berkeley Laboratory*
LLL	*Lawrence Livermore Laboratory*
ORNL	*Oak Ridge National Laboratory*
NYU	*New York University (Courant Institute)*
PARC	*Palo Alto Research Center*
SDC	*Systems Development Corporation*
SRI-ARC	*Stanford Research Institute - Augmentation Research Center*
TTI	*Transition Technology Incorporated*
UCB	*University of California at Berkeley*
UCLA	*University of California at Los Angeles*
UCI	*University of California at Irvine*
UI-CAC	*University of Illinois - Center for Advanced Computation*
USC-ISI	*University of Southern California - Information Sciences Institute*

Workshop Participants

MR. ROBERT P. ABBOTT
LAWRENCE LIVERMORE LABORATORY
MAIL STOP L-60
LIVERMORE, CALIFORNIA 94550

MR. TED ALBERT
ENVIRONMENTAL INFORMATION SYSTEMS
ENVIRONMENT AND SAFETY
ERDA
WASHINGTON D. C. 20545

MR. SCOTT ALBERT
10400 DEMOCRACY LANE
POTOMOC, MARYLAND 20854

DR. PETER ALSBERG
CENTER FOR ADVANCED COMPUTATION
121 ADVANCED COMPUTATION BUILDING
UNIVERSITY OF ILLINOIS
URBANA, ILLINOIS 61801

MR. LARRY AMIOT
ARGONNE NATIONAL LABORATORY
APPLIED MATHEMATICS DIVISION
9700 SOUTH CASS AVENUE
ARGONNE, ILLINOIS 60439

MR. TIMOTHY ANDERSON
COMPUTER CORPORATION OF AMERICA
545 TECHNOLOGY SQUARE
CAMBRIDGE, MA 02139

DR. DONALD AUSTIN
COMPUTER SCIENCE AND APPLIED MATH
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94720

MR. RONALD BARE
ARGONNE NATIONAL LABORATORY
APPLIED MATHEMATICS DIVISION
BUILDING 221
9700 SOUTH CASS AVENUE
ARGONNE, ILLINOIS 60439

MR. LARRY L. BARNES
COMPUTER LOCK SYSTEMS
2126 6TH STREET
BERKELEY, CALIFORNIA

PROFESSOR HERBERT BASKIN
COMPUTER SYSTEMS RESEARCH LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94704

MS. CARMEN BENKOVITZ
DEPARTMENT OF APPLIED SCIENCE
BROOKHAVEN NATIONAL LABORATORY
UPTON, NEW YORK 11973

MR. WILLIAM BENSON
COMPUTER SCIENCE AND APPLIED MATH
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIF. 94720

PROFESSOR ROBERT BIRGE
PHYSICS, COMPUTER SCIENCE AND MATHEMATICS
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94720

MR. GARRET L. BOER
LAWRENCE LIVERMORE LABORATORY
P. O. BOX 808, MAIL STOP L-60
LIVERMORE, CA 94550

PROFESSOR KEN BOWLES
UNIVERSITY OF CALIFORNIA AT SAN DIEGO
UCSD COMPUTER CENTER
BOX 109
LA JOLLA, CALIFORNIA 92037

DR. ROBERT BRADEN
CAMPUS COMPUTING NETWORK
U. C. L. A.
LOS ANGELES, CA 90024

DR. A. A. BROOKS
UNION CARBIDE CORP
COMPUTER SCIENCES DIVISION
P. O. BOX X
OAK RIDGE, TENNESSEE 37830

MR. NOEL BROWN
COMPUTER SCIENCE AND APPLIED MATH
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CA 94720

DR. RICHARD BUCHNESS
BRAIN RESEARCH
UNIVERSITY OF CALIFORNIA, L. A.
LOS ANGELES, CA 90024

MS. ELIZABETH CAMPBELL
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LABORATORY FOR NUCLEAR SCIENCE, RM 24-017
77 MASSACHUSETTS AVENUE
CAMBRIDGE, MASSACHUSETTS 02139

DR. GRAHAM CAMPBELL
APPLIED MATHEMATICS DIVISION
BROOKHAVEN NATIONAL LABORATORY
UPTON, LONG ISLAND
NEW YORK, NEW YORK 11973

DR. CHUN-FAI CHAN
COMPUTER SCIENCE AND APPLIED MATH
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CA 94720

DR. PAUL CHAN
COMPUTER SCIENCE AND APPLIED MATH
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94720

MS. TRICIA COFFEEN
COMPUTER SCIENCE AND APPLIED MATH
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94720

MR. VERN CROW
BATELLE NORTHWEST
MATH BLDG, BOX 999
RICHLAND, WASHINGTON 99352

DR. YOGEN DALAL
STANFORD UNIVERSITY
DIGITAL SYSTEMS LABORATORY
ERL 404
STANFORD, CALIFORNIA 94305

MR. PAUL DIONNE
SYSTEMS DEPARTMENT
PACIFIC NORTHWEST LABORATORY
P. O. BOX 999
RICHLAND, WASHINGTON 99352

MR. DOUG DODDS
BOLT, BERANEK, & NEWMAN, INC.
50 MOUTON ST.
CAMBRIDGE, MASS 02138

MR. JAMES E. DONNELLEY
LAWRENCE LIVERMORE LABORATORY
P.O. BOX 808, L-307
LIVERMORE, CALIFORNIA 94550

MR. VICTOR ELISCHER
LAWRENCE BERKELEY LABORATORY
BUILDING 90, ROOM 3132
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94720

PROFESSOR GERALD ESTRIN
COMPUTER SCIENCES DEPARTMENT
SCHOOL OF ENGINEERING AND
APPLIED SCIENCE
UNIVERSITY OF CALIFORNIA
LOS ANGELES, CALIFORNIA 94305

MR. FAMES A. FAIR
RM. 3118, BLDG 90
LAWRENCE BERKELEY LABORATORY
BERKELEY, CA 94720

MR. GERALD P. FARRELL
COMPUTER CORPORATON OF AMERICA
575 TECHNOLOGY SQUARE
CAMBRIDGE, MASSACHUSETTS 02139

DR. ELIZABETH FEINLER
STANFORD RESEARCH INSTITUTE
NETWORK INFORMATION CENTER
333 RAVENSWOOD AVENUE
MENLO PARK, CALIFORNIA 94025

MR. ROBERT FINK
COMPUTER CENTER
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94720

MR. JOHN G. FLETCHER
LAWRENCE LIVERMORE LABORATORY
COMPUTATION SYSTEMS DIV. L 60
LIVERMORE, CALIF. 94550

MR. ED FRANCESCHINI
COURANT INST. OF MATHEMATICAL SCI.
NEW YORK UNIVERSITY
NEW YORK, NEW YORK 10012

PROFESSOR GENE F. FRANKLIN
DURAND 131
STANFORD UNIVERSITY
STANFORD, CALIFORNIA 94305

MR. SHELDON FURST
COMPUTER SCIENCE AND APPLIED MATH
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIF. 94720

DR. STOCK GAINES
THE RAND CORPORATION
1700 MAIN STREET
SANTA MONICA, CA 90406

DR. ROBERT GARDNER
COMPUTER SCIENCE DEPARTMENT
SCHOOL OF ENGINEERING AND APPLIED SCIENCE
UNIVERSITY OF CALIFORNIA
LOS ANGELES, CALIFORNIA 90024

MS. WEN SUE GEE
COMPUTER SCIENCE AND APPLIED MATH
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIF. 94720

MR. FRED GEY
COMPUTER SCIENCE AND APPLIED MATH
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94720

MR. STEVE GLASEMAN
THE RAND CORPORATION
1700 MAIN STREET
SANTA MONICA, CA 90406

MR. GEOFFREY GOODFELLOW
STANFORD RESEARCH INSTITUTE
ARTIFICIAL INTELLIGENCE CENTER
333 RAVENSWOOD AVENUE
MENLO PARK, CALIFORNIA 94025

MS. PATRICIA GRACIAN
COMPUTER SCIENCE AND APPLIED MATH
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CA 94720

MR. JIM GRAY
K55-282
I B M
5600 COTTLE ROAD
SAN JOSE, CA 95193

DR. TERRY GRAY
COMPUTER SCIENCES DEPARTMENT
SCHOOL OF ENGINEERING AND APPLIED SCIENCE
UNIVERSITY OF CALIFORNIA
LOS ANGELES, CALIFORNIA 90024

MR. NORTON GREENFELD
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
545 TECHNOLOGY SQUARE
CAMBRIDGE, MASS. 02139

DR. WILLIAM GREIMAN
LAWRENCE BERKELEY LABORATORY
BUILDING 50A, ROOM 1144
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIIFORNIA 94720

DR. GARY R. GROSSMAN
UNIVERSITY OF ILLINOIS
CENTER FOR ADVANCED COMPUTATION
309 ADVANCED COMPUTATION BUILDING
URBANA, ILLINOIS 61801

MR. DENNIS HALL
COMPUTER SCIENCE AND APPLIED MATH
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94720

MR. VIKTOR E. HAMPEL
COMPUTATION DEPARTMENT
LAWRENCE LIVERMORE LABORATORY
P. O. BOX 808, L-380
LIVERMORE, CALIFORNIA 94550

MS. LEE HANDELAND
COMPUTER SCIENCE AND APPLIED MATH
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94720

MR. ROBERT HARVEY
COMPUTER CENTER
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94720

DR. LYMAN HAZELTON
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ROOM 24-017, LNS COMPUTER FACILITY
77 MASSACHUSETS AVENUE
CAMBRIDGE, MASSACHUSETS 02139

MR. ROBERT HEALEY
COMPUTER SCIENCE AND APPLIED MATH
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94720

MR. BRAD HECKMAN
COMPUTER SCIENCE AND APPLIED MATH
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94720

DR. JESSIE HERR
INFORMATION RESEARCH GROUP
LAWRENCE BERKELEY LABORATORY
BLDG. 50 ROOM 130
BERKELEY, CALIFORNIA 94720

DR. HARVARD HOLMES
COMPUTER SCIENCE AND APPLIED MATH
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94720

DR. ROBERT HONEA
ENERGY DIVISION
OAK RIDGE NATIONAL LABORATORY
P. O. BOX X
OAK RIDGE, TENNESSEE 37830

MR. HENRY HONECK
E. I. DUPONT
SAVANNAH RIVER LABORATORY
AIKEN, SC 29801

MR. VAN JACOBSON
LAWRENCE BERKELEY LABORATORY
BUILDING 90, ROOM 3130
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94720

MR. EDWARD JACQUES
COMMUNICATIONS AMD COMPUTER OPERATIONS
ERDA
WASHINGTON, D.C. 20545

DR. DANIEL KANE
COMPUTER SCIENCE AND APPLIED MATH
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94720

MS. MURIEL KANNEL
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LABORATORY FOR NUCLEAR SCIENCES
77 MASSACHUSETTS AVE. RM 24-021
CAMBRIDGE, MASSACHUSETTS 02139

DR. PEGGY M. KARP
TELENET COMMUNICATION CORPORATION
2710 SANDHILL ROAD
MENLO PARK, CALIFORNIA 94025

DR. JOHN KILLEEN
CTR
LAWRENCE LIVERMORE LABORATORY
P.O. BOX 808 - L-318
LIVERMORE, CALIFORNIA 94550

DR. STEVE KIMBLETON
CHIEF, COMPUTER NETWORKING DIVISION
B212 TECHNOLOGY BLDG.
NATIONAL BUREAU OF STANDARDS
WASHINGTON, DC 20234

DR. FRANK KING
IBM RESEARCH
K55-282
5600 COTTLE ROAD
SAN JOSE, CALIFORNIA 95193

MR. JEREMY KNIGHT
COMPUTER CENTER
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94720

MS. SUZANNE KRANZ
COMPUTER SCIENCE AND APPLIED MATH
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CA 94720

DR. JAN H. KREMERS
STANFORD RESEARCH INSTITUTE
333 RAVENSWOOD AVENUE
MENLO PARK, CA 94025

MR. PETER KREPS
COMPUTER SCIENCE & APPLIED MATH
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CA 94720

DR. IVY KUO
COMPUTER SCIENCE AND APPLIED MATH
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94720

MR. RALPH B. LANTZ
9700 SOUTH CASS AVENUE
ARGONNE NATIONAL LAB.
ARGONNE, ILLINOIS 60439

MR. RICHARD LA PIERRE
ENGINEERING
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIF. 94720

DR. DAVID LAYTON
REGIONAL STUDIES PROGRAM
LAWRENCE LIVERMORE LABORATORY
P. O. BOX 808, L-523
LIVERMORE, CALIFORNIA 94550

MS. BARBARA LEVINE
COMPUTER SCIENCE AND APPLIED MATH
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CA 94720

MR. HOWARD LEWIS
COMMUNICATIONS & COMPUTER OPERATIONS
ERDA
MAIL STOP CA-327
WASHINGTON, DC 20545

MR. WILLIAM LIDINSKY
APPLIED MATHEMATICS DIVISION
ARGONNE NATIONAL LABORATORY
9700 SOUTH CASS AVENUE
ARGONNE, ILLINOIS 60439

DR. ART LINDEMAN
APPLIED MATHEMATICS DIVISION
ARGONNE NATIONAL LABORATORY
9700 SOUTH CASS AVENUE
ARGONNE, ILLINOIS 60439

DR. ANDREW LOEBL
ENERGY DIVISION
OAK RIDGE NATIONAL LABORATORY
P. O. BOX X
OAK RIDGE, TENNESSEE 37830

MR. MIKE LYLE
INFORMATION SYSTEMS AND COMPUTER SCIENCE
UNIVERSITY OF CALIFORNIA
IRVINE, CALIFORNIA 92717

MS. JUSTINE J. LYNCH
WORKSHOP COORDINATOR
P.O. BOX 40279
SAN FRANCISCO, CALIFORNIA 94140

MR. PETER MACLEAN
COURANT INSTITUTE OF MATH. SCI.
NEW YORK UNIVERSITY
251 MERCER STREET
NEW YORK, NEW YORK 10012

MR. THOMAS MARILL
COMPUTER CORPORATION OF AMERICA
575 TECHNOLOGY SQUARE
CAMBRIDGE, MASS. 02139

MR. DAVID S. MAYNARD
STANFORD RESEARCH INSTITUTE
333 RAVENSWOOD AVENUE
MENLO PARK, CA 94025

PROFESSOR JOHN MCCARTHY
ARTIFICIAL INTELLIGENCE LABORATORY
STANFORD UNIVERSITY
STANFORD, CALIFORNIA 94305

MR. FRANK MC GIRT
GROUP C9 MS-255
LOS ALAMOS SCIENTIFIC LAB.
P. O. BOX 1663
LOS ALAMOS, NEW MEXICO 87545

DR. ROBERT METCALFE
TRANSACTION TECHNOLOGY INC.
10880 WILSHIRE BOULEVARD
LOS ANGELES, CALIFORNIA 90024

DR. DEANE MERRILL
COMPUTER SCIENCE AND APPLIED MATH
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94720

DR. JAMES C. MICHENER
INTER-METRICS ASSOCIATION
701 CONCORD AVENUE
CAMBRIDGE, MASSACHUSETTS

MR. PAUL MOCKAPETRIS
UNIVERSITY OF CALIFORNIA AT IRVINE
P. O. BOX 4942
IRVINE, CA 92716

MR. JIM MORRIS
XEROX CORPORATION
3333 COYOTE HILL ROAD
PALO ALTO, CA 94304

MR. JOSEPH NARDI
APPLIED MATHEMATICS DEPARTMENT
BROOKHAVEN NATIONAL LABORATORY
UPTON, NEW YORK 11973

MR. LOU NELSON
UNIVERSITY OF CALIFORNIA
3732 BOELTER HALL
LOS ANGELES, CALIFORNIA 90024

DR. ROBERT NYHOLM
LAWRENCE LIVERMORE LABORATORY
P.O. BOX 808 L-523
LIVERMORE, CALIFORNIA 94550

MR. O. K. OLMSTEAD
C 9 DIVISION
LOS ALAMOS SCIENTIFIC LABORATORY
P. O. BOX 1663
LOS ALAMOS, NEW MEXICO 87545

MS. LORRAINE J. OSTERER
APPLIED MATHEMATICS DEPARTMENT
BROOKHAVEN NATIONAL LABORATORY
UPTON, NY 11980

MR. LARRY PECK
COMPUTER SCIENCES DIVISION
OAK RIDGE NATIONAL LABORATORY
P. O. BOX X
OAK RIDGE, TENNESSEE 37830

DR. DAVID L. PEHRSON
LAWRENCE LIVERMORE LABORATORY
COMPUTATION DEPARTMENT
P. O. BOX 808, L-60
LIVERMORE, CA 94550

MR. ARNOLD PESKIN
APPLIED MATHEMATICS DEPARTMENT
BUILDING 215
BROOKHAVEN NATIONAL LABORATORY
UPTON, NEW YORK 11973

DR. JAMES C. T. POOL
APPLIED MATHEMATICS DIVISION
ARGONNE NATIONAL LABORATORY
9700 SOUTH CASS AVENUE
ARGONNE, ILLINOIS 60439

DR. JONATHAN POSTEL
STANFORD RESEARCH INSTITUTE
AUGMENTATION RESEARCH CENTER
333 RAVENSWOOD AVENUE
MENLO PARK, CALIFORNIA 94025

MR. G. R. PUTZOLU
IBM RESEARCH
K55-282
MONTEREY AND COTTLE ROADS
SAN JOSE, CALIFORNIA 95193

MR. CARL QUONG
COMPUTER SCIENCE AND APPLIED MATH
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94720

MR. JOHN RANELLETTI
COMPUTING SYSTEMS
LAWRENCE LIVERMORE LABORATORY
P. O. BOX 808
LIVERMORE, CALIFORNIA 94550

DR. DAVID RETZ
STANFORD RESEARCH INSTITUTE
AUGMENTATION RESEARCH CENTER
333 RAVENSWOOD AVENUE
MENLO PARK, CALIFORNIA 94025

DR. DAVID RICHARDS
COMPUTER SCIENCE AND APPLIED MATH
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94720

MR. DANIEL RIES
COMPUTATION DEPARTMENT
LAWRENCE LIVERMORE LABORATORY
P. O. BOX 808
LIVERMORE, CA 94550

MR. JOSEPH RINDE
TYMSHARE
20705 VALLEY GREEN DRIVE
CUPERTINO, CA 95014

DR. MILTON ROSE
MATHEMATICS AND COMPUTER SCIENCE RESEARCH
MOLECULAR, MATHMATICAL & GEOSCIENCES
MAIL STATION J-309
ERDA
WASHINGTON, D.C. 20545

DR. RICHARD ROYSTON
APPLIED MATHEMATICS DIVISION
ARGONNE NATIONAL LABORATORY
9700 SOUTH CASS AVENUE
ARGONNE, ILLINOIS 60439

MR. KAL SAFFRAN
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LABORATORY FOR NUCLEAR SCIENCE
BUILDING 24-019
77 MASSACHUSETTS AVENUE
CAMBRIDGE, MASSACHUSETTS 02139

MR. D. SAGALOWICZ
STANFORD RESEARCH INSTITUTE
333 RAVENSWOOD AVENUE
MENLO PARK, CA 94025

DR. STEWART SCHUSTER
COMPUTER SYSTEMS RESEARCH GROUP
UNIVERSITY OF TORONTO
TORONTO, ONTARIO
M5S184 CANADA

DR. ARIE SHOSHANI
SYSTEMS DEVELOPMENT CORPORATION
2500 COLORADO AVENUE
SANTA MONICA, CALIFORNIA 904006

MR. PHIL SPIRA
SYSTEMS CONTROL INC.
1801 PAGE MILL ROAD
PALO ALTO, CA 94304

PROFESSOR MICHAEL STONEBRAKER
COMPUTER SCIENCE DIVISION
573 EVANS HALL
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94720

DR. CARL SUNSHINE
RAND CORPORATION
1700 MAIN STREET
SANTA MONICA, CALIF. 90406

DR. JACQUES VALLEE
INSTITUTE FOR THE FUTURE
2740 SANDHILL ROAD
MENLO PARK, CALIFORNIA

DR. STEPHEN T. WALKER
ADVANCED RESEARCH PROJECTS AGENCY
1400 WILSON BOULEVARD
ARLINGTON, VIRGINIA 22209

MR. RICHARD WILEY
COMPUTER SCIENCE SERVICES DIVISION
C-3, MS265
LOS ALAMOS SCIENTIFIC LABORATORY
P. O. BOX 1663
LOS ALAMOS, NEW MEXICO 87545

DR. NAN SHU
IBM RESEARCH
K55-282
MONTEREY AND COTTLE ROADS
SAN JOSE, CALIFORNIA 95193

DR. ROBERT SPROULL
XEROX PALO ALTO RESEARCH CENTER
3333 COYOTE HILL ROAD
PALO ALTO, CALIFORNIA 94304

DR. ROB STOTZ
UNIVERSITY OF SOUTHERN CALIFORNIA
INFORMATION SCIENCES INSTITUTE
4676 ADMIRALTY WAY
MARINA DEL REY, CA 90291

DR. IRV TRAIGER
IBM RESEARCH
K55-282
MONTEREY AND COTTLE ROADS
SAN JOSE, CALIFORNIA 95193

MR. ALEX VRENIOS
APPLIED MATH DIVISION
ARGONNE NATIONAL LABORATORY
BLDG 221, C207
ARGONNE, ILLINOIS 60439

DR. JAMES WHITE
STANFORD RESEARCH INSTITUTE
AUGMENTATION RESEARCH CENTER
333 RAVENSWOOD AVENUE
MENLO PARK, CALIFORNIA 94025

MR. DAVE WILNER
LAWRENCE BERKELEY LABORATORY
BLDG 90, RM 3090
BERKELEY, CALIF. 94720

MS. SILVIA SORELL
COMPUTER SCIENCE AND APPLIED MATH
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94720

DR. PAUL STEVENS
PHYSICS DEPARTMENT
CAL TECH
PASADENA, CA 91126

MR. JOHN SUICH
SAVANNAH RIVER LABORATORY
AIKEN, SOUTH CAROLINA 29801

MR. LAROY TYMES
TYMSHARE
20705 VALLEY GREEN DRIVE
CUPERTINO, CALIFORNIA 95014

MS. PHYLLIS WALKER
ENERGY AND ENVIRONMENTAL SYSTEMS DIVISION
ARGONNE NATIONAL LABORATORY
9700 SOUTH CASS AVENUE
ARGONNE, ILLINOIS 60439

MR. KEN WILEY
LAWRENCE BERKELEY LABORATORY
BLDG 90, RM 3102
UNIVERSITY OF CALIFORNIA
BERKELEY, CA 94720

MR. PETER WOOD
COMPUTER SCIENCE AND APPLIED MATH
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIF. 94720