ARPA ORDER NO. 2223

ISI/SR-74-2

A Research Program in the field of Computer Technology

ANNUAL TECHNICAL REPORT: May 1973 - May 1974

prepared for the **Advanced Research Projects Agency**

INFORMATION SCIENCES INSTITUTE

UNIVERSITY OF SOUTHERN CALIFORNIA

4676 Admiralty Way/ Marina del Rey/ California 90291

(213) 822-1511

STEVE CASNER

ISI/SR-74-2

A Research Program in the field of Computer Technology

ANNUAL TECHNICAL REPORT: May 1973 - May 1974

prepared for the **Advanced Research Projects Agency**

EFFECTIVE DATE OF CONTRACT:	17 May 1972
CONTRACT EXPIRATION DATE:	15 July 1975
AMOUNT OF CONTRACT:	$6,616,798
PRINCIPAL INVESTIGATOR:	Keith W. Uncapher (213) 822-1511

THIS RESEARCH IS SUPPORTED BY THE ADVANCED RESEARCH PROJECTS AGENCY UNDER CONTRACT NO. DAHC15 72 C 0308, ARPA ORDER NO. 2223, PROGRAM CODE NO. 3D30 AND 3P10.

INFORMATION SCIENCES INSTITUTE

UNIVERSITY OF SOUTHERN CALIFORNIA

4676 Admiralty Way/ Marina del Rey/ California 90291

(213) 822-1511

RESEARCH/ADMINISTRATION SUPPORT

Gerhard W. Albert	Business Manager
Nancy L. Bryan	Editor
Deborah L. Dunn	Reception
Patricia A. Hagedorn	Secretary to Deputy Director
Rose L. Kattlove	Librarian
G. Nelson Lucas	Graphic Arts
Mary J. Winchell	Business Office
Ruth White	Secretary to Director; Principal Secretary

C O N T E N T S

F I G U R E S

ABSTRACT

This report summarizes the research performed by USC/Information Sciences Institute from 17 May 1973 to 16 May 1974. The research is aimed at applying computer science and technology to problem areas of high DoD/military impact.

The ISI program consists of ten research areas: Automatic Programming- the study of acquiring and using problem knowledge for program generation; Program Verification- logical proof of program validity; Programming Research Instrument- development of a major time-shared microprogramming facility; Protection Analysis- methods of assessing the viability of security mechanisms of operating systems; Command and Control Message Processing Technology- the study of advanced computer-based techniques for military message handling; Information Automation- development of a user-oriented message service for large-scale military requirements; Network Secure Speech- work on low-bandwidth, secure voice transmission using an asynchronous packet-switched network; Technology Support- development of Xerox Graphics Printer facilities, portable terminals, and

military office terminal system; Network Management Information Center- development of a network performance-measurement methodology; and Research Resources- operation of TENEX service and continuing development of advanced support equipment. Programmed Automation, an investigation of the feasibility of a computer-based manufacturing technology, consisted of a study phase only.

EXECUTIVE OVERVIEW

The Information Sciences Institute (ISI), a research unit of the University of Southern California's School of Engineering, was formed in May 1972 to do research in the fields of computer and communications sciences with an emphasis on systems and applications. The Institute, located off-campus, has sufficient autonomy within the University structure to assure it the freedom required to identify and engage in significant research programs.

A close relationship is maintained with USC academic programs through active cooperation among the Institute, the School of Engineering, the Department of Electrical Engineering, and the Computer Science Department. Ph.D. thesis supervision is an integral part of ISI programs. Also, participating faculty and graduate students from other departments provide interdisciplinary capabilities for ISI projects.

At the end of the second year of operation, the full-time professional research staff numbers 32. The total number of ISI employees, including full-time research staff, participating faculty and graduate students, and support personnel, is 80.

The activities of ISI's eight major areas of research and associated support projects are summarized briefly below. Some of the research projects reported in this document are discrete activities in themselves; others can be seen as parts of a larger whole. For example, Automatic Programming, Program Verification, and the Programming Research Instrument projects should be considered as individual parts of an overall research effort in Programming Methodology; Command and Control Message Processing Technology, Information Automation, Network Secure Speech, and Technology Support are linked elements of a major investigation into Network Communications Technology. These mutual interdependencies among the various projects at ISI contribute largely to the fruitfulness of the Institute's research activities.

AUTOMATIC PROGRAMMING

The Automatic Programming project is composed of three main activities: first, the design and preliminary implementation of an automatic programming system; second, the implementation of a Programmer's Interface, a language-independent interface between a programmer and his language; and third, the initiation (in cooperation with ARPA) of the National Software Works, an effort to transfer technology for program development to military commands.

The automatic programming system represents a major attempt to deal directly with nonprofessional computer users without the intervention of computer programmers. The primary goal of the system is to acquire from a dialogue with the user the elements of the problem domain, then translate the ill-defined specifications into a precise form and write a program to accomplish the user's desired task. An initial automatic programming system and the associative data base mechanisms it relies on have been implemented, and it has been used in turn to implement an initial version of the Model Completion phase. This version is able to take carefully selected small, simple domain and problem statements of an imprecise form and convert them to running programs by performing a series of structuring and program building transformations that reorder the arguments of relations and actions, convert the arguments to the correct type, and fill in omissions.

The Programmer's Interface effort addresses the general problem of creating a suitable on-line environment for programming. It attempts to exploit the observable uniformity among programming environment systems by creating a single programming environment capable of easily interfacing users to a variety of on-line programming languages. With minimal cost and effort, the Programmer's Interface thus transforms a programmer's language into a programming system with powerful debugging, editing, and filing capabilities. The first such Programmer's Interface has been constructed and interfaced to the programming language ECL. It was implemented and debugged in approximately three weeks, including the interface to ECL. Although no other language interfaces have yet been built, it is estimated that an interface to another suitable language could be designed, implemented, and debugged in less than a week.

The concept of the National Software Works arose from the realization that serious progress in improving the production of large programs would be made only as the result of a major attempt to improve general access to tools, centralizing the management of a vast inventory of currently existing hardware and software. To do this, it is proposed to link the great variety of tools now available on the ARPANET into a coherent system for software development, employing a standard interface for users and a very large secondary memory for storing and managing user files. ISI was instrumental in the system design and functions as technical project engineer.

PROGRAM VERIFICATION

The verification of a computer program is the demonstration by a mathematical proof of the consistency between the program and its specification or documentation. Program verification can greatly increase the reliability of software by assuring that programs actually do what they are intended to do. The large amount of time and effort now devoted to testing activities can be significantly decreased by verification procedures, which will ultimately reduce software costs. Program verification can also have an influence on the design of programming languages and can serve to test advances in both programming methodology and the semantic definition of programming languages.

In addition to experimenting with new specification languages, structuring methods, and proof methods, the Program Verification project is building software tools that will aid in the design and construction of verified programs. Verification condition generators for two programming languages are already in operation; algebraic simplification and interactive proving of the lemmas produced by the verification condition generator are also being accomplished. Current plans are to develop these components into a smoothly integrated system whose capabilities can be demonstrated on significant, real programs.

PROGRAMMING RESEARCH INSTRUMENT

The Programming Research Instrument (PRIM) project has created a fully protected experimental computing environment with continuous multiuser access. An ARPANET-based system, PRIM allows each researcher to create his own specialized computing engine capable of being changed and adapted to his specific needs. The PRIM hardware and software together

provide a working environment in which the user can implement his own computer in microcode and run that computer in his target program environment. PRIM can be used to explore computer architecture, language development, and special-purpose processor design--all of especial relevance to DoD selection and use of computer equipment.

The PRIM facility makes it possible to simulate new hardware architectures and designs in microprogrammed software. That is, software can be created for hardware not yet available, and hardware designs may be extensively used and changed even before the prototype stage of development is reached, which should both cut lead time and improve decisions connected with the special-purpose machine procurement cycle.

At present, most microprogrammed processors operate in a single user environment, with a minimal operating system and a single application. The particular value of the PRIM project and its introduction of a microprogrammed supervisor (microvisor) state has been to make an easily accessed, sharable, powerful design tool for the computer development community.

To familarize potential users with the operation of the PRIM system, ISI will provide introductory seminars and an extensive documentation package. PRIM user documentation, consisting of an Overview, User's Guide, MLP-900 Reference Manual, and GPM Reference Manual, is nearing completion and should be available to interested potential users by mid-1974.

PROTECTION ANALYSIS

The Protection Analysis project has developed and is continuing to refine techniques and standards for testing and evaluating the protection features of operating systems. Its goal is to provide answers to the question of what tests can and should be applied to operating systems in order to determine to what extent a given system meets its requirements for preventing unauthorized or improper operations and how systems can best be designed and implemented to reflect such requirements. The research directly supports the software security requirements issued by DoD security policymaking agencies.

During the last reporting period, what is now the Protection Analysis project was designated as the Empirical System Study and Protection Theory

projects. These projects merged to study ways of applying empirical techniques toward achieving a "production" evaluation tool. The study was influenced by the observations that the security community lacks a production evaluation tool; that there is currently no known organized effort to collect and analyze protection errors; and that such errors do fall into distinct classes or "patterns." The study focuses on the way in which the output of protection evaluations (i.e., errors and their patterns) can be utilized as feedback in the development and improvement of an evaluation method itself. The result is the design of a systematized pattern-driven evaluation scheme that utilizes the output of error analysis both directly, to govern the evaluation process, and indirectly, to increase the comprehensiveness of the tool based on this method. The Empirical System Study group also devised a simple, economical, and reliable approach to the security retrofit problem for batch and remote job entry systems, termed encapsulation.

COMMAND AND CONTROL MESSAGE PROCESSING TECHNOLOGY

This project explores the use of advanced computer and communication techniques in military environments. The implementation of an automatic message handling service on a packet-switched digital network (exemplified by the ARPANET) has immediate and significant usefulness to the military community. The possible applications of such a service have served as the principal focus of the work to date.

A specific example of such an environment was the object of a study performed by ISI in the spring of 1973 to investigate the possible application of network technology to the COTCO (Consolidation of Telecommunications on Oahu) program, a DoD effort to improve military communications on Oahu.

The Navy, which has been assigned the task of implementing the COTCO program, currently has the ISI plan under consideration. Interest has been expressed in performing, with the cooperation of ARPA, a test of an interactive system such as the one proposed.

In the meantime, the Command and Control Message Processing Technology (CCMPT) project has addressed the issues involved in the implementation of automated message handling services for other military environments besides COTCO. In order to achieve the sort of system

envisioned, it is necessary to provide the following basic components and attributes:

- Core-system hardware architecture that serves as the foundation for building the service.
- Core-system software architecture and programs whose facilities are easy to learn and operate by users unfamiliar with computers.
- Application software that performs the functions required.
- Reliability of service.
- Security of data.

The CCMPT project is currently in a study phase, identifying the problems and the opportunities for message processing in the military environment. The program outlining detailed areas for research is now being prepared. In addition, we are exploring opportunities for joint development with military users that will apply our message technology in experimental form to actual military situations.

INFORMATION AUTOMATION

The Information Automation project is currently in the design phase of a telecommunications task conceived to aid the military and users of the ARPANET. The goal is to implement an on-line message-handling system (the core of the Command and Control Message Processing Technology Project above), which will be usable by people unfamiliar with computers. The initial study for this task investigated the user style and functional requirements of a large multiservice military community. The functional specifications for this system include message preparation (creation, editing, coordination), message transmission (routing, release, status query), and message delivery (priority, delivery, sorting, scanning, forwarding). Additionally, the system will support off-line report generation and message archives. The research goal of the Information Automation project is to develop the necessary human-factors methodology needed to introduce interactive computer terminals into office environments. The project thesis is that current systems technology will suffice for many of the current automatable problems and that what is required is a collection of techniques to make the computer acceptable and useful to non-technical personnel. The target system will be implemented at ISI on TENEX and the ARPANET. It is expected that an experimental system providing the above

functions will be available to ARPANET users early in 1975.

NETWORK SECURE SPEECH

In response to the high military priority of developing a digital means for secure speech transmission, the ISI Network Secure Speech project is attempting to establish the means to use a packet-switching network (the ARPANET) for secure, high-quality, real-time, full-duplex voice communication. The resulting voice communications system will have the following advantages: it will be able to be secured (encrypted) to any desired level of complexity and security; it will be able to achieve extremely high quality transmission with a low error rate; it will exploit the natural pauses and breaks in human speech to achieve a lower overall data rate; it will be highly compatible with future communications systems (satellites, lasers, etc.). The project's major goal is to establish the methodology of using the ARPANET for communication and to implement it with real-time bandwidth compression (vocoding) as a proof of feasibility. Present efforts are to perform network measurements for the development of the required protocols and to implement the chosen vocoding technique; future plans are to optimize the communications protocols and to improve the quality and reduce the bandwidth of the vocoding technique.

TECHNOLOGY SUPPORT

The Technology Support section describes three of the major advanced hardware systems being developed at ISI: the enhancement of a Xerox Graphics Printer system to produce a high-quality document printing capability in the form of a network terminal; a video display system; and a briefcase-sized portable terminal designed for use with the ARPANET. Each of these hardware efforts was undertaken either to demonstrate a capability for a recognized DoD application or to provide the necessary support for the several software projects that compose the Network Communications Technology effort (Command and Control Message Processing Technology, Information Automation, and Network Secure Speech).

NETWORK MANAGEMENT INFORMATION CENTER

Computer networks service a highly heterogeneous user population, possess a complex structure of hardware and software, and require careful arbitration of the potential conflicts among users,

hosts, and the network budget. The need for a performance measurement methodology is great, particularly because of the dynamic short- and long-term variations in network workload.

The Network Management Information Center (NMIC) at ISI, launched in March 1974, will provide an effective means for resolving technological issues in network performance by developing a report/display capability that will furnish information on network status and operations as well as providing an alternatives assessment capability that will permit rapid management assessment of the relative desirability of network alterations and enhancements.

Achieving the capabilities described above requires accurate knowledge of network/subnetwork status and performance. Thus, the initial focus of the project will be to establish an alternate Network Control Center (NCC) which will (1) provide a scheduled backup capability to the existing NCC at BBN, (2) develop operating policies and procedures for concurrent cooperative operation of multiple NCC's, thus assuring the integrity of the network in the event of an NCC outage, and (3) develop requirements and objectives for a minimum-manpower, minimum-skill-level NCC, thus clearly separating the roles of equipment vendor and operations.

NMIC will be of direct use to all DoD and domestic agencies concerned with the design and implementation of computer networks. Since it will provide a centralized means for data reduction and transformation into forms permitting direct interpretation by management, this research will also constitute a performance-analysis capability of direct interest to agencies dealing with sensitive material--agencies traditionally unable to gain full use of performance tools and consultants because of the sensitive nature of their job stream. In addition, the research will also be of direct benefit to ARPA/IPTO in managing the ARPANET, since on-line performance information maintained at NMIC will provide a centralized access point for all network performance information for use in management decisionmaking.

PROGRAMMED AUTOMATION

From July 1973 to December 1973 the Programmed Automation project evaluated the feasibility of various advancements in computer-based manufacturing systems, evaluated the economic impact on the U.S.

economy and the DoD from implementing those advances, and defined the specific development program and resources required to achieve them. This project consisted of a study phase only, and further work has since been discontinued.

RESEARCH RESOURCES

The ISI time-sharing facility, operated as a research and service center in support of a broad set of ARPA projects, consists of two PDP-10 CPUs, paging box, large-capacity memory, high-performance paging drum, on-line file storage, and associated peripherals. The hardware acquired in the past year as part of the general effort to improve system availability, capacity, and efficiency includes a second PDP-10 CPU, an additional 128K of high-speed memory, six disk drives that will approximately double the present on-line file storage, and several interfaces and switches designed and built at ISI.

In an effort to reduce system load, a group allocation scheme was introduced in March. This scheme divides the total number of available job slots into general-access and limited-access categories, with each type further subdivided into groups assigned a quota (varying in size throughout the day) specifying how many group members will be guaranteed access to the system. Group members whose group quota is filled can log in off-quota to fill other slots not in use at the time. When the system's total available high-quota is set to a sufficiently low level, the system is responsive and effective to those allowed to log on. In all, the initial experience with group allocation has been good.

The report that follows presents a detailed view of the goals and accomplishments of the many different areas of ISI research during the Institute's second year of operation.

AUTOMATIC PROGRAMMING

PROJECT LEADER: Robert M. Balzer

RESEARCH STAFF: Norton R. Greenfeld
William C. Mann
Walter R. Ryder
David S. Wile
Albert L. Zobrist

CONSULTANTS: John Brown
Martin J. Kay
William S. Mark
Ann D. Rubin
B. A. Sheil

RESEARCH STAFF SUPPORT: Chloe Holg

RESEARCH ASSISTANTS: Richard Hale
Robert Lingard
Nadine Malcolm
David Wilczynski

INTRODUCTION

As reported last year, work in the Automatic Programming (AP) project has centered upon three main activities. First, we have designed and begun to implement an automatic programming system. Functionally, the two most important characteristics of this system are its independence from any particular problem domain and its attempt to deal directly with nonprofessional computer users without the intervention of computer programmers. These choices have largely dictated the direction of the project. Domain independence requires that the "physics" of the domain--its objects and their relationships with other objects, its laws, its transformations, and its constraints--be available in a processable form within the system and that the system be general enough to deal effectively with a wide variety of such physics. Direct interaction with nonprofessional computer users means that both the physics and the problem statements will be in problem-oriented (as opposed to computer-oriented) terms, preferably in natural language, and that they will be "loose" descriptions containing incomplete, inconsistent, and irrelevant statements rather than a precise formal structure. The primary goal of the system is to acquire from a dialogue with the user the physics of the loosely defined domain, structure it, and use it to understand further communication

and write a program to accomplish the user's stated task.

Toward this end, we have implemented an associative data base in which to store the rules, objects, and relations of a domain and have integrated into LISP a pattern match facility to access the items stored. Using this facility we have also implemented a preliminary version of our system's structuring portion, called Model Completion, which is able to structure and convert to executable form simple small domain and problem descriptions. The other major component of our system, Domain Acquisition, which will acquire the loose model of the domain, is still in the planning stage; we are at present simulating its behavior in scenarios to learn more about how it should perform. During this next year, we expect to implement an initial version of Domain Acquisition, achieve a complete running system, and concentrate on extending both the capabilities of each phase and the system's ability to deal with domains of significant size and complexity.

Second, we have completed work on a language-independent interface between a programmer and his language, called the Programmer's Interface (PI), which transforms his language into a programming system with almost all of the capabilities of the INTERLISP (formerly BBN-LISP) system: powerful debugging, editing, and filing capabilities, as well as the ability to modify, group, and reissue previous commands. This work is significant at two levels. First, it provides a very quick and inexpensive way of enhancing the programming environment of an interactive language. Much more importantly, however, it demonstrates the relative language independence of many of the tools required for program building and debugging.

This recognition led directly to the initiation, in cooperation with ARPA, of our third area of involvement, the National Software Works (NSW). This is a major new ARPA project to transfer technology for program development to military commands within DoD. Because the initial NSW is planned for test installation in July 1975, its scope is deliberately limited to make this early delivery feasible. Nevertheless, it incorporates many advanced concepts and is explicitly designed for expansion. It significantly extends the PI concept by separating the tools used for programming from those used for program execution.

The NSW provides a framework that controls the access to and use of tools through a standard user interface and enables new tools to be easily added. Because these tools can exist on different machines, the framework acts as an operating system by making the use of a tool independent of its location; it automatically initiates the tool, establishes communication with it, transfers necessary files for its use, and saves any output generated. The NSW thus ties together tools that exist anywhere on the ARPANET into a coherent system for software development. Part of the NSW is a user interface that standardizes the way in which users initiate a tool, communicate with it, ask for help, or examine their output. The final component is a very large secondary memory for the storage and management of user files.

In addition to these components of the NSW, which merely provide a convenient home for tools and which enable users to uniformly access and communicate with them, the tools themselves exist, determining the system's capabilities to affect software development. These tools are divided into two categories: (1) "centralized" tools, like editors, flowcharters, test case generators, etc., which are independent of the target machine on which the program being developed is to run while in production mode and which therefore only require implementation on a single machine, and (2) execution machine tools, like compilers and run-time monitors, which actually run on the target machine and which must be re-implemented on each target machine.

AUTOMATIC PROGRAMMING EFFORT

After an initial survey of current work in the field, the AP project members developed a plan for attacking what appeared to be the fundamental issues of automatic programming. This took the form of an actual system, the design and implementation of which is now in its early stages. The results of the initial survey and the overall view of the field adopted are reported elsewhere [1,2]. It should be emphasized that this project is seen not as an incremental advance in computer languages or the art of programming, but rather as an attempt to make the power of the computer available to a large class of users without the necessity of a step like that now called programming. Ultimately, a client should be able to negotiate directly with a

computer system in much the same way as he now negotiates with a programmer.

Computer use generally falls into two categories: use of existing programs or creation of new ones. There is no sharp distinction between the two, because data fed into existing programs can be thought of as instructions that program their behavior and because the creation of new programs utilizes either compilers or interpreters that treat such instructions as data. Also, the techniques for translating a task into appropriate input for the two are very similar. Nevertheless, we have chosen to deal only with programming activities, which we regard as the process of translating a task to be performed into a computer language, taking into account the constraints and limitations of both the computer and the domain of interest from which the task was drawn.

The constraints and restrictions of the computer have increasingly been incorporated and internalized in programming advances for several years. They are manifest in better languages, automatic storage mechanisms, and optimizations of many forms. On the other hand the structure, constraints, and limitations of the problem domain have generally not been incorporated into programming systems. The use of such knowledge is a major theme of automatic programming, characterizing the distinction between it and conventional programming, and raises a number of issues. If the system is to understand something of a domain--a particular universe of discourse--how is the knowledge on which this understanding is based to be represented? What procedures can be made available for exploiting this knowledge in guiding the system's interaction with a user and in generating programs? How, in particular, is the essentially nonprocedural information in constraints and limitations to be reflected in a procedural form? What can be done to help identify inconsistencies? How can the system be given a capacity for inference similar to that which forms the mainstay of human communication and which allows obvious details to be left unspecified? Will the system be able to understand its own products well enough to be able to modify them in response to changed requirements? Answers to these questions define the front on which important advances in automatic programming will be made.

The designers of programming systems have until now concentrated their attention on creating instruments that would be easy to play. Like all instruments, the system had a purely passive role in the programming enterprise. We, on the other hand, took the view that the problem of programming is largely a problem of communication and that communication, to be easy and natural, must be with an active agent.

Thus the main distinction between conventional and automatic programming is the latter's use of a semantic model of a domain to structure the dialogue between the system and the user, to understand the user's responses, and to translate the user's responses into actions. The major distinctions between the work reported here and other automatic programming efforts are the following: first, its independence of any particular domain and its acquisition of the domain model through a dialogue with the user; and second, the informal and typically ill-structured manner in which both the domain semantics and the task to be programmed are specified. In fact, these two areas represent the two main focuses of the project: dialogue-driven acquisition of a domain and translation of ill-defined specifications into a precise form.

Overall System Structure

In our plan, the AP system consists of four processing modules and six data bases. The data bases consist, as much as possible, of descriptive (rather than imperative) knowledge, organized so that the system can use this knowledge in many different ways. These data bases have been segregated because of the different logical functions they perform and because of the way they are treated by the different processing modules.

Data Bases

The Domain Knowledge data base contains all the descriptive information about the problem domain, such as the types of objects that can exist in the domain and their descriptions, the types of actions that can occur in the domain, the relations that may exist between objects or events (action occurrences), and any constraints that must be satisfied by the domain.

The Domain Model contains, at any point in time, an instantaneous snapshot of the instantiated objects in the domain

and their relationship to other objects in the domain. It represents, through time, a direct simulation of the problem domain.

The Loose Model contains the problem statement in an imprecise form that may be incomplete or ambiguous and that can be understood only in the context of the information in the Domain Knowledge and Domain Model data bases.

The Precise Model, on the other hand, represents a precise, complete, unambiguous, and directly interpretable process for solving the problem posed.

The Strategy Knowledge data base consists of information that guides the choice of actions and/or objects for those actions when alternative possibilities exist within the domain.

Finally, the Script data base contains partially-filled-in forms that guide the dialogue between the system and the user and are dynamically altered on the basis of the user's input and by the demands of the Model Completion module.

Processing Modules

Initially, to simplify the implementation, the processing modules will be highly self-contained and will have only a limited knowledge of the processing and requirements of other modules. Later they will be more highly integrated and cooperative.

The Domain Acquisition module is responsible for communicating with the user, building the Domain Knowledge and Domain Model data bases, obtaining the Loose Model statement, determining on syntactic grounds the well-formedness of all this information, building and modifying the Script, and using it to direct the dialogue for the acquisition of further information necessary for such syntactic well-formedness or requested by the Model Completion module.

The Model Completion module determines the semantic well-formedness of the Loose Model on the basis of the information in the Domain Knowledge and Domain Model data bases. It is responsible for transforming the Loose Model into an operational, interpretable form called the Precise Model. Any inability to perform this transformation causes a description of the reason to be passed back through the Script to the Domain Acquisition phase, which then interacts with the user to correct the deficiency (usually by adding more

knowledge about the domain to the Domain Knowledge data base).

The Interpreter executes the action sequences in the Precise Model and updates the Domain Model accordingly. It is responsible for locating objects defined descriptively, evaluating conditions to select alternative sequences of actions, and maintaining restrictions on domain behavior.

The Data Base Handler is responsible for maintaining the various data bases, deciding on store-recompute policy, maintaining consistency, and (through inference) obscuring the difference between explicit and implicit data.

A primary objective of our project has been to create a core experimental system for testing progress on Domain Acquisition and Model Completion. As such, the Interpreter and Data Base Handler, which have been completely specified and implemented, are being used for both the Precise Model and the implementation of the AP system itself. To fully utilize these implementation capabilities, the Domain Acquisition and Model Completion modules will be treated as domains with their own actions, objects, constraints, and rules of inference. This bootstrapping will focus attention on the real problems of using our approach in complex domains.

A more detailed description of the system is given in the following subsections, which focus on the representation of knowledge and the form of the Precise Model produced by Model Completion.

Knowledge Representation

Throughout the system, knowledge is represented as stored tuples. The first element of any tuple specifies the type of tuple; the rest of the elements are the arguments for that tuple. Each stored tuple is associated with a particular domain. Data bases are compartmentalized into separate domains that form a lattice. Each domain is defined as A-KIND-OF (AKO) another domain; this structure forms the basis of the domain lattice. The interpretation of the lattice structure is that, unless specifically prohibited, properties (of all types) from higher level domains are inherited by lower level ones.

The structure of knowledge in the system is highly constrained by two mechanisms: types and constraints. Each

element of a tuple must be of a type acceptable for that argument as specified in the definition of that kind of tuple. Like domains, types are defined by AKO relation and form lattices. (This structure is very similar to MAPL [3].) An element of a tuple is acceptable if its type is the same as that specified in the tuple definition or if its type is a lattice descendant of the specified type. In addition to type acceptability, the elements of a tuple must also satisfy arbitrary constraints specified in the tuple definition. These constraints are checked at the time that the tuple is added to a domain.

A domain consists of types (objects), actions, relations, constraints, rules of inference, and instantiations of all of the above. Together with the type and constraint mechanisms for tuples, this knowledge of the kinds of information contained within a domain represents the syntactic basis used by the Domain Acquisition module to construct and modify its Script, and hence its dialogue with the user.

Precise Model

The Precise Model is the restatement of the user's problem in the programming language AP/1 [4]. This language is an extension of LISP [5], which supports associative relational data bases with domain compartmentalization, strongly typed variables, compound pattern matches, and failure control. Strong typing and compound patterns are especially important in simplifying the system's writing of the Precise Model by minimizing the translation between it and the Loose Model and by reducing and simplifying the control structures required. In fact, compound patterns have enabled the elimination of backtracking and its replacement by a single FOR loop that iterates through a set of instantiations of the compound pattern. It also makes it possible to apply intelligence, within the pattern matcher, to determine how best to obtain valid instantiations.

Additionally, Model Completion utilizes only a subset of AP/1 (which is also the implementation language for the project) to further simplify the writing and analysis of Precise Model programs. The major difference is that the Precise Model utilizes no free or local variables, except for pattern-match variables, that are instantiated during the matching process. All communication between

routines is either by way of explicit parameter passing or through data contained in the Domain Model.

AP/1 generally allows the arbitrary mixing of tuples to be instantiated and functions to be evaluated. This includes the functions AND, OR, and NOT, as well as any other defined LISP functions. It is assumed that such functions have no side effects. Each tuple in an expression is treated as a function and evaluated if it has a function definition. If not, then it is treated as a pattern to be instantiated. Because there are no free variables and the only local variables are pattern-match variables, the rule for instantiation is very simple. Any parameter or variable unbound at the time it is encountered within a pattern is instantiated; already bound variables are left unchanged.

The value of a pattern is always the instantiated version of that pattern if the match was successful, or NIL otherwise. Because no other possibilities exist, all pattern matches return either the instantiated pattern or NIL. The concept of failure does not exist within the pattern matcher, since it always returns to its caller with one or the other of these values.

The routines (statements) that invoke the pattern matcher may take other actions upon the returned value. They may extract from it particular bindings or subexpressions or cause failure when a NIL value is returned. Each of the "statements" in AP/1 is, in fact, a function that uses the value returned from the pattern matcher as it sees fit. In this regard, the AND, OR, and NOT functions are no different than any other in the system.

Current Project Status

AP/1 and the associative data base mechanisms it relies on have been implemented, and it has been used in turn to implement an initial version of the Model Completion phase. This version is able to take (carefully selected) small, simple domain and problem statements in Loose Model form and convert them to running programs by performing a series of structuring and program building transformations that reorder the arguments of relations and actions, convert the arguments to the correct type, and fill in missing ones. They also fill in omitted relations between objects, convert

actions, implications, and constraints to procedural form, and infer implied relations needed to make sense of structures in the Loose Model.

In all cases, these transformations are not complete. They recognize only specific simple instances of the desired situation. During the next year, we plan to extend the range of these transformations, include new ones, and implement an initial version of Domain Acquisition to form a complete system.

One problem in particular will be attacked. Currently the system attempts to make each part of the generated program "precise" (that is, well-defined), and stops transforming each part when this goal is satisfied. However, in the generated program, only certain "precise" forms will occur in the data base, and all those used must work in harmony with each other. Thus only certain "precise" forms should be acceptable; this set must be determined on a global basis.

THE PROGRAMMER'S INTERFACE

The Programmer's Interface (PI) addresses the general problem of creating a suitable on-line environment for programming. The amount of software to support such an on-line environment, and the effort required to produce it, is very large relative to that needed to produce a programming language, a fact that largely accounts for the scarcity of such programming environments. This factor was largely responsible for the discarding of a major language (QA4 [6]) as a separate entity and its inclusion instead as a set of extensions in a LISP [5] environment. The few systems that do exist (e.g., LISP, APL, BASIC, and PL/I) have greatly benefited their users and have strongly contributed to the widespread acceptance of the associated language.

At a bare minimum, a suitable programming environment consists of an on-line interpreter (or incremental compiler), an integrated interactive source-level debugging and editing system, and a supporting file structure. More extensive environments would include such facilities as automatic spelling correction, structural editors, tracing packages, test case generators, documentation facilities, and so forth.

Examining several programming environment systems, one recognizes a great deal of uniformity. Most of the software supporting these systems is

similar in both its organizational structure and functions. In fact, the systems differ in detail more because of differences in style among system designers than because of differences required by the programming languages.

The PI concept attempts to exploit this uniformity by creating a single programming environment capable of easily interfacing users with a wide variety of on-line programming languages. The PI is thus responsible for transforming these programming languages into systems. The cost of providing such an environment for a language would drop from the several man-years now required to the few man-days (estimated) to interface to a PI. Additionally, the existence of a common programming environment for many different languages would justify including further capabilities.

This common programming environment provided by a PI should include facilities for creating, modifying, storing, and retrieving programs; on-line debugging, including trace and break facilities as well as the facilities of the language for evaluation of expressions at breaks; modifying the interface between routines (via an ADVISE [7] capability); automatic spelling correction; remembering, modifying, and reissuing previous inputs; and undoing the effects of any of these PI facilities.

Such a PI has been constructed and interfaced to the programming language ECL [8]. The remainder of the PI discussion explains the PI concept in terms of this implemented program. The deficiencies of this particular implementation are discussed in the Evaluation subsection.

System Architecture

The facilities provided by the implemented Programmer's Interface (PI-1) are based on the INTERLISP (formerly BBN-LISP) system. In fact, they are the facilities of this system, as modified for language independence. PI-1 itself, implemented in INTERLISP, coexists with the facilities it invokes to provide the programming environment. INTERLISP was chosen as the basis both because it already had an extensive set of programming tools in an accessible form and because the structure and operation of the tools could easily be altered to operate as required for a PI.

The system structure is shown in Figure 1.1. The ARPANET [9] is used as the communications mechanism between PI-1 and the user's language processor, a choice which has three advantages. First, it allows PI-1 to be interfaced to any language processor available on the ARPANET, independent of what machine it runs on. Second, this interfacing can be done by PI-1 without the knowledge of the language processor; thus no modifications to the language processor are required. Finally, the use of the ARPANET greatly simplifies implementing the interconnection by allowing external character strings to be used for communication rather than internal data structures with the attendant incompatibility problems.

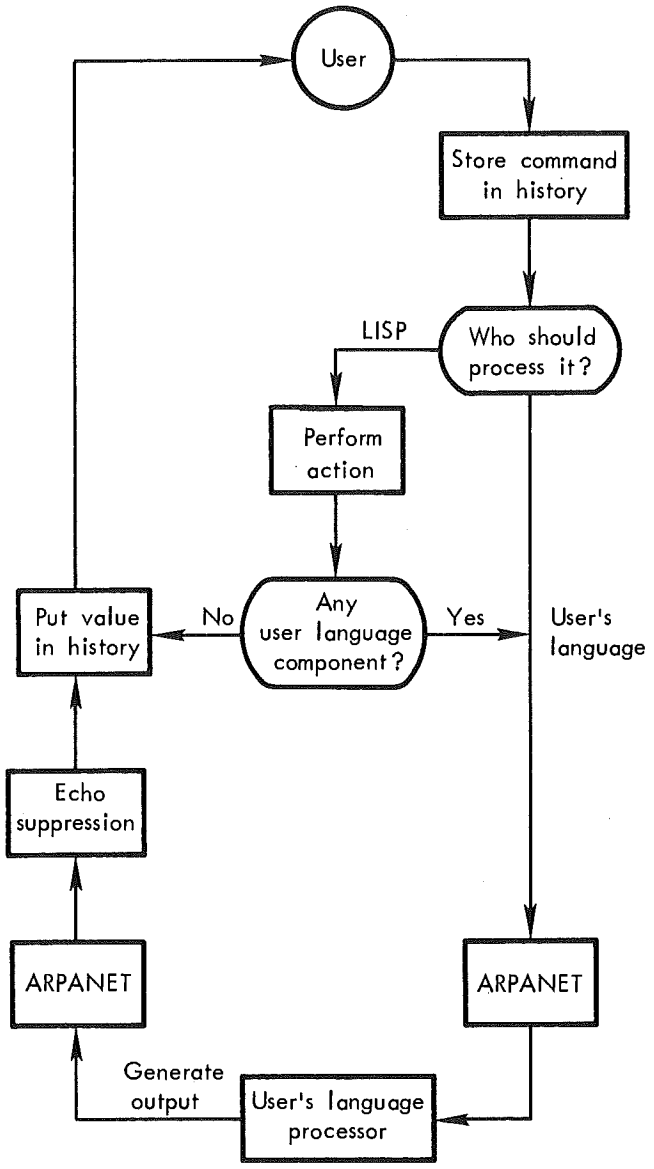

Figure 1.1 System architecture

Three properties are required of a language processor to be used with a PI:

1) A way must exist to form a coroutine linkage between the language processor and the PI by interconnecting their I/O ports. This type of linkage is discussed in detail in Ref. 10. With PI-1, the ARPANET provides this linkage. Thus, for PI-1, any language processor available on the ARPANET satisfies the first requirement.

2) It must have an on-line evaluator (either an interpreter or fast compiler) and be able to field breaks or errors within a computation.

3) It must be able to evaluate arbitrary forms in that language either at

breakpoints or at the top level.

PI-1 begins processing user input by storing it in a history list used by the Programmer's Assistant [7] (an INTERLISP subsystem) to retrieve, edit, group, reissue, or undo previous commands. PI-1 then examines the input to determine whether it should be processed by an INTERLISP facility or by the user's language processor. Basically, environment-type activities, such as loading files, editing programs, advising a function, etc., are performed within PI-1, while expressions in the user's language to be evaluated are passed to the language processor.

If the user's input is intended for his language processor, it is passed across the ARPANET to that language processor. Any output generated by the processor is received across the ARPANET, again by PI-1. It suppresses the echo of the input and passes the output to the user, extracting from it the "value" and putting it into the history list for use by the Programmer's Assistant.

If the user's input is an environment-type command and should be performed within PI-1, the appropriate facility is invoked. In simple cases the operation completes, returns a value that is put in the history, and another input is processed. In more complex situations, some interaction is required during the operation with the user's language processor; this is accomplished by dynamically generating a series of inputs for the language processor that will have the desired effect or return the desired information. These are passed through the communications mechanisms to the processor; its output is captured, and either the success of the modifications is verified or the desired information is extracted. Any number of such cycles may be required before the PI-1 facility completes its processing of the user's command. As an example, consider the loading of a file. As the function definitions are read in, they are stored as a property of the corresponding atoms to be used by the PI-1's editor for any modifications required later. The function definitions are also passed to the language processor so that it can use these for evaluation. Thus, one cycle is required for each function defined in the file.

PI-1 maintains a copy of all functions defined by the user, and this is used by PI-1's editor when the user alters

the definition. Whenever this definition changes (by redefinition or through exiting the editor), the resulting definition is passed to the language processor as a new definition of the function.

Interfacing a Language to a Programmer's Interface

Most of PI-1 is language-independent, but the following portions must be modified to accept a new language: syntax modification, synchronization, program writing, and debugging.

The INTERLISP editor used by PI-1 is structural rather than string-oriented. To be effective, the text it is manipulating must have a structural basis. The syntax modification routines are responsible for introducing the structure into the user's language (only for use within PI-1). This structure is of two forms. First is the grouping of characters into lexical units. The user's language may have very different lexical grouping rules than LISP, and the syntax modification package is responsible for the lexical analysis. Second, the lexical units thus produced are grouped into larger units by the use of parentheses. These units can be nested within one another to form the familiar LISP S-expression structure. The designer of the syntax modifier must decide where to introduce this structual grouping. In ALGOL-like languages, it would be natural to group the lexical units of a statement together and groups of statements within blocks together. The structural groupings selected are introduced into all program text input by the user and employed by him to direct the editor in its modifications of this text. When this text is passed to the language processor, those structural groupings artificially introduced for editing purposes are removed before transmission.

PI-1 and the language processor must be kept in synchronization with each other. Logically this is very simple; it is accomplished by having PI-1 wait until the language processor has completed evaluating the previous input before giving it another. This situation is signaled by the language processor's attempt to read the next input. Unfortunately (due to a deficiency in the network protocol), this information is not available. Therefore the language processor's state of readiness must be determined by examination of its output

stream. Fortunately, most on-line language processors explicitly indicate their readiness for more input by providing the user with a Prompt character. The language processor's output must be scanned for this character, and this action is used as a synchronization mechanism between PI-1 and the language processor.

Several facilities within PI-1, such as Break, Trace, and Advise, cause additional statements to be written into the user's program for evaluation at runtime. The interfacer of a new language must specify the form of these additions.

PI-1 contains many advanced debugging capabilities not found in most language processors. All of these aids are based on information gathered during execution or at a breakpoint within the program. To use these facilities, the designer of the language interface must supply routines that provide the basic information on which these debugging aids are built.

PI-1 was implemented and debugged in approximately three weeks, including the language interface to ECL. Although no other language interfaces have yet been built, it is estimated that an interface to another suitable language could be designed, implemented, and debugged in less than a week.

Evaluation

The significance of the PI effort lies neither in the particular interface provided between INTERLISP and ECL nor in the extensive capabilities provided the user, but rather in 1) the observation that very little of the interface itself, or of the capabilities provided, is language-dependent; 2) the recognition that the programming environment can be effectively split into an "environment" part and an execution and evaluation part; and 3) the experience gained from building such a system and interfacing a language to it.

PI-1, however, suffers from a number of deficiencies, the most important of which is the use of already existing tools in environments more general than those for which they were designed. This was most notable in the use of LISP's editor for nonstructured text (making it necessary to introduce structure by parentheses) and the requirement to replace LISP's input routines to provide the proper lexical analysis for the interfaced language. Both of these

problems could be avoided in a PI if it used the syntax description of the language to guide the input and the editing and display of programs.

While one of the strengths of the PI concept is the split between "environment" and evaluation, this split introduces the problem of communication and synchronization, for each part must keep the other informed about changes it makes that affect the other. In PI-1, this communication and synchronization was partial and clumsy. The flow of information from the environment to the evaluation part was adequate, but the reverse flow was not. The need to communicate to another program suitable explanations of the state of the evaluation, the cause of the error, or even that an error occurred, was simply not envisioned or planned for.

PI-1 has thus demonstrated that a moderately integrated PI can be built whose facilities are far beyond what is typically available at a fraction of the cost. However, development of a highly integrated PI will have to await a better understanding of the functional requirements of a language processor in such an environment.

Although the PI has been interfaced to only one language (ECL), and although it contains only a small fraction of the capabilities ultimately desired, it is having a major effect by acting as a prototype for the NSW described below [11,12], which is being undertaken to develop this understanding and provide a single, common, comprehensive programming environment interfaced to a wide variety of languages running on many different machines communicating through a network. New languages or machines could be interfaced to the system at a fraction of the cost of providing a separate programming environment. Widespread usage would justify the expenditure of more resources to augment and improve the capabilities provided. Such a PI could free users from having to develop their programs using only software available on their own machines and could provide a much more comprehensive and coordinated software development package than is currently available.

THE NATIONAL SOFTWARE WORKS

The production and maintenance of large programs is still an outrageously expensive activity; the costs are not only high, but also difficult to predict or

control. Aside from the manifest benefits derived from the use of compilers and (some) operating systems and a certain amount of improvement experienced by programmers who code interactively, it is not at all clear that the last twenty years of research and development in programming technology have made any serious inroads upon the problem. This situation is particularly interesting in the light of a general suspicion that, in principle, the problem ought to be eased by the creation of better software to support the program production and maintenance process, for surely a great deal has been spent in the effort to invent just such software. The reasons for our failure are arguable; a variety of hypotheses have been put forward:

- That the necessary tools--or, at least, many of them--exist in the research centers but are not being effectively delivered to the practical programming community.
- That feedback from the user community has insufficient influence on the research laboratories, so that research emphasis is unrelated to user needs.
- That the necessary tools exist, but are diffused over a variety of hardware in many physical locations, the problem being that of difficulty of access.

Each of these hypotheses--and the list may readily be extended--doubtless contains a certain amount of truth, and collectively they surely suggest that dramatic improvements in the way programs are built are less likely to come from marginal improvements in present tools (or the invention of some magical new one) than from better methods of tool access and delivery and better communication between research laboratory and end user.

The idea of a National Software Works (NSW) on the ARPANET [9] arose fairly naturally from these considerations. If some number of end users were put on the network, and enough additional off-the-shelf software were brought up on the network to supply a complete set of conventional tools--compilers, documentation aids, debugging systems, etc.--for normal program development work, some useful results might be expected to follow:

- The user would immediately have more convenient access to standard tools

unavailable on his own hardware (or seldom available if his hardware is often tied up running production).

- The user would find it easy to access novel tools in use at research facilities presently on the network, but not otherwise available to him.
- Contact between the research laboratories and the user community would naturally improve.

In sum, the NSW might both immediately improve the present situation of the user and, in the long term, provide an effective vehicle for the communication of need from user to researcher and of responsive tool from researcher to user.

It was soon recognized, however, that a view of the NSW as a mere lash-up of tools that happened to reside on the ARPANET would be extremely short-sighted. The fact that all programmer contact with tools would pass through a common communication mechanism with immense computing resources created a golden opportunity for the study--and perhaps control--of the whole process of creating and maintaining large programs. This thought was particularly attractive in the light of our feeling that one of the most weakly supported areas in the production and maintenance process is project management--a tool that keeps track of what is going on, relating particular programmer activities to each other and to the overall project.

The NSW Environment

Against the background of our feeling that serious progress in making the production of large programs a more rational process will come less from the polishing of particular tools than from a frontal attack on the issue of improved access to tools and centralized management of the vast inventory of text floating around a large project, the logic of our NSW strategy becomes easy to see.

- It is our intent to put a project's programmers on-line to the ARPANET, which has the immediate effect of giving them access to many tools unavailable on their own local hardware.
- We will supply interactive editing packages, both a general text editor and editors that "speak" one or two common programming languages. The effect of such tools in facilitating program preparation and modification

is too well known to require any defense here.

- Projects will be able to store these files on very inexpensive on-line mass storage devices (such as the Datacomputer [14]). This should relieve a considerable part of a project's local off-line file maintenance problems and facilitate load-sharing when the project's local computer is busy.

- A File Manager will always be on-line monitoring the content and structure of the project's files and keeping the books up to date, as text pieces are created and manipulated.

The presence of the first three facilities will permit the project to conduct its business more or less as it does now (using the same languages, the same tools, etc.) with certain improvements in ease of tool access and foreign hardware access, editing, and file management. In addition, the project may at its option experiment with using different tools scattered around the network.

The fourth facility opens the door to some genuinely new ways of controlling projects in the future. To begin with, a fairly powerful query system will be provided to answer questions about any filed entity: what it is, where it came from, what other entities depend on it, etc. Later we will introduce a variety of experimental tools for project control that use the File Manager's books as their primary data or that use the fact of the File Manager's existence as their means of invocation (after all, the latter provides a single control point "awakened" every time anything interesting happens).

Supporting Technology

Virtually all multiprogram operating systems have attempted to create a suitable programming environment by providing a set of tools. Some merely provided a library from which tools could be selected one at a time by the programmer. Others, like MULTICS, CP-67, VS, and TENEX, have provided an on-line environment for program building and debugging.

All of these systems have been built on a single computer, which has severely limited their capability to provide the type of environment described in the previous subsection. In fact, until

recently a combination of several such hardware and software technical problems existed that prevented the conception and implementation of this type of environment. These problems and their solution in the NSW are discussed below.

Single-machine implementation of all tools. Computer networks, such as the ARPANET, have established a communication mechanism whereby cooperating programs in different machines can function together as a single system. The technical basis for eliminating these problems is provided by computer networks, centralized mass storage, the PI [14] described in the previous section, ACTORS [15], and execution machines (see the System Description subsection below). The PI has utilized this network technology to create an on-line programming environment combining tools that run on different machines.

Nonintegrated "tool-at-a-time" systems. Current systems either segregate their tools into noninteracting components invoked one at a time or provide highly complex integrated versions of these, with the interactions between them built into the systems themselves. The type of programming environment we envision requires that actions or events in one part of the system permeate the rest to maintain consistency and coordination between the components. The ACTORS concept, by externalizing and removing the control and communication between the component parts, greatly simplifies the construction of an integrated and coordinated system.

Machine independence. Although tools running on different machines may be integrated into a single tool, the technology does not exist to run a single program on several different machines and obtain the same results. Therefore, software being produced must be executed and tested on the machine for which it is intended to run in production mode. Thus, if the software environment is to be used to produce programs for more than one machine, each of these must be connected through the computer network and a small portion of the system replicated on each execution machine to provide for translation and run-time monitoring capabilities. The rest of the software environment is common and can be shared independent of the machine for which execution is intended.

Language Independence. Currently, if software is to be produced for more than one language, then the tools must either be duplicated in separate and distinct integrated programming environments or else available in a nonintegrated tool-at-a-time mode. The PI has shown that many of these tools are language-independent or only slightly language-dependent and has demonstrated how such tools can be extended to handle a wide set of programming languages. It utilizes the programming tools (editors, file systems, debuggers, Programmer's Assistant [7], etc.) developed for one language (LISP [5]) for the development of software in other languages (e.g., ECL [8]). It has established interface requirements for other languages that would greatly reduce the effort required to transform these from simple interactive programming languages into an extensive programming environment.

Economics. In addition to the costs of creating an appropriate programming environment addressed above, several economic factors currently limit the use and utility of existing programming environments. Most machines are sized for their production, not development, requirements. Hence, typically they contain neither enough mass storage for the files that would be required in an on-line environment nor enough memory to support both the code being developed and the tools for that development. Also, access to the system is limited by the priorities of the production workload. Networking and economies of scale offer solutions by providing access to a system specifically designed and sized for software development and on which no production workload exists. Charges would be based on usage and development costs for the system, spread over a much wider community of users because of the language- and machine-independence aspects of the system. In addition, very cost-effective mass storage can be provided by the Datacomputer, which provides a trillion-bit on-line memory at a cost of about a dollar per megabit per year.

System Description

The hardware for the NSW, shown in Figure 1.2, consists of three logical components interconnected by the ARPANET: mass storage, the execution machine, and the interactive machine. The first consists of the Datacomputer, composed of a trillion-bit store and a file management system. The second, the execution machine, is a set of machines responsible for

running the program being developed and for collecting data on its execution. For each program being developed, the execution machine chosen is automatically the same as the production machine for that program. Thus, during development, a program is executed on the same (actually a copy of the) machine it will run on during production. This mechanism eliminates all machine-dependence compatibility issues at the cost of replicating the execution software in each machine for which this capability is desired and having that machine available in the NSW. On the other hand, it provides the great advantage of allowing the final component, the interactive machine (or machines), to be independent of the choice of production machine, thereby allowing it to handle a wider set of implementation efforts. The interactive machine contains most of the system's software and provides all of the facilities of the NSW except those described above.

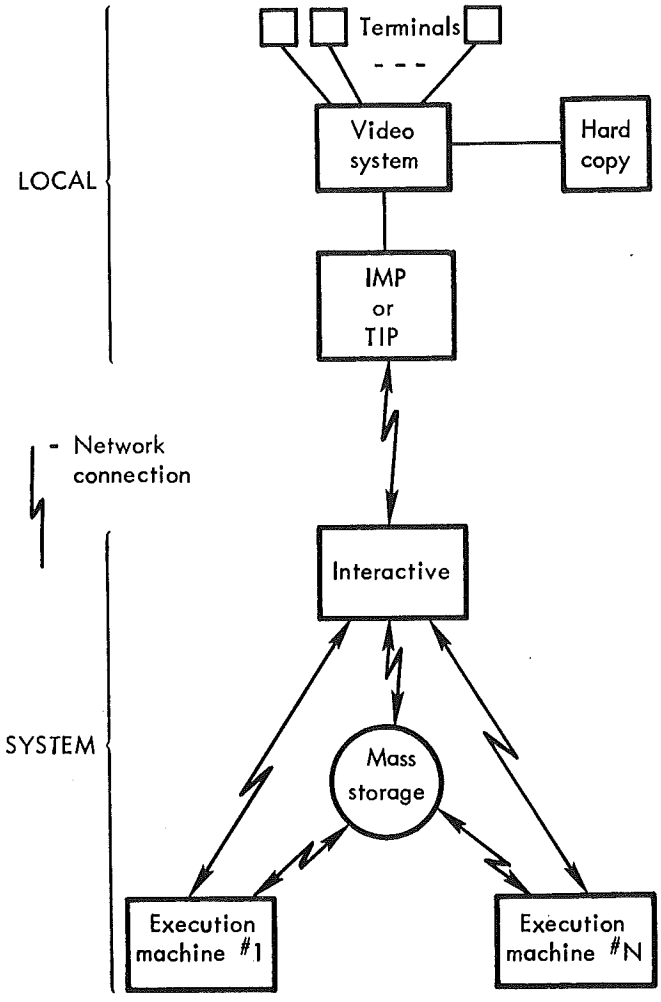

Figure 1.2 NSW hardware

The ARPANET not only interconnects the NSW components, but also provides access for users to the system and supports a variety of terminals. However, the NSW will be oriented toward the use of high-capacity video terminals.

Although the system is distributed across the ARPANET, it is organized so that neither the user nor the component software modules are aware of this. The user sees only a single integrated facility. The framework enables modules either locally or remotely connected to communicate without knowing each other's precise location.

System Growth

Though we have described the NSW system and its tool interface at length, we have not discussed the tools themselves, partly because we believe it is the delivery and access system rather than specific tools which is significant, partly because many of the tools to be included already exist, and mainly because NSW does not plan on becomming involved in a massive tool development effort. Rather, NSW has been designed to create a competitive marketplace in which vendors make tools available on a usage charge basis. Such a marketplace has advantages for all concerned. Because the NSW will eventually have a very large user base, vendors will have a wider audience from which to recover their development costs. More importantly, since the entire user population can access and use the tools independent of their own production hardware (unless the tool is dependent on the execution machine), a single implementation is sufficient. The implementer is free to choose the most cost-effective machine for development and execution of his tool. Users are able to choose the best tool available and are not restricted to software running on their own hardware. Finally, because NSW does all the accounting for all users, the decision of which tools to install in the NSW and which ones to use can be distributed. Vendors who believe in the economic viability of their tools can install them and make the user aware of their availability. Users can, on an individual basis, decide which tool is most appropriate for their own needs and can try new tools without any administrative arrangements.

ISI, together with other members of the ARPA community, was instrumental in the conception and design of the National Software Works. During the remainder of the project, we will provide technical coordination and planning for its development and growth.

REFERENCES

1 Balzer, R. M., Atuomatic Programming, USC/Information Sciences Institute, RR-73-1 (in progress).

2 Balzer, R. M., "A Global View of Automatic Programming", Proceedings of Third International Joint Conference on Artificial Intelligence, Stanford University, August 20-23, 1973, pp. 494-499.

3 Project MAC Progress Report X, July 1972-July 1973, The Massachusetts Institute of Technology, Cambridge, Mass., 1973, pp. 174-176.

4 Balzer, R. M., AP/1: A Language for Automatic Programming, USC/Information Sciences Institute, RR-73-13 (in progress).

5 Teitelman, W., D. G. Bobrow, A. K. Hartley, and D. L. Murphy, BBN-LISP TENEX Reference Manual, July 1971.

6 Rulifson, J. F., J. A. Derksen, and R. J. Waldinger, QA4: A Procedural Calculus for Intuitive Reasoning, Stanford Research Institute Artificial Intelligence Center, Technical Note 73, November 1972.

7 Teitelman, W., "Automated Programming: The Programmer's Assistant", AFIPS Conference Proceedings, Fall Joint Computer Conference 1972, Vol. 41, AFIPS Press, Montvale, N.J., 1972, pp. 917-921.

8 Wegbreit, B., "The ECL Programming Systems," AFIPS Conference Proceedings, Fall Joint Computer Conference 1971, Vol. 39, AFIPS Press, Montvale, N.J., 1971, pp. 253-262.

9 Roberts, L. G., and B. D. Wessler, "Computer Network Development to Achieve Resource Sharing," AFIPS Conference Proceedings, Spring Joint Computer Conference 1970, Vol. 36, AFIPS Press, Montvale, N.J., 1970, pp. 543-549.

10 Balzer, R. M., Ports: A Method for Dynamic Interprogram Communication and Job Control, The Rand Corporation, R-605-ARPA, August 1971.

11 Balzer, R. M., T. E. Cheatham, S. Crocker, and S. Warshall, The National Software Works, USC/Information Sciences Institute, RR-73-18 (in progress).

12 Balzer, R. M., T. E. Cheatham, S. Crocker, and S. Warshall, Design of a National Software Works, USC/Information Sciences Institute, RR-73-16 (in progress).

13 Datacomputer Software Architecture, Datacomputer Project Working Paper 5, February 1972, Computer Corporation of America.

14 Balzer, R. M., A Language-Independent Programmer's Interface, USC/Information Sciences Institute, RR-73-15, March 1974.

15 Hewitt, C., P. Bishop, and R. Steiger, "A Universal Modular ACTOR Formalism for Artificial Intelligence," Proceedings of the Third International Joint Conference on Artificial Intelligence, Stanford University, August 20-23, 1973, pp. 235-245.

PROGRAM VERIFICATION

PROJECT LEADER: Ralph L. London

RESEARCH STAFF: Raymond L. Bates
Peter L. Bruell
Lawrence M. Fagan
Donald I. Good

RESEARCH STAFF SUPPORT: Betty L. Randall

RESEARCH ASSISTANT: Donald S. Lynn

"The concepts of program verification are actually the cornerstone of any deeper understanding of algorithms, without which the programmer would have no other tool besides his own insufficient intuition."

Niklaus Wirth, Systematic Programming: An Introduction, 1973, p. xii.

=== * ===

INTRODUCTION

To verify a computer program means to demonstrate, by a mathematical proof, the consistency between the program and the specifications or documentation of what the program is to do. Program verification can have a great impact on the construction of reliable software by assuring that programs actually do what is intended and by ultimately reducing software costs. In addition, program verification helps to influence the design of programming languages. It also serves as an important test-bed for advances in both programming methodology and the semantic definition of programming languages.

Program verification is not inexpensive or rapidly completed, but it need not be an additional cost in software projects. In Boehm's data on current software practices [1], it is especially relevant to note the large amounts of time and effort (45 to 50 percent) devoted to testing activities, much of which can be eliminated by verification procedures.

It is now technically possible to verify small programs automatically. Techniques and experience exist to permit

the computer-assisted verification of more ambitious programs. If necessary, it is even possible to verify still more ambitious programs manually.

A PROGRAM CONSTRUCTION AND VERIFICATION SYSTEM

The Program Verification project is building an integrated system of software tools to aid in the design and construction of verified programs and to experiment with new specification languages, structuring methods, and proof methods. We fully expect to demonstrate the system's feasibility and practical applicability on significant, real programs.

Previous experience with actual proofs of a variety of programs and with various experimental program verification systems supports two conclusions: (1) automatic or interactive program verifiers can be of significant help in proving actual programs, and (2) the best way of exploiting existing program proving methods is to develop a program and its proof simultaneously, that is, by considering the proof at each step of the program design and implementation process rather than writing the program first and then attempting to prove it. In short, if we are to prove programs of significant size and complexity with current methods, those programs must be designed to be proved.

Programs can be designed to be proved because current verification techniques are perfectly compatible with currently advocated methods of program construction (for example, the concepts of structured programming, levels of abstraction, and their numerous aliases). One can verify each elaboration of a developing program at the time of elaboration rather than wait to verify only the final version of the program. The use of this approach should result in a structured, modular, and understandable verification. An encouraging example is the verification [2] of a structured algorithm of several levels. The proof follows that structure exactly. Small changes to the algorithm will mean corresponding small changes to the proof.

Currently, it is not intended that the program construction and verification system include any significant automatic programming or code synthesis facilities. The system is not expected to aid in selecting an algorithm for accomplishing a

task or in choosing data and control structures. The human programmer still retains primary responsibility for designing, and for proving, his program. The purpose of the system is to provide a set of facilities that make it possible for the programmer to interactively develop, from a precise specification of his problem, both a program and a proof that the program solves his problem. It is assumed that the programmer is highly knowledgeable both in his problem area and in the methods of program proof supported by the system.

The system focuses on providing the programmer with three fundamental and interrelated capabilities: program construction, program execution, and program verification. The program construction part of the system provides a basis for designing the program on a structured, segment-by-segment basis while maintaining the integrity of the proof as the program is expanded and modified.

The program execution capability is supplied by an interpreter that provides a sophisticated debugging tool along conventional lines. By this means, faulty programs can be detected without using the considerably more expensive verification machinery. The verification part of this system is based on the conventional inductive assertion method: generate verification conditions, simplify them, and prove them.

The spirit of the verification component, as well as the entire system, is to provide automatic tools where practical and to rely on interactive capability for manual intervention otherwise. This philosophy is motivated by our beliefs that for real programs, (1) large parts of the total proof can and should be done automatically, and (2) in the foreseeable future, some parts will have to be done manually.

CURRENT STATUS

Our currently running system consists of a text editor, a program and specification parser, a program interpreter capable of running both actual and symbolic data, a verification condition generator, a simplification and substitution package, and a theorem prover.

The program verification portion of the system is based on decomposing the verification task as shown in Figure 2.1. The inputs are (1) the program to be

verified and (2) the specifications and assertions to which the program is to be shown consistent. The verification condition generator or lemma generator, invoking the syntactic and semantic rules of the programming language, reduces the verification task to proving a derived set of mathematical lemmas; the program itself has essentially been eliminated. Large segments of the derived lemmas are proved by relatively straightforward simplifications and substitutions involving global algebraic facts, global Boolean facts, and problem-specific facts. The remaining lemmas are then passed to an interactive theorem prover that invokes global, generally useful theorems as well as problem-specific theorems.

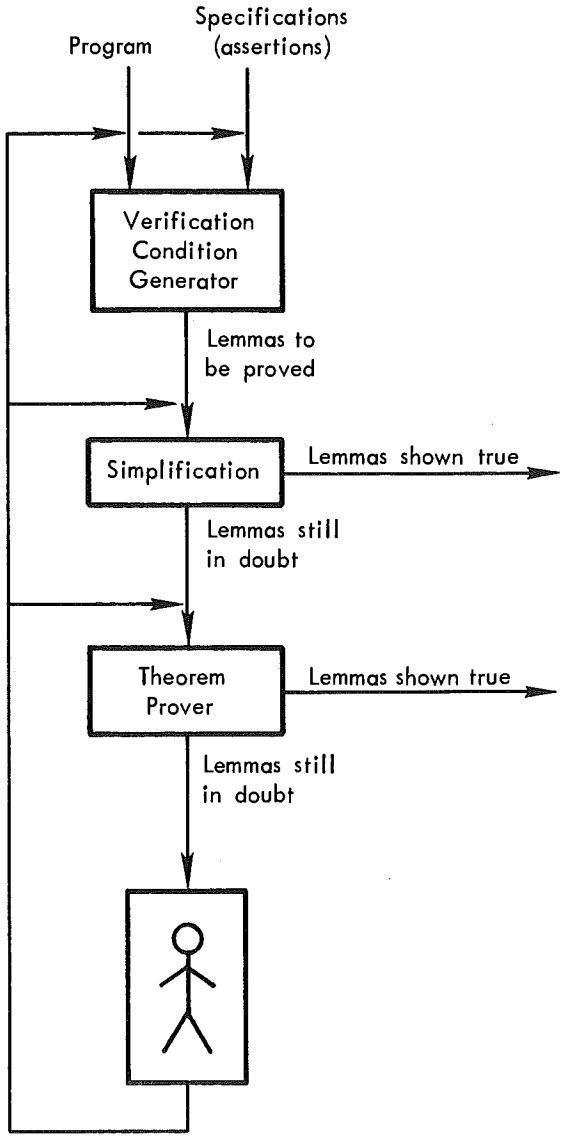

Figure 2.1 Decomposition of the verification task

The human user, represented by the stick figure, can guide the verification interactively. He is an important component of our verification system--in fact, one that distinguishes our system from most others. Such interaction is being used successfully in at least one theorem prover [3]. If, as we expect (and are beginning to demonstrate), the human is left to supply to the system only a minimal amount of crucial advice and hunches to complete the proofs of the remaining lemmas, then this will be an appropriate and cost-effective use of the human user in the verification task. Of course, ensuring that the system does not in the end ask the user to do what he considers too obvious is a nontrivial problem. Making the human an important

component seems a proper response to the genuinely open-ended nature of the facts, theorems, and deductions needed to verify real programs. If we wish the system to be able to handle a particular type of reasoning or set of facts, we can hopefully add the appropriate capability. In the meantime, however, the system/human combination can together verify programs.

Note in Figure 2.1 that the human can advise both the simplifier and the theorem prover, in particular, by supplying additional facts and theorems to be used. He can also change either the original program or the specifications, or both, if that seems appropriate in light of any false lemma (i.e., a counter-example has been found) or a lemma whose truth or falsity cannot be determined (i.e., the can't tell case).

A more detailed view of the verification part of the system is shown in Figure 2.2. Input programs are written in an interesting subset of the frequently-used Algol-like language Pascal, to which assertions have been added. The input specifications and assertions are written as (Boolean) expressions of Pascal, augmented by implication, equivalence, and limited quantification. Because function calls are permitted in expressions, this assertion language is theoretically adequate. In practice it is somewhat inelegant, but not inconvenient. Other notations for assertions are beginning to be developed and implemented.

The input is converted to a prefix representation used throughout the rest of the system. One use is as input to the interpreter for Pascal, permitting

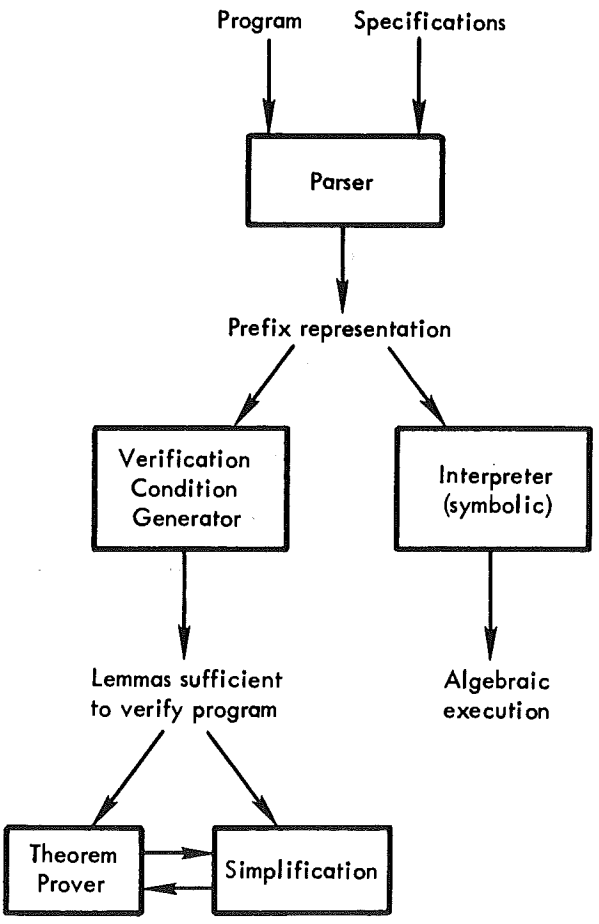

Figure 2.2 Overall organization of currently running verification system

algebraic execution, in addition to the usual numeric execution. One use for the algebraic execution is to assist in the (human) construction of the assertions needed in verification, although, of course, some program construction strategies suggest the assertions will be known before the actual program code is written. The interpreter can also serve as an alternative basis for a verification condition generator.

The other use for the prefix representation is as input to the verification condition generator, the output of which serves as input to the simplifier and to the theorem prover (whose function and use were described above). The verification condition generator is based directly upon the axioms and rules of inference that constitute the definition of Pascal. We also have available a standard Pascal compiler, an interesting convenience but not a necessity to the verification system.

A different view of the construction and verification system, in Figure 2.3, shows additional relationships of the components and emphasizes the outputs of each.

The construction and verification system is programmed in Reduce [4] and is therefore Lisp-based. Besides the virtues of programming in a high-level compilable language, Reduce provides three important advantages. First, nearly all of the

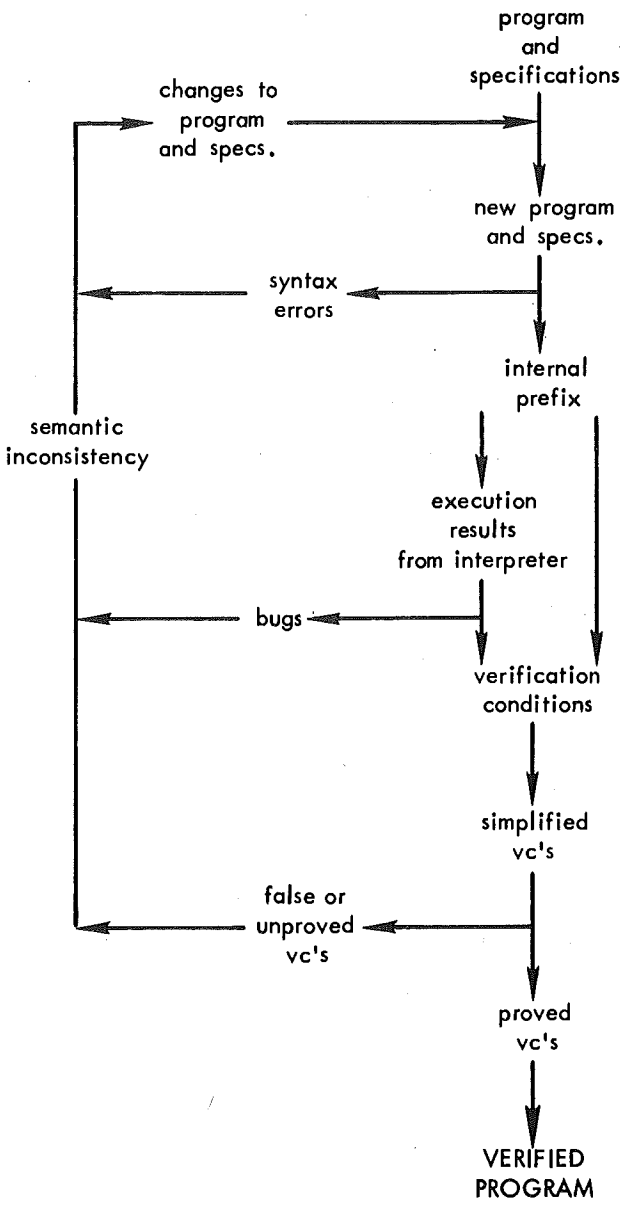

Figure 2.3 Outputs from components of program design and verification system

simplification and substitution is provided with minimal effort by the existing Reduce capabilities; the remainder is provided by easily constructed Reduce procedures. The pattern-matching capabilities of Reduce are exploited to permit the easy input of additional, user-defined simplification rules. Second, Lisp programs written independently of Reduce can be made part of the system; this capability has been exploited to advantage. The previously existing verification condition generator [5] was imported via ARPANET and essentially plugged into the system. The interactive theorem prover [3] that we have incorporated into the system took more effort because it first had to be converted from one Lisp system to another. Third, because Hearn has invested considerable resources in making the Reduce system available on a wide class of computers and operating systems, the corresponding availability of the verification system will be achieved merely by keeping the programming of the system in Reduce.

Currently, the verification system can verify an interesting class of small programs from the program verification literature, from various programming manuals and texts, and from our own test examples. Furthermore, the use of such simple structuring notions as procedures and functions eases the verification task. Although this system has not yet verified all of the programs of other existing program verifiers, we are nevertheless encouraged by the power inherent in Reduce and in the theorem prover that we have just begun to exploit. We remain confident that our system will be able to verify such programs shortly, and that the system will be an extremely flexible and valuable asset in experimenting with assertion languages, structuring methods, and proof methods for verifying significant, real programs, such as a compiler for Pascal, an editor, parts of the verification system itself, or appropriate application programs.

REFERENCES

1 Boehm, B. W., "Software and Its Impact: A Quantitative Assessment," Datamation, Vol. 19, No. 5, May 1973, pp. 48-59.

2 London, R. L., Proof of Wulf's Prime Sieve Program, contained in W. A. Wulf, "ALPHARD: Towards a Language to Support Structured Programs," Computer Science Department Report, Carnegie-Mellon University (in progress).

3 Bledsoe, W. W., P. Bruell, "A Man-Machine Theorem Proving System", Third International Joint Conference On Artificial Intelligence, Advanced Papers of the Conference, 1973, pp. 56-65.

4 Hearn, A. C., "Reduce 2: A System and Language for Algebraic Manipulation", Proceedings of the Second Symposium on Symbolic and Algebraic Manipulation, ACM, 1971, pp. 128-133.

5 Igarashi, S., R. L. London, D. C. Luckham, "Automatic Program Verification I: A Logical Basis and Its Implementation," USC/Information Sciences Institute, ISI/RR-73-11, May 1973. Also accepted for publication in Acta Informatica, 1974.

PROGRAMMING RESEARCH INSTRUMENT

PROJECT LEADER: Louis Gallenson

RESEARCH STAFF: Joel Goldberg
Raymond L. Mason
Donald R. Oestreicher
Leroy C. Richardson

RESEARCH STAFF SUPPORT: R. Jacque Bruninga
George W. Dietrich
Rennie Duge
Oralio E. Garza
Lloyd G. Jensen

RESEARCH ASSISTANTS: John M. Malcolm
Ronald Tugender
Martin D. Yonke

INTRODUCTION

The Programming Research Instrument (PRIM) project has created a fully protected experimental computing environment with continuous multiuser access. An ARPANET-based multiaccess system, PRIM allows each researcher to create his own specialized computing engine capable of being changed and adapted to his specific needs. The PRIM hardware and software together provide a working environment in which the user can implement his own computer in microcode and run that computer in his target program environment. PRIM can be used to explore computer architecture, language development, and special-purpose processor design--all of especial relevance to DoD selection and use of computer equipment.

The PRIM facility makes it possible to easily simulate new hardware architectures and designs in microprogrammed software. That is, software can be created for hardware not yet available, and hardware designs may be extensively used and changed even before the prototype stage of development is reached. This should both cut lead time and improve decisions connected with the special-purpose hardware procurement cycle.

To familiarize potential users with the operation of the PRIM system, ISI will provide introductory seminars and an extensive documentation package. PRIM user documentation, consisting of an Overview, User's Guide, MLP-900 Reference Manual, and GPM Reference Manual, is nearing completion and should be available to interested potential users by mid-1974.

This documentation is currently available via the ARPANET at ISI, directory PRIM.DOCUMENTATION. The PRIM Overview has already been published and distributed [1]. Interested individuals will be invited to a user's seminar, scheduled for June, at which time PRIM will be operating in the environment for which it was designed. ISI personnel will continue to be available to assist users and correct bugs within the system as required.

HARDWARE

PRIM's hardware system is based on two processors: the Digital Equipment Corporation's PDP-10 and the STANDARD Computer Corporation's MLP-900 prototype processor (see Fig. 3.1). The PDP-10 and MLP-900 share memory as dual processors; the MLP-900 is a device on the PDP-10 I/O bus (see Figure 3.2).

PDP-10

The PDP-10, connected to the ARPANET, runs under the TENEX time-sharing system of Bolt Beranek and Newman, Inc. on a paged virtual memory. Its processor contains 256K words of 36-bit memory. The I/O operations performed by TENEX include file, terminal, and network handling, swapping, and all other accesses to peripheral devices.

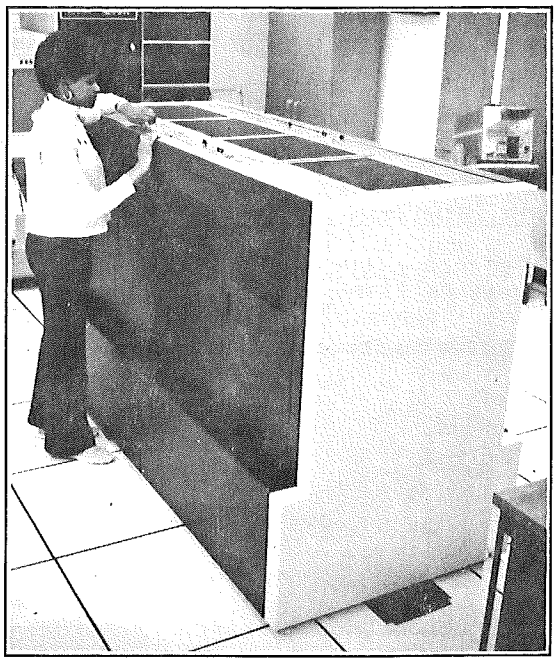

Figure 3.1 The MLP-900

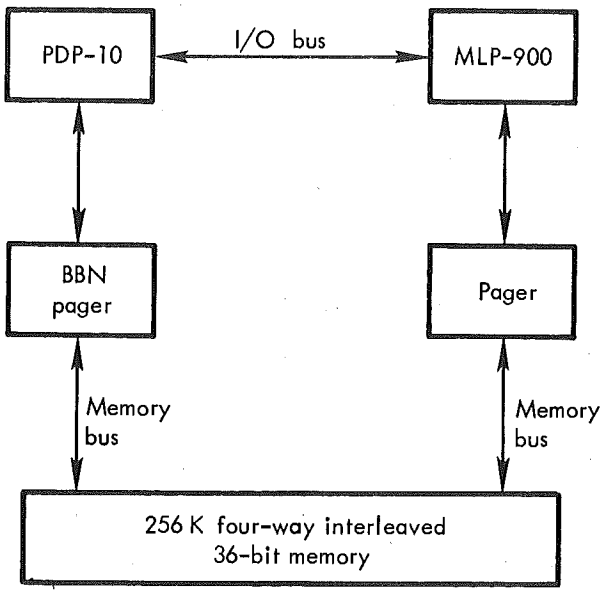

Figure 3.2 Basic PRIM configuration

MLP-900

The MLP-900 is a vertical-word microprogrammed processor that runs synchronously with a 4-MHz clock. It is characterized by two parallel computing engines: the Operating Engine (OE), which performs arithmetic operations, and the Control Engine (CE), which performs control operations (see Figure 3.3). The OE contains 32 36-bit general-purpose registers for operands and 32 36-bit mask registers to specify operand fields. A 1K 36-bit high-speed auxiliary memory is associated with the OE. The CE contains 256 state flip-flops, a 16-word hardware subroutine return stack, and 16 8-bit pointer registers.

The MLP-900 is accessible only through the PDP-10 as outlined above (i.e., the I/O bus and shared memory); no provisions have been made for direct connection of any peripheral devices.

OPERATING ENGINE (I/O, arithmetic, logic)	CONTROL ENGINE (Branches, testing)
General registers 32x36 bits, R.0-R.37	Flip/flops 256x1 bits, F.0-F.377
Auxiliary memory 1Kx36 bits, A.0-A.1777	Pointer registers 16x8 bits, P.0-P.17
Mask registers 16x32 bits, M.0-M.17	Subroutine stack 16x16 bits, S.0-S.17
CONTROL MEMORY 4Kx36 bits	

Figure 3.3 MLP-900 configuration

The introduction of a microvisor state has been of major importance to the PRIM project. Prior to this project, little had been done toward making the multitude of available microprogrammed processors potentially sharable resources. This initial experiment goes a long way toward making microprogrammed processors widely and inexpensively available.

The major hardware effort was conducted in four broad areas:

1) Reconfiguring the MLP-900 mainframe for necessary expansion, improved reliability, better cooling, and improved power distribution.

2) Interfacing the MLP-900 to the PDP-10 (including I/O and memory bus interfaces and a paging facility).

3) Creating a microprogrammed supervisor (microvisor) state within the MLP-900, with facilities for protection of the privileged resources and appropriate communication to change state.

4) Enhancing the user environment within the MLP-900.

The modification of the MLP-900 and the interface to TENEX (items 1, 2, and 3) were essentially completed during the last reporting period; the details of these operations can be found in Ref. 2.

Enhancement of User Environment

Although this year's hardware effort consisted primarily of debugging the MLP-900, a few hardware developments are worthy of mention. These include language boards, Control Memory Address Compare (CMADRC), Virtual Memory Address Compare (VMADRC), auxiliary memory, and streaming mode transfers.

As originally conceived, the language boards were to perform a variety of functions for the MLP-900 relating to the interpretation of target instructions. These boards were to be designed and uniquely used for specific target level languages, one board per language. The functions performed simplified and/or reduced the ministeps required to execute target level instructions. The concept is being modified to generalize the function of the language boards such that one set of language boards will enhance all target level languages and minimize the need for redesign and change of language boards throughout the life of PRIM. Because the functions and registers required by microvisor interaction and user control are protected, two types of language boards have been designated: the Supervisor Language Board (SLB) and the User Language Board (ULB). The SLB utilizes language board outputs for task assignments, page faults (communicating with the TENEX system), and MLP-900 mode control. The ULB, which is currently a null board, passes data from target memory to the MLP-900 without modification or interpretation, but will grow as users' requirements become known. All the currently required ULB functions are being performed with MINIFLOW and will be modified as speed, space, and logic eventually become a factor.

The compare registers VMADRC and CMADRC (18 bits and 12 bits, respectively) have been implemented as user debugging tools. These registers can be loaded and compare-enabled to assist the user when debugging software. One register operates at the MINIFLOW (control memory) level, the other at the target (main memory) level.

A 1024-word (36 bits) 200-ns auxiliary memory has been added to the OE

to be used as a cache or scratchpad. The memory is implemented with a Cogar Corporation module identical to that used for control memory, and the memory boards are interchangeable in all MLP-900 memories (control, translator, and auxiliary).

The streaming mode (fast block transfers) from the PDP-10 core to the MLP-900 is operating satisfactorily at an average transfer rate of 330 ns (never more than 350 ns); three-way memory overlap can be achieved. The design goal was to maximize the speed of the the context swapping, taking advantage of four-way memory overlap capabilities of the PDP-10 memory.

SOFTWARE

There are three principal items of PRIM software:

- The General Purpose Microprogramming Language (GPM) compiler.
- The MLP-900 microprogram supervisor (microvisor).
- The TENEX MLP-900 programs, i.e., the MLP-900 driver and MLP-EXEC.

The basic PRIM software architecture is shown in Figure 3.4.

The GPM compiler was esentially completed in early 1973; for a more detailed account of its development the reader should consult Refs. 3 and 4. The major effort of this year, and the major emphasis of this section, is the development of the TENEX software support and the microvisor.

GPM and the GPM compiler

GPM is a high-level machine-oriented language, written in TENEX BLISS, designed explicitly for the MLP-900. As a high-level language, GPM offers a block structure and statement syntax similar to PL/1 or ALGOL. The specific statement types defined in GPM are generalizations

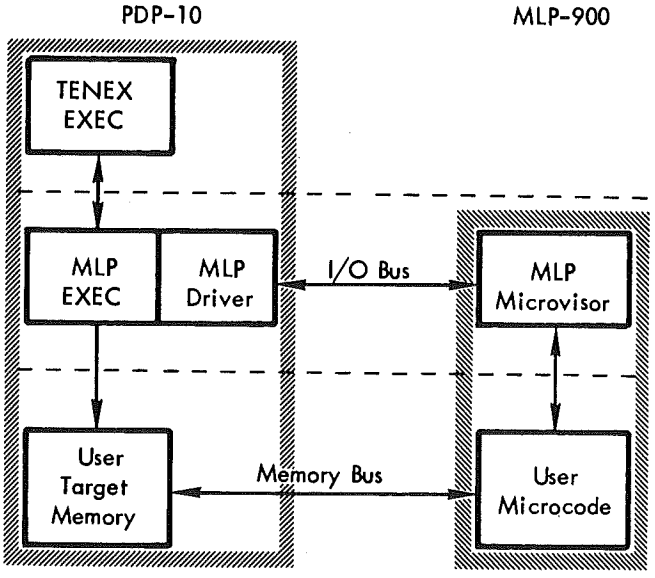

Figure 3.4 Basic PRIM software architecture

of the actual MLP-900 MINIFLOW instruction set; constructs completely foreign to MINIFLOW (e.g., multiplication) do not appear in GPM. As a simple example of MINIFLOW generalization, consider that the result of a GEAR (GEneral ARithmetic) ministep may be shifted left or right only by 0, 1, 2, 4, 6, 8, 12, or 16 bits; in GPM, any shift amount may be specified, and the complier will generate multiple shifts as required.

As the production language for the MLP-900, GPM is constrained to satisfy many of the usual requirements of an assembly language. First, there is a well-defined subset of GPM statements that produce exactly one ministep per statement; the subset is capable of generating all possible ministeps. Second, multi-ministep statements do not generate implicit side effects; for example, a complex arithmetic assignment that requires a temporary register for an intermediate result will generate a compile-time error unless the programmer has explicitly declared some register to be available as a temporary.

The GPM compiler is successfully being used to write diagnostics for the MLP-900 and test user software (emulation of a PDP-10). Experience with the compiler reveals that minor modifications and suggested speed improvements may be required. The improvements will be considered as more measurement data is accumulated and specific critical code is further identified.

MLP-900 Microvisor

The MLP-900 microprogram supervisor (microvisor) is a small, fully protected resident system that controls the MLP-900 and its communication with the PDP-10. It loads and unloads the user's MLP-900 context upon command from the PDP-10, supports paging of the user target program, protects main memory and the rest of the PDP-10 system from user interpreter errors, and provides the interpreter with a few services, such as an extended subroutine stack and calls for external communication. (The microvisor requires 756 (octal) words of control memory, including its Action Request locations.)

The microvisor performs the functions normally expected of an operating system, the difference being that it is written in microcode and supervises the execution of microcode. The microvisor interacts only with the user microcode and the TENEX MLP

driver; it does not provide any facilities for--or impose any restrictions upon--the user target system. User microcode is subject to the restrictions imposed by the user mode MLP-900 hardware.

PDP-10 Support Programs

The PDP-10 TENEX software for support of the MLP-900 consists of a driver to control communication with--and sharing of--the MLP-900, and a subsystem (MLP-EXEC) to allow interactive access to the MLP-900 for a user at a TENEX terminal. The MLP driver and its TENEX JSYS's comprise the interface to the MLP-900 used by MLP-EXEC.

The TENEX MLP-900 Driver. As mentioned above, access to the MLP-900 from a TENEX process is accomplished via the MLP driver in TENEX. The MLP-900 driver is the extension in TENEX of the microvisor; all communication with the MLP-900 goes through the driver. While new microcode "machines" can be designed and debugged under the MLP-EXEC, completed ones will work directly through their own terminal subsystems, which will communicate directly with the driver. Communication with the driver is accomplished through a series of JSYS's which mimic (roughly) the JSYS's for subsidiary fork control. The two principal elements involved in creating and running the MLP are the MLP context (the user microcode together with all the MLP registers) and the target system upon which the context is to operate. The calling process must build both before establishing access to the MLP.

The context is a structure that contains all the data necessary to load the MLP and begin (or resume) execution of the desired microcode. It includes not only an image of the MLP-900 control memory, but also the internal MLP-900 registers and some cells used by the driver to implement MLP-900 communication with the PDP-10. The context is 10 memory pages (5120 words) long, and must begin on a page boundary in the caller's address space.

The target system is the memory upon which the MLP context is to operate. It is defined as a TENEX fork (or process), either the caller or a subsidiary fork established solely for this purpose. Typically, the target system fork (SFORK or SFRKV) will never be started on the PDP-10; it exists to define an address space for MLP execution.

To protect the ISI TENEX system and lessen the impact of MLP debugging, both hardware and software, the initial version of the driver has been implemented almost entirely as a normal user process rather than as part of the TENEX operating system. This preliminary driver is being used in debugging the entire system, including the interfaces between the microvisor and the driver, and between MLP-EXEC and the driver. While the differences between preliminary and final driver are transparent to both the microvisor and the user microcode, there are some unavoidable differences for the calling TENEX process. MLP-EXEC is aware of the differences, and handles them properly; to the user of MLP-EXEC, the only visible difference is that the response time is longer.

MLP-EXEC and Its Commands. MLP-EXEC is a user program, called via TENEX, written primarily in BLISS. The program basically consists of two modules: the I/O handler (which includes file access and target memory allocation) and the debugging facility (MLP DDT). The MLP-EXEC commands assume a familiarity with TENEX Exec commands; a subset of TENEX commands is implemented for functions similar to those of the TENEX Exec.

MLP-EXEC provides an environment in which the user at a terminal can compile, load, execute, and debug MLP-900 microcode in a manner similar to that used for debugging programs on the PDP-10. In addition, he can create and debug target programs and environments--although these tools must be provided at a very primitive level, since MLP-EXEC cannot know the nature of the target environment.

The MLP-EXEC "ready" character, ">," signals the user to enter a command. Commands to MLP-EXEC can specify any of several types of actions:

1) Controlling the loading, execution, or debugging of the MLP context.
2) Controlling the loading and debugging of the target system.
3) Setting up the input/output files for the MLP.
4) Providing access to the TENEX within MLP-EXEC as a convenience.

All the commands for user context manipulation begin with a period (".). These include LOAD, RESET, CONTINUE, RUN, SAVE, GET, and DDT commands. All of the commands for the target system begin with the character "/" and use standard TENEX

subsystems in responding to the command (i.e., /LOAD invokes the standard TENEX loader to load a relocatable binary file into the target system's address space). These include GET, MERGE, DDT, SAVE, SSAVE, and RESET commands.

The command format, key words, arguments, and separators are identical to those used in TENEX. MLP-EXEC prompts for each field required by the user's command, and the escape terminator will complete abbreviated commands. Additionally, two characters (Control T and Control C) act as commands in themselves to control MLP execution and to provide status information on the MLP. Editing control characters are also included to edit command key words and arguments.

User Interpreter and Target Program

The user's interpreter is a program written in GPM to run on the MLP-900; it defines a (re-entrant) MLP-900 control memory image. This image, together with all the nonprivileged registers and flip-flops within the MLP-900, comprises the MLP-900 context; user's contexts are loaded and unloaded as the MLP driver shares the MLP among different users.

The context defines the user's interpreter (or target machine) and operates upon the user target program in a totally arbitrary way. The only constraint upon the target program is that it fit into a 512K, 36-bit (virtual) memory space.

FUTURE EFFORT

The ISI TENEX environment currently includes a KA-10 and a KI-10 CPU, one as a backup for the other in providing TENEX service to the ARPA user community. Initially the PRIM project will use the backup CPU to provide the flexibility required by the development effort with the user interaction without jeopardizing the service operation. The KI/KA CPU compatibility introduces problems requiring modifications to the microvisor, especially in the page faulting routine, which will be performed during the next few months. The remaining hardware efforts are to investigate faster clock speeds (currently 4 MHz) and to design the general-purpose language boards. The system integration, documentation, and software debugging is currently nearing completion.

The major effort of the near future

will be that of maintaining the PRIM facility and of indoctrinating users. The introduction of users via the ARPANET will be the final system test and will help to identify possible areas of improvement. Initially, the MLP-900 will operate with a single user with locked pages of target memory. With increased confidence and experience the PRIM system will evolve into the time-shared resource originally specified. User interest has been expressed by military services interested in emulation of CPU's currently in the procurement cycle, by researchers interested in direct high-level language interpreters, and by computer science instructors preparing curricula for microprogrammed processor design. PRIM should be supporting one of these users by June of 1974.

REFERENCES

1 Richardson, L. C., PRIM Overview, USC/Information Sciences Institute, ISI/RR-74-19, February 1974.

2 Annual Technical Report, May 1972-May 1973, USC/Information Sciences Institute, ISI/SR-73-1, 1973.

3 Oestreicher, D. R., General Purpose Microprogramming Language Reference Manual, USC/Information Sciences Institute (in progress).

4 TENEX JSYS Manual, Bolt Beranek and Newman, Inc., Cambridge, Mass., 1973.

PROTECTION ANALYSIS

RESEARCH STAFF: Richard L. Bisbey, II
Jim Carlstedt

CONSULTANTS: Richard J. Feiertag
Gerald J. Popek

RESEARCH STAFF SUPPORT: Betty L. Randall

INTRODUCTION

This project has developed and is continuing to refine methodologies, techniques, and standards for the analysis, testing, and evaluation of the protection features of operating systems. Its goal is to provide answers to the question of what tests and procedures can and should be applied to operating systems in order to determine (1) to what extent a given system meets its requirements for preventing unauthorized or improper operations and (2) how systems can best be designed and implemented to reflect such requirements. The research directly supports the software security requirements issued by DoD security policymaking agencies.

The Protection Analysis project was formed in September, 1973 by the union of the Empirical System Study and Protection Theory projects, reported last year as part of the Software Assurance project [1]. The former focused on near-term solutions, while the latter was a more deductive approach to discovering more complete and systematic evaluation methods. These are reported separately below, followed by a report of activities of the Protection Analysis project since September.

EMPIRICAL SYSTEM STUDY

During 1972, the Empirical System Study group at ISI devised a method for finding security errors in operating systems by identifying types of "error patterns". This approach was based on the empirical observations that (1) the majority of errors in operating systems can be categorized using a limited number of generic error types and (2) it is easier to find errors by systematially searching for instances of generic error types than by randomly searching for logic flaws.

Field Tests

In late 1972 a test was made, using the Multics operating system as a test subject, to determine the usefulness and effectiveness of the method. As previously reported [1], security errors were found, and the method was shown to be effective. A second test was conducted in the summer of 1973 using the error pattern approach, again using Multics as a test subject. The purpose of this test was again to verify the error pattern approach and to accumulate more information on its use. During this test, a member of the Multics staff was added as a test participant, having been briefed on the expected error patterns. The protocol used for the test was the same as that reported in Ref. 1, i.e.:

- All system information was made available to the test participants prior to the test.
- The system being tested was not modified during the test.
- Proposed errors were verified by logical analysis and not by writing or running programs to exploit the errors on live systems.

The purpose of the protocol was both to expedite the test and to eliminate any possible "gaming" situations in which systems personnel could retroactively alter the system to disguise or eliminate faults.

The goal, which was to find a system error using the above procedure, was immediately achieved. A design error in Multics affecting operating system security was discovered, necessitating a redesign and reprogramming of portions of the system. The error was reported to both MIT and Honeywell, and changes are currently being made to correct it. The results of this and the previous test, as well as insights into the use of error patterns, have formed the basis for an automated protection evaluation system based on generic error types, which is described below.

Encapsulation

Enormous sums are presently invested in computing equipment and operating system software. The need for security in those systems is strongly felt in government and business as well as in the military. The problem is intensified by the current unreliable state of

information controls in contemporary systems.

Today a certifiably secure multiuser operating system does not exist. No operating system has been able to withstand malicious attacks by skilled penetrators. Retrofitting these systems (in the general sense of repairing the respective operating systems in all their various versions) is an enormous task, not well understood. To date, all such attempts have failed. Even systems created with security as a major design parameter have been easily penetrated. In general, retrofitting the multiple versions of various operating systems by revising their code is impractical.

Nevertheless, the security problem of existing systems is important and will not diminish in the coming years. The only solution now available to those installations requiring reliable protection has been physical isolation: to have separate operating systems for each security category or level and run them sequentially. For the military, a separate operating system is required for each of the four security levels. For the typical commercial installation, separate operating systems might be required for payroll, accounts receivable, and general computing. A considerable amount of useful machine time is thus wasted changing systems, rigid schedules must be established, and sharing can be achieved only through off-line procedures, controlled adminstratively.

The Empirical System Study group has developed an approach to the security retrofit problem for batch and remote job entry (RJE) systems. It is fairly simple and appears economical; in addition, the same solution will provide certifiable security for a variety of manufacturers, computers, and operating systems. A small minicomputer controlling several banks of switches is added to a currently existing configuration. These switches, which are placed in the read/write circuit of the peripheral devices, allow the minicomputer to enable or disable those devices (see Figure 4.1). As in the case of virtual machine systems, each encapsulated user program will run with its own operating system. Depending on which operating system is currently in control of the production CPU, the minicomputer sets the device switches accordingly, so that the operating system can physically access only appropriate devices. For many existing peripheral devices, such switches

already exist. Estimates of the cost of the above encapsulation unit, including the one-time expense of system design and software verification as well as the per-installation expenditure for hardware, appear to be eminently reasonable.

The unit has several attractive aspects. Original software can run without changes, read-only sharing is provided, and scheduling flexibility is increased. Most important, the software relevant to security is small enough that its security properties can be formally verified; also, its code is isolated and protected from any other software. In this fashion, encapsulation provides a certifiable, reliable means for multilevel computer security on existing batch systems, possibly ending their retrofit problem. This work is documented in more detail in Ref. 2.

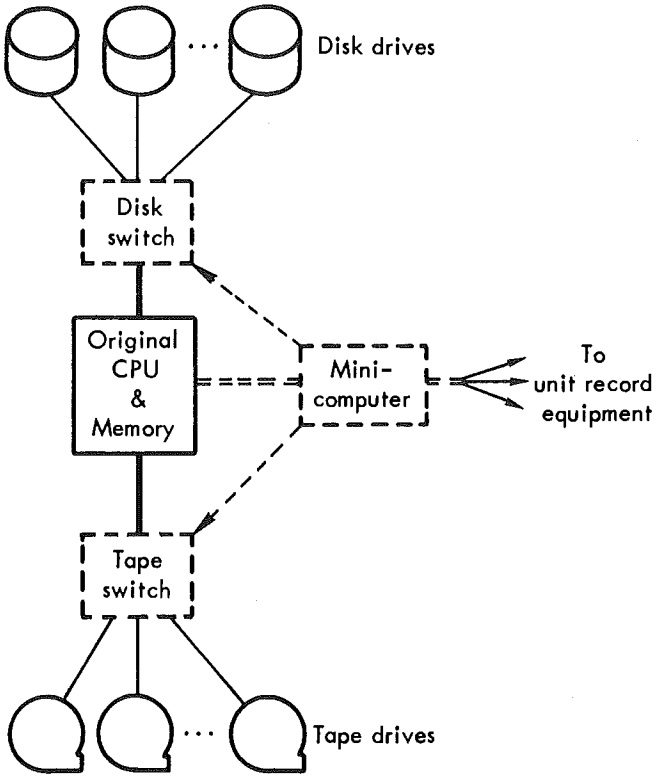

Figure 4.1 Security encapsulation unit

PROTECTION THEORY

Protection is that central aspect of computer security concerned with the control of operations within the domain of the operating system, i.e., by internal processes on internal objects. Although the security of a computer system depends on the correctness and completeness of its protection mechanisms, there is a well-known shortage of methodologies either for designing these mechanisms or for evaluating existing systems for errors.

The goal of the Protection Theory project was to design and develop an evaluation method that would be both thorough and systematic, properties that current methods lack. A method specific to the protection problem is likely to

become cost-effective sooner than tools intended for the verification of computer programs in general [3]. The project was begun in January 1973, starting with an extensive survey of the field to determine the current state of evaluation methodology and the boundaries of the protection problem. The work reported here was concerned with two major areas:

1) Outlining an evaluation scheme. The result is an independent policy-based (IPB) evaluation method, utilizing concepts from the field of structured design.

2) Formulating the protection problem and deriving a notational basis for the expression of protection policy. A level-independent formulation was derived.

The IPB Evaluation Method

From the point of view of structured design [4], system development is a process of transformations on some hierarchy of objects and specifications of increasing concreteness and decreasing abstractness toward the lower levels, ordered by the relation of representation (see Figure 4.2).

With respect to protection, the more abstract elements are those of policy and the more concrete those of mechanism. If, during development, all the representational relations were maintained explicitly, the protection mechanisms of a target system could be evaluated constructively. Otherwise, assuming that the elements at some level of policy (Psys) are stated as explicitly as those at the mechanism level (Msys), evaluation must consist of an independent comparison of Msys with Psys--hence the name "IPB".

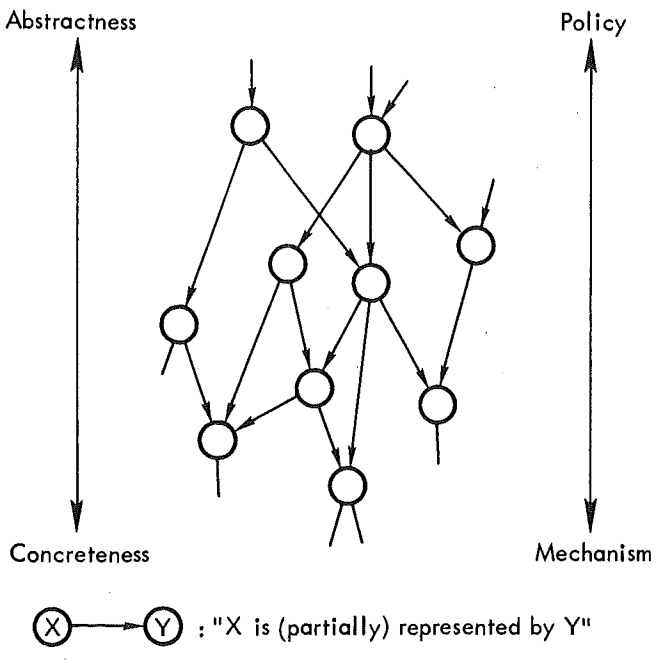

Figure 4.2 Representational hierarchy

Such a method requires a separate hierarchy (see Figure 4.3) whose ordering relation is one of logical implication from policy to mechanism. An input to such an evaluation is the set Psys of target system policy. The implications define a mapping that produces a corresponding set of reference specifications (Mref) at the mechanism level, with which Msys can be compared for correctness. If the IPB method is to be applicable to a variety of systems, the domain of IMPL must be a collection of protection policies that encompasses the policies of all those systems, and its range must be a generalized set of protection mechanisms, such as that proposed in Ref. 5, suitable to enforce them. A major difficulty is that in order to compare Msys with Mref, the system-specific description of the former must first be "normalized" into generalized terms.

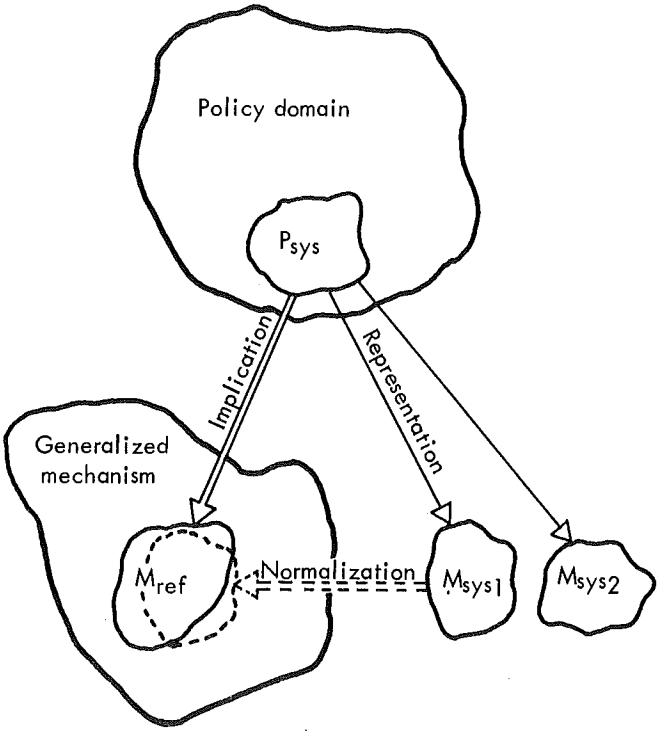

Figure 4.3 IPB evaluation scheme

Policy and its Expression

Though many protection schemes and models exist at the mechanism level, little work has been done at the policy level, so that the explicit basis needed for IPB evaluation is missing. The essence and boundaries of the protection problem are not readily apparent. Among many formulations, the one most appropriate from a number of standpoints is that protection is the prevention of unintended use of certain (protected) objects, which translates in more concrete terms into the following:

- The attachment to each protected object (explicitly or implicitly) of a conditional governing its involvement in certain operations.

- The prevention of such an operation when the conditional evaluates to false.

An operation is a combination {X,W,Y} where X is a process, W is an operator, and Y is the set of operands in the application of W by X. Any of these may be protected objects. Preventing the operation {X,W,Y} means either preventing the selection of W by X to form an activation Wx or preventing the binding of one or more elements of Y to either W (static) or Wx (dynamic).

Taken at the mechanism level, this formulation exhibits the key role of primitive linking and binding operators, which must enforce all occurring conditionals. For the sake of economy, actual operating systems severely restrict the generality inherent in this formulation and also make heavy use of stored representations of the results of condition evaluations, e.g., in the form of "capabilities".

However, because the concepts of this formulation can be interpreted at any level of representation, it is also well-suited to be the basis for specifications at the policy level. This is demonstrated in Ref. 4.

PROTECTION ANALYSIS

In September 1973, the Empirical System Study and Protection Theory projects began a study of ways in which their complementary approaches to protection evaluation could be combined into a single method, with the empirical techniques of the former applied to achieving the goals of the latter. The study was influenced by three observations:

1) The security community lacks a production evaluation tool. Such a tool must be exportable, requiring for its application only a familiarity with the target system rather than a high level of expertise in evaluation techniques; reliable, finding consistently all errors of the types covered; and economical, not requiring large evaluation projects or tooling-up phases. These attributes imply that such a tool will be largely automated or computer-assisted.

2) There is currently no known organized effort to collect and analyze protection errors detected during various evaluations of existing operating systems. Such an

effort could have significant value to the computer security field, both in facilitating future evaluation efforts and in identifying classes of errors to be avoided in the development of future systems.

3) Protection errors do fall into distinct classes or "patterns."

Guided by these observations, the study focused on the way in which the output of protection evaluations (i.e., errors and their patterns) can be utilized as feedback in the development and improvement of an evaluation method itself. The result was the design of a systematized error-driven evaluation scheme that utilizes the output of error analysis both directly, to govern the evaluation process, and indirectly, to increase the comprehensiveness of the tool based on this method. The two projects merged into the Protection Analysis project, using the collection and analysis of protection errors as the immediate phase in the development of the evaluation tool.

Error and Pattern Processing

This work, which began in early 1974, consists of the following activities:

Error Collection. The primary input is an "error base" consisting of informal descriptions of protection errors --detected as well as potential--both contributed by evaluation personnel and gleaned from the literature and other sources. In addition to the Empirical System Study project reported above, current sources include Project RISOS at Lawrence Livermore Laboratory, the Computer Systems Research Group at MIT, and a Security Analysis Group at The Rand Corporation.

Error Analysis. The analysis proceeds in two steps: identification of the raw pattern and identification of features. The raw pattern is the (minimal) set of conditions that together constitute a possible error: either those holding for the operating system as a whole or possible sequences of actions that can occur more locally. "Raw" means described in terms of a particular operating system or line of systems. Features are the individual conditions of the raw error, as well as the operating system entities they involve. The feature terminology becomes the technical vocabulary of the project.

Classification and Generalization. These two activities are closely related. Each raw pattern is compared with others in the current set to determine differences and similarities. As the set grows, a grouping or classification occurs, with the patterns in each group seen as instances of some generalized pattern, described in correspondingly more abstract terms. As this process continues, generalized patterns in turn become associated with still more general patterns, resulting in a hierarchy with the most general and abstractly described patterns at the upper levels and the most specialized and concrete ones at the lower levels.

Data Management. The collection of raw errors, the hierarchy of patterns, and the glossary of features will be represented as a computerized data base in such a way that classifying and generalizing activities can be carried out efficiently. The following retrieval functions will be implemented:

- The retrieval of error description, pattern description, or feature definition by name.
- The generalizing pattern of a given error or pattern, respectively.
- The errors or patterns of which a given pattern is a generalization.
- The errors or patterns in which a given feature occurs.

The facilities of ISI's TENEX operating system will be used to maintain this data base.

The Error-Driven Evaluation Method

Protection evaluation is regarded here as a two-step process: normalization and comparison. The first is the identification, extraction, and formalized description of the protection aspect, which is generally embedded in the large and complex volume of information representing a target operating system. The second is a comparison of the normalized protection description with a given set of reference information to determine the presence of errors.

In the error-driven scheme, these steps are directed by the output of the error and pattern processing activity (see Figure 4.4). Normalization is directed by the generalized feature set. It is essentially a systematic search of the target system for instances of the generalized features. This is basically a

manual process, but can be computer-assisted by means of a program similar to computer-assisted instruction. Comparison is directed by the generalized pattern set; it is a search of the normalized desciption for combinations matching any of the given error patterns, a process that can easily be automated. The output is a list of error indications.

The error-driven method is similar to the IPB method described under Protection Theory above in that a normalized description of the protection mechanisms of the target system is compared with a set of reference information. The most important difference is that with the IPB method the reference information is the complete set of generalized mechanisms logically implied by the stated policy of the target system, so that the completeness and consistency of the target mechanisms can be determined. With the error-driven method, only errors of the types represented by the given patterns will be detected.

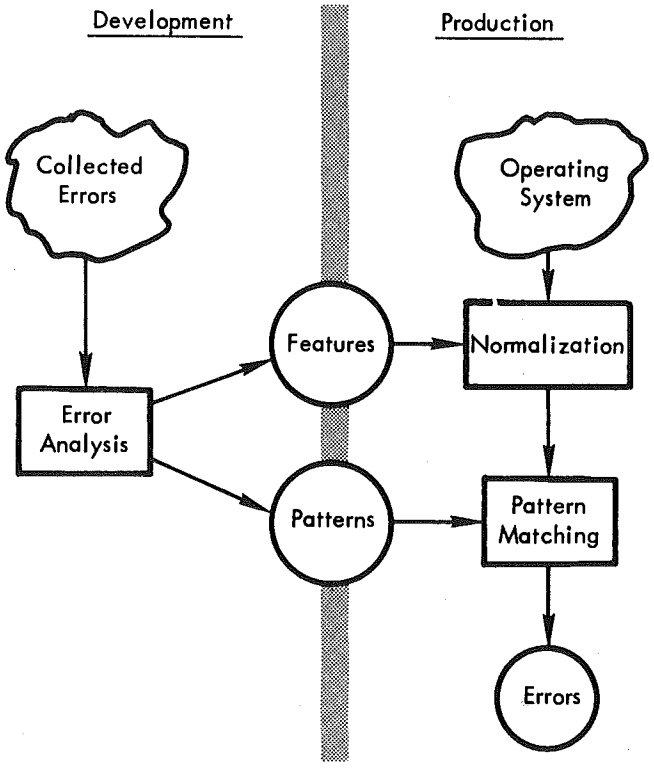

Figure 4.4 Error-driven evaluation process

The advantages of the error-driven method are the following;

- It systematically capitalizes on accumulated protection evaluation experience.
- It is applicable in the short-term future as a standard evaluation tool.
- As the error and pattern processing activity continues and its scope expands, the comprehensiveness of the method increases correspondingly.
- Because of the insights gained, the error and pattern processing activity is also valuable as a contribution to protection theory and to the development of the IPB method,

especially in identifying different protection policies and understanding their implications. The error-driven evaluation method provides a production tool for testing operations that is suitable for use as a test standard.

REFERENCES

1 Annual Technical Report, May 1972 - May 1973, USC/Information Sciences Institute, ISI/SR-73-1, 1973.

2 Bisbey, R. L., and G. J. Popek, Encapsulation: An Approach to Operating System Security, USC/Information Sciences Institute, ISI/RR-73-17, October 1973.

3 Linden, T. A., "A Summary of Progress Toward Proving Program Correctness," AFIPS Conference Proceedings, 1972 Fall Joint Computer Conference, Vol. 41, Part 1, AFIPS Press, Montvale, N.J., 1972, pp. 201-211.

4 Carlstedt, J., Toward Explicit Policy: A Structured Approach to Design and Evaluation of Protection Systems, USC/Information Sciences Institute (in progress).

5 Jones, A. K., Protection in Programmed Systems, Carnegie-Mellon University, Department of Computer Science, June 1973 (available from NTIS as AD-765-535).

COMMAND AND CONTROL MESSAGE PROCESSING TECHNOLOGY

PROJECT LEADER: Robert H. Stotz

CONTRIBUTING STAFF: Thomas O. Ellis
Louis Gallenson
John F. Heafner
Donald R. Oestreicher

CONSULTANT: J. Clifford Shaw

RESEARCH STAFF SUPPORT: Norma B. Johnston

INTRODUCTION

This project explores the use of advanced computer and communication techniques in military environments. The use of packet-switched digital networks to provide a message handling service (exemplified by the ARPANET) has immediate and significant usefulness to the military community. The possible applications of such a service have served as the principal focus of the work to date.

Current military communications systems use the Autodin system to handle formal messages at electronic speeds between command communications centers. However, manual message delivery from the communication center to the addressee typically requires from several hours to an entire day. Preparing a message at the origination point can involve hours of an action officer's time (and several days of total elapsed time) to obtain the necessary approvals, each of which may require modification of the message. This coordination of messages generally has to be done in person, which consumes large amounts of time. In fact, nearly all communication about classified material must be handled in this manner because of the lack of secure telephones.

A specific example of such an environment was the object of a study performed by ISI in the spring of 1973 [1], investigating the possible application of network technology to the COTCO (Consolidation of Telecommunications

on Oahu) program. COTCO is a DoD effort to improve military communications on Oahu. The ISI report transmitted via ARPA to the Joint Chiefs of Staff recommended that the present largely manual message transmission system be replaced by an interactive computer-based system providing direct writer-to-reader service for some 6000 action officers on 24 military bases.

The system proposed was based upon an ARPA-like network connecting 2000 widely distributed CRT terminals to five message processing computers. Such a system would not only improve existing services but also provide several new capabilities as well: informal communication channels to aid in conducting everyday business; faster coordination of formal messages transmitted via Autodin; and broad support services related to communications, such as facilities to file and retrieve messages by user-defined subject titles, suspense (tickler) files for action, and automatic message status reports. A mechanism for automatically creating duplicate files was proposed, as well as dual connections between the host computer and the Intermediate Message Processors (IMPs), so that no equipment failure could interrupt communications to a large segment of users. Actual installation of a test model of the system on Oahu was proposed to prove the feasibility of the approach.

The Navy, which has been assigned the task of implementing the COTCO program, currently has the ISI plan under consideration. Interest has been expressed in performing, with the cooperation of ARPA, a test of an interactive system such as the one proposed.

ISSUES FOR IMPLEMENTING AN AUTOMATED MESSAGE SERVICE

In the meantime, the Command and Control Message Processing Technology (CCMPT) project has addressed the issues involved in the implementation of automated message handling services for other military environments besides COTCO. The goals of such systems are the following:

- Enhance the current formal message service.
- Provide an informal message service.
- Enhance current off-line message handling (letters and reports).
- Enhance functions that support message handling (file retrieval, suspense files, etc.).

In order to achieve the sort of system envisioned, it is necessary to provide the following basic components and attributes:

- Core-system hardware architecture that serves as the foundation for building the service,
- Core-system software architecture and programs whose facilities are easy to learn and operate by users unfamiliar with computers.
- Application software that performs the functions required.
- Reliability of service.
- Security of data.

Although eventually the system must incorporate all of these considerations simultaneously, current research is investigating them independently. The form of the core-system hardware architecture exists today within the ARPANET, although the specific implementation does not meet the reliability and security criteria for a military environment. Most of the other issues have only partial solutions at this time; research programs directed toward completing these solutions are being identified or are under way.

HARDWARE ARCHITECTURE

The core system must allow interactive response, connection to many terminals at distant sites, and processing power for the message service. The system architecture is shown in Fig. 5.1. Terminals connect to Terminal Interface Processors (TIPs) like those in the ARPANET. TIPs interconnect through high-speed (e.g., 50-kb) communication lines. This arrangement of TIPs serves as a distributed, packet-switched, store-and-forward network. Through this network the terminals have access to several message processors that supply the message service functions. In principle any message processor on the network can provide that service as long as it can access the user's files (which reside on more than one message processor).

Also shown connected to the network in Fig. 5.1 are the special-function computers INT and WWMCCS. These illustrate the way this message service permits interface to external networks (INT interfaces to Autodin) and task-oriented systems (WWMCCS is a special-purpose command and control computer system). By extending this concept to include other networks and

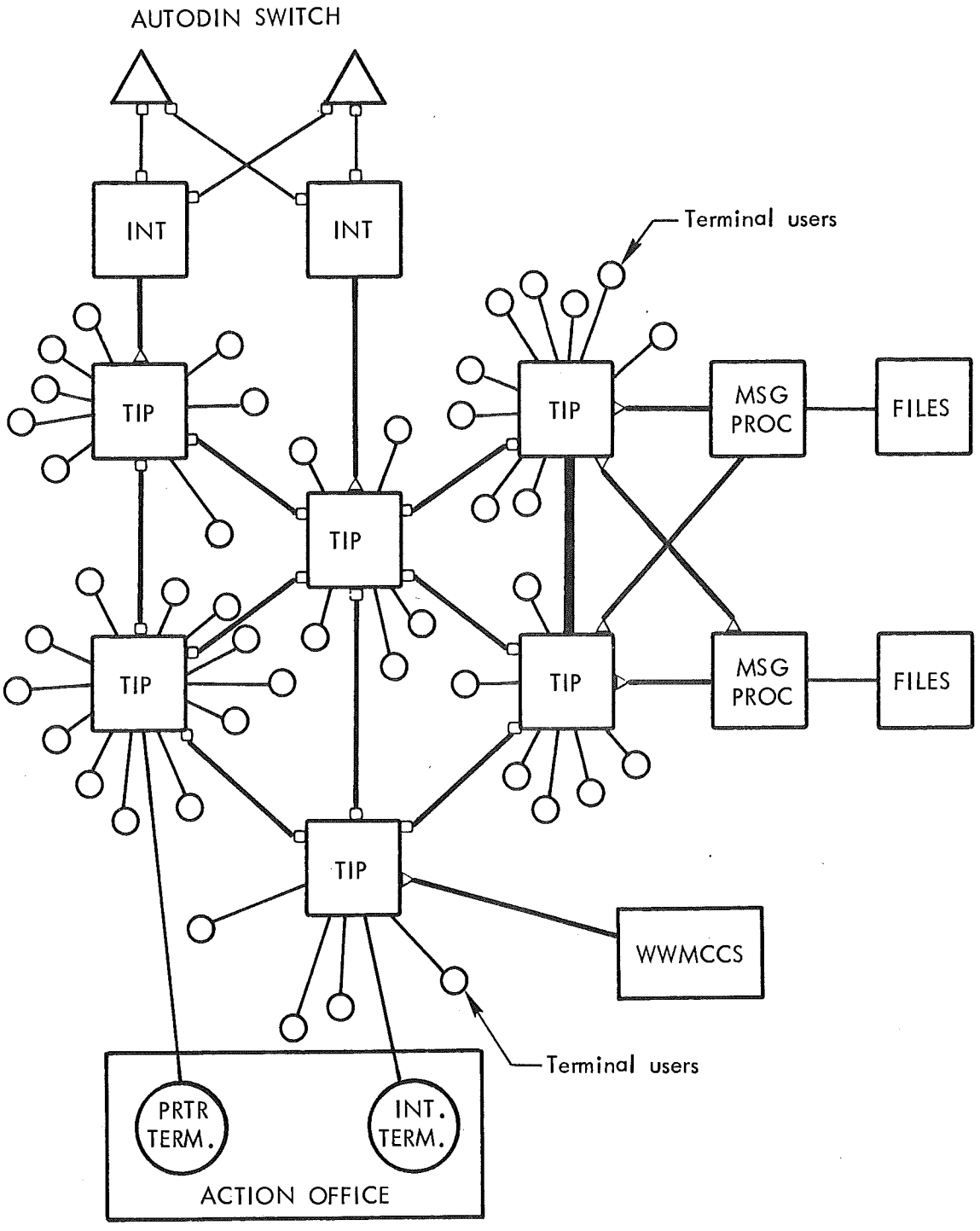

Figure 5.1 Abbreviated block diagram of proposed communication system architecture.

other task-oriented computer systems, a mechanism is provided for integrating into a single framework the multitude of independent systems being built in the military today.

In this scheme a terminal always talks to and through a message processor. If the user engages in a conversation with another terminal or a special purpose computer, all of the message processor functions (editor, files, etc.) are still at his fingertips.

This architecture provides redundant paths for messages, which results in high inherent reliability. Its flexibility in terms of scalability and performance optimization makes the approach extremely applicable to the military community.

SOFTWARE ARCHITECTURE

The software architecture provided must, of course, support the functional operation of an interactive message handling service. This service must possess good response time and be consistent and easy to use. It must also provide data security and highly reliable service.

The CONNECT system being developed at ISI is a human-factors-oriented research system capable of encompassing a message service such as that described here. CONNECT is discussed in more detail in Section 6 of this report. Briefly, however, it is an on-line comunication service designed to interact with its users in a way that seems natural to them. It gives the appearance of personalized service, makes extensive use of tutor facilities to guide the user, and employs a consistent language for interaction that is easy to learn and use.

The application program for an interactive message service is that part of the total enterprise most apparent to its users as they employ it. To understand users' needs in sufficient detail, a team of four ISI staff members spent three days at CINCPAC headquarters at Camp Smith on Oahu in February 1974 in an intensive study of procedures currently employed for generating and distributing messages.

The required functions of a message service identified by that study include facilities to aid in message preparation, transmission, and reception. Sophisticated text editors can make it easier to prepare the various initial

drafts of a message. Coordination of the message (i. e., getting the necessary approvals) up to its final release can be automated, eliminating the present need for hand-carrying. The routing of a message can be assisted through an automated "directory assistance" service. Message transmission will be virtually immediate, regardless of the distance covered. Many aids can be provided the addressees for reading, printing, filing, and retrieving messages. An alerting mechanism will call a user's attention to the arrival of a high-priority message. Facilities for scanning messages, readdressing them (sending on to others not on the address list), and informing the originator of action taken will also prove useful. Other requirements identified were various facilities for status inquiry, programming, responsibility tracing, accounting, training, and interface to off-line processes.

Many of the functions identified are straightforward and already exist in present systems. Others are more subtle, requiring research into methodologies for their implementation. As a part of the CONNECT development, the existing pieces are being integrated into a single system to which are being added the necessary programs to conduct the research required.

An experimental message service within an actual military user community would provide an excellent basis for conducting some of the needed research on the application program. In addition, if the user community were carefully selected, it would provide the opportunity of performing some direct technology transfer without the usual intervening development cycles. Investigation of potential sites for such a test is currently under way.

SECURITY

System security must be considered at several levels. One aspect is simply control of access to the computer, files, communications links, and terminals. A second is radiation security (i.e., the shielding of electromagnetic and acoustic emission from the active elements of the system). A third is "privacy" or operating system security, so that one user cannot accidentally or intentionally have access to files to which he is not authorized. This last consideration extends also to reliability in that one user should not be able to render part or

all of the system useless by any operation he can perform at a terminal. Nor should any load condition render the system ineffective to critical users.

With a terminal-oriented service such as that proposed for COTCO, security can be violated through a large number of devices (i.e., all terminals and communications lines). This fact necessitates either stringent access control or a multilevel plan of system security. If a system can be developed that cannot be broken, it can be used for both unclassified and classified messages: communications lines and terminals that do not carry classified traffic will not need special security measures. Otherwise, all users must be cleared to the highest security level, and all communications lines and terminals must be provided the same level of security control. Clearly, a multilevel secure system is an important ingredient to realizing the full potential of message technology in the military community.

Even in a multilevel secure system, those terminals that handle secure traffic will need to meet the access and radiation restrictions. Radiation control of CRT terminals generally requires expensive filtering and shielding. Display technologies that do not require refreshing in a fixed time pattern are much less prone to deciphering. For this reason, plasma and other direct-view image-storage displays are being investigated as potential "secure" terminals.

Communication lines between terminals and TIPs and between TIPs and TIPs must also be protected from being read by unauthorized persons, which can be done by specialized "hardened" wires for short runs through moderately controlled environments. Otherwise, encrypting devices can be used that scramble the data on the line so it is unintelligible except to a matching decryptor. This encryption/decryption equipment currently tends to be rather expensive ($10,000 to $15,000 apiece). Since the number of lines between TIPs is relatively small, encrypting these is reasonable; however, encryption at the terminal level appears extremely costly. One approach to solving this problem is to install local multiplexors to reduce the number of encrypted lines. For environments in which this is difficult, new technology must be applied to develop less expensive encryption devices.

Encrypting between TIPs on the network will serve no purpose unless the TIPs themselves are also secure--which current TIPs are not. Designing a certifiably secure TIP is a challenging but achievable feat (Autodin, for instance, is considered secure). It involves security clearances for programmers, careful design, classified program listings, and complete analysis and testing.

An alternative approach to TIP security is to develop software that can be proved to be secure (see Section 2 of this report). To date no such system is known to exist, but several programs are under way (at Mitre Corporation and UCLA, for example) to develop secure operating systems for minicomputers.

Another alternative is to encrypt secure traffic at the terminal and the message processor only and have the network operate in the clear on the encrypted traffic (i.e., all the network header data will be in the clear while the body is encrypted). This is somewhat awkward operationally and requires changes to the ARPA TIP software that currently does interpretive functions on the data. It also opens the information about the flow of traffic within the network to possible penetration. For many military purposes, this too is classified information.

The Message Processor also must meet rigid security specifications. Because there are relatively few Message Processors in the system, the problem of physical access and radiation security is manageable. However, development of an operating system that provides multilevel security with the complexity required for message processing is another major project. If the system is closed (not connected to other systems) and is limited to transactions only (no programming allowed) it appears to be feasible. Security of full general-purpose operating systems will probably have to await a breakthrough in the development of mathematically provable secure systems.

RELIABILITY

The current reliability of the network and host computers, although possibly adequate for research, will not satisfy the requirements of an operational military message system. Unfortunately, the various factors underlying the reliability of such a complex system as

the ARPANET are imperfectly understood. A program is being outlined to determine how to collect the data necessary to better understand these factors, which will help in formulating specific action to improve network reliability.

It is imperative to produce not only a program to reduce the probability of component failure in the system, but also a means to recover from failures smoothly and rapidly. If a terminal, encryptor, communications line, or TIP in a system such as COTCO fails, the Message Processor will retain the state of the user so that he can recover it later on a different channel. If a Message Processor fails, the total system must recognize this fact and switch his connection to a backup Message Processor. In order to preserve the files in this case, every file written by a Message Processor will be sent to a backup Message Processor as well. When a Message Processor fails and a user's Job is switched over to a back-up Processor, it is important to maintain as much apparent continuity as possible to the user. This implies a process residing outside the user's primary Message Processor that is monitoring the status and context of the user's activity. This process then controls the switchover and keeps the user informed of events as they happen. These recovery functions, which are not currently implemented in the network, must be completed before a reliable system can be provided.

CONCLUSION

The CCMPT project is currently in a study phase, identifying the problems and the opportunities for message processing systems in the military environment. A program outlining detailed areas for research is now being prepared. In addition, we are exploring opportunities for joint development with military users that will apply our message technology in experimental form to actual military situations.

REFERENCE

1 Ellis, T. O., L. Gallenson, J. F. Heafner, J. T. Melvin, A Plan for Consolidation and Automation of Military Telecommunications on Oahu, USC/Information Sciences Institute, ISI/RR-73-12, May 1973.

INFORMATION AUTOMATION

PROJECT LEADER: Donald R. Oestreicher

RESEARCH STAFF: John F. Heafner
Richard C. Mandell
Jeff Rothenberg

CONSULTANTS: Russell Abbott
J. Clifford Shaw

RESEARCH STAFF SUPPORT: Katie Patterson

RESEARCH ASSISTANT: Larry Miller

INTRODUCTION

Many large segments of the military (most critically, perhaps, Command and Control) depend as heavily on the transmission of information as on the transportation of people and materiel. At present, sending and responding to a typical military message (using a variety of manual and semiautomatic techniques) requires, at best, days. Using an automated computer-based sender/receiver on-line message service system, the same communication could be completed within minutes.

The technology exists today to produce systems for automatic communication. What does not exist is a methodology for making such systems effective and attractive for people unfamiliar with the use of computers. The Information Automation (IA) project at ISI is designing and developing this kind of fully automated easy-to-use interactive communication system to improve the effectiveness of action officers' message generation and sending capability. The proposed system, called Communications Network Nodes Effected by Computer Terminals (CONNECT), is designed to serve a large class of both technical and nontechnical users (including clerical personnel and managers) and could be used to aid many everyday military base communications activities as well.

CONNECT could also be useful to organizations with geographically distant offices needing to maintain constant communication with each other. Because CONNECT will be designed to serve a

variety of users with a variety of communications requirements, it should be able to meet many civil as well as military needs.

Functional Specifications

The functions required of a message service like CONNECT can be divided into five classes: message preparation, message transmission, message reception, information maintenance, and off-line functions. (The word "message" loosely refers to any formal or informal written document.) In addition to the message-related functions, the on-line capabilities include access to computational facilities, since users of CONNECT may communicate with computational processors linked to the communications network.

Message preparation. Message preparation basically consists of composing text and obtaining approval for sending it. The user corresponds with a message creation function by means of a simple dialogue. CONNECT is designed to handle standard formats and can prompt users for necessary completions. When several people are involved in message preparation, the hand-carry phase of coordinating the message for approval is automated.

Message transmission. The message transmission function provides facilities to verify destinations and to monitor the message's status. The service validates the addresses of the list of recipients provided by the originator of the message. If the recipient is not currently accessible, CONNECT can deliver the message to a responsible alternate, or maintain the message until the recipient is available. CONNECT users can ascertain the status of a message by means of such queries as "Has it been read?" and "By whom?", or "Is action pending?", and "By whom?" (Query rights of addressees can be limited by the message originator.)

Message reception. While several modern systems provide adequate message transmission facilities, few give in-depth consideration to the receiver's needs, some of which are the following:

- The service should provide some audible and/or visual alarm to alert a receiver to a new pending message.
- Messages should be ordered to reflect priority, originator, subject, etc.,

making it easier for the user to scan messages

Browsing should be available, either for reading large messages or for familiarizing new personnel with a particular correspondence file. Aids for increasing efficiency in scanning large numbers of messages (e. g., key word searches) should be provided.

Users should be able to forward or copy messages to others, although in special situations the sender may limit this capability.

Recipients should be able to automatically generate feedback to senders, such as "Message not read," "Read but no action," and "Action pending."

Messages should be received in a form that allows them to be incorporated into other documents.

Information maintenance. Information structures such as archives, correspondence files, name and address files, and schedule files are maintained. All information is automatically archived to provide a reliable repository for messages. Users may create specific correspondence files for all messages pertaining to a particular subject. Other users may subsequently address messages to an addressee's particular correspondence file rather than to his general delivery files.

Off-line functions. CONNECT supports the generation of documents such as reports and letters destined for off-line distribution. Functions available will include report preparation, editing, and formatting.

EXISTING TECHNOLOGY

Hardware

The CONNECT service operates within existing network and time-sharing technology. Its components are (1) the ARPANET, (2) the TENEX time-sharing operating system, (3) the Xerox Graphics Printer (XGP), and (4) high-bandwidth/ soft-copy terminals. Some of the components of CONNECT are unique, while others have functional equivalents throughout the industry. Most were chosen because of their accessibility to the project.

1) The ARPANET's unique concatenation of computer resources provides redundant communications paths for reliable

connections to remote points, and its modularity allows the total system to expand or contract incrementally as required.

2) Each host computer on the ARPANET that will be providing a message service will be a Digital Equipment Corporation PDP-10 using the TENEX operating system. CONNECT will shield the user from the varied system load levels to produce uniform response time. In a later test environment the user will communicate directly with CONNECT, omitting the dialogue with the TENEX Executive.

3) The XGP, a high-quality raster printer connected to a support computer, provides the necessary hard-copy facility to disseminate messages and reports outside the service. XGP can be used as a typewriter replacement to maintain machine-readable copies of all outgoing correspondence.

4) Of the many suitable terminals, two particular terminals are being considered by the IA project staff: the Institute Terminal System (ITS) and the TTY40. ITS is a double-density TV-based system being built for ISI by Systems Concepts Inc. Terminals display 50 lines of 80 characters each; they have variable control of character fonts and Xerox Graphics Printer compatibility. The TTY40 is a new soft-copy/hard-copy terminal to be offered by Teletype Corporation, purported to be of particular interest where security is paramount. A large number of medium-bandwidth soft-copy terminals are available on the market in the $2,000 to $5,000 range. The above two are of particular interest in that the first is already available and highly flexible, while the second is expected to become available for classified applications.

User Interface Techniques

It is an important part of the CONNECT philosophy to treat each user as an individual. Among the techniques previously used to smooth the user interface, CONNECT emphasizes the following: feedback responses, homogeneity, help features, error handlers, and individualized interfaces.

Feedback responses. In its simplest form, the response merely tells the user that CONNECT is operational and understands his request. However, many of the feedback responses also prompt the user for input. These short messages (often

one character) change as the user moves from state to state; they might differ when the user is typing commands, as opposed to entering text. Other kinds of feedback responses are messages that describe the service's last action; some are more elaborate prompts that give more detailed instructions describing the expected input. Other responses are associated with abbreviations or short forms of commands. These include ways for the user to request the expansion of an abbreviation in order to confirm the service's recognition of his intentions.

Homogeneity. To insure natural use, actions common to services are carried out in a consistent way. If the user knows how to do something in one context (supply a date, specify someone's name, refer to a file, etc.) he can do it the same way in any other context, which increases user confidence and reduces errors and learning time. This is not to say that parameters that are not meaningful in some context are required just for consistency, nor that the usual construction of a request must specify enough information to resolve the most ambiguous context. In the former case simplification is accepted; in the latter case ambiguous requests result in further interaction to clarify the user's intent. Homogeneity does not interfere with natural operation. Recognition of abbreviations, for example, may be highly context-dependent (e.g., recognition of file names is limited to the user's current set of files). To enforce consistency would require prohibiting such recognition. Rather, CONNECT provides consistent ways of asking for further specification when necessary.

Help features. Help features aid the user in determining his choice of actions at any given time and the consequences of these actions. They are of two varieties: one initiated by the user, the other by the service. User-initiated help features include requests such as the local time at a message destination, the status of an operation, or a tutorial on some service. CONNECT-generated help features are triggered by user actions. For example, if the user attempts to use a command he has never used before, the service may recommend a short tutorial on the use of the command. If the user is making repetitive errors, assistance is offered, perhaps in the form of an explanation of the service's current expectation from the user or a more detailed tutorial dialogue with him. Also, if the user is doing something awkwardly, CONNECT can teach him

a new command to alleviate the difficulty. In this case the motivation for the new command will be clear to the user. For example, CONNECT may suggest a composite command to replace a frequently repeated command sequence.

Error handlers. The proper response to errors is of major importance in a well-designed user interface to an on-line service. CONNECT is concerned with detection, prevention, correction, and notification of any detected errors. When an error is detected, the cause is explained to the user in detail, and he may then be invited to use the tutorial. Terse or coded messages are avoided. When an error is detected, the offending command is aborted. When the user discovers a semantic error, he has convenient ways to remedy its effect by specific commands that undo the effects of previous commands. In concert with error correction, he has powerful intracommand editing features.

Individualized Interfaces. CONNECT permits users to select personal designations for commands and even create macro commands for themselves to simplify or expedite their work. On the basis of a profile maintained for each user, CONNECT adjusts its internal operating parameters to fit his individual style, tailoring its prompting and feedback responses to reflect his familiarity with the service.

RESEARCH DIRECTIONS

In addition to technology already available for implementing CONNECT, some innovative methods are needed to deal with the problems facing the nontechnical user. These methods fall into four categories:

- Adaptive processes
- Program structures
- Integrated tutorials
- Response-time adjustments

Adaptive Processes

A common fault of many man-machine systems is that, although they provide the necessary functions, they fail to fit any particular application very well. This problem may be due to a rigid command structure that requires using overly complex commands to perform often-used functions. CONNECT, on the other hand, is much more adaptable to each of its potential applications. The wide range of message-like functions it supports were

determined by studying potential (semi-automated and manual) user environments. The user interface is incompletely specified except with respect to a particular group of individuals. The command language and service idiosyncrasies are adjusted "on site" to suit the individual user environment. The "on-site" adjustments are aided by models of users and services that are used as predictors in choosing.

In addition to this site-dependent preadjustment, the service/user interactions and intraservice dependencies are instrumented. Data samples, collected through real-time measurements, are analyzed on the spot, and immediate adjustments are recommended to the user. The purpose of such dynamic evaluation is to further refine the user's performance. While techniques for service self-regulation are perhaps the least well known and understood of the various techniques discussed below, the potential increase in efficiency to be gained by the use of instrumentation and adaptation is thought to be great enough to warrant including this aspect in the project study.

Program Structure

Most large programming systems are structured to minimize the interactions between modules, which provides clean intermodule interfaces (in many cases null interfaces). This approach also tends to produce vertical partitioning, in which each module decides and provides for itself in all situations, resulting in an uneven, heterogeneous system. CONNECT is partitioned horizontally, with each module responsible for a single service-wide function; all other modules needing this function use the single module designated for that purpose.

Horizontal structure prevents several classical human-related problems. First, no application module interacts directly with the user. All input or output activities are handled by a collection of interaction modules, which in turn insures that each implementer will give more thought to the user interface standards. Second, each module (several may be active simultaneously) maintains state information in a data table, which allows CONNECT to know its total state, a requisite for service monitoring of user interaction. Finally, all modules must indicate on return how to reverse their

effects, which gives the user methods to abort or undo previous actions.

Integrated Tutorials

Today many examples can be found of either computer-aided instruction (CAI) or computer systems with help features. CONNECT integrates the two into a tutorial service with novel capabilities. When the user requests help, he is automatically connected to the tutorial module. Aside from conducting fixed dialogues with the user, the tutorials are able to demonstrate commands to the user, and can also ask the user to try things and observe any problems. In fact, a novice user might spend his first sessions totally under tutor observation.

In addition to the above method of operation, the tutorial service aids the user in personalizing language constructs to reduce or eliminate those forms that lead to inefficient and ineffective performance.

Response-time Adjustments

Most interactive application systems attempt to minimize response time. However, we believe that response-time characteristics should be a stated, realizable goal, not an instantaneous, unrealizable ideal only erratically approached. CONNECT does not try to minimize response time; instead, it adjusts response time in concert with the user's psychological expectations. Considerations include the following:

- Providing constant response for a given action, so that the user knows what to expect.
- Making small response-time adjustments to decrease the user's error rate.
- Allowing the user to do other work while waiting, if a request is not an interactive response-time task.

EVOLVING DESIGN

The design of the CONNECT system has evolved to meet the needs and problems of prospective users. The design now consists of two parts: the core system and a set of application modules. While the application modules have not been specified in much detail to this point, the core system has been designed to a fair level of detail.

This core system consists of five parts: 1) an Executive that supervises the execution of the system and provides the

interface between the operating system and the rest of the service (i.e., the rest of the core and the application modules); 2) a Command Language Processor that parses and examines all input to the service and maintains a consistent interface between the user and the application modules; 3) an Editor responsible for all text manipulations the user may require; 4) an instrumentation and adaptation package, called the User Monitor, that monitors the user's interactions and suggests new dialogue elements to personalize the interface for each user; and finally, 5) a Tutor that teaches the user and aids him in regulating the dialogue forms for maximum user's performance.

The Executive

The CONNECT Executive (Exec), functioning as the interface between the operating system and the remainder of the CONNECT system, serves the following purposes:

1) It buffers the terminal user from interaction with the primary operating system and channels all error interaction and housekeeping communication through the appropriate CONNECT module.

2) It acts as a common channel through which all modules request service from the operating system, in order to buffer the rest of the system from changes to or replacement of the primary operating system.

3) It provides the primary operating system two supplementary services: fulfillment of requirements not satisfied by TENEX and reformulation of TENEX service in a form suited to the CONNECT system or to particular CONNECT functions.

Executive services provided include error control and system request routing. The Exec supplements TENEX services both by providing services not available through TENEX and by reformulating certain TENEX services for the convenience of the programmer.

The Command Language Processor

The Command Language Processor (CLP), which processes all user inputs, is the complete logical input interface between the user and the rest of the system (and may be the complete output interface as well). The CLP must satisfy four sometimes conflicting requirements:

1) Because the CLP provides the language and mechanism for service modules to communicate with users, it must be general and flexible. It must provide a command language definition capability powerful enough for the interactive service modules not as yet specified; at the same time, however, it must be simple and convenient enough to use so that the service module authors are willing and able to use it.

2) It must, at the same time, establish an interface that can be understood by computer-naive users with minimal training. Its commands must be simple and consistent. While providing a language for writing commands for service module authors, the CLP must simultaneously represent the needs and interests of the intended users. Experience with other computer systems has shown that these two requirements are often in direct conflict.

3) CLP must provide alternate ways for users to express their needs. Alternate forms are used when the User Monitor detects inefficient or ineffective dialogue and, via the Tutor, suggests an improved form for a particular user. If the user elects to employ a new dialogue element, the CLP must be able to parse and understand it.

4) Finally, CLP must perform all its functions in an especially transparent manner. It is expected to make available to the Tutorial and Help subsystems information about existing commands, the user's knowledge and use of them, and a recent history preceding an error.

The CLP may be seen as two discrete (horizontally structured) pieces: (1) the Parser, String Processor, or Compiler and (2) the Interpreter, Virtual Machine, or Executor. In effect the CLP is both a compiler (for the command language as entered by the user) and the virtual machine whose "machine language" is the target language into which the compiler translates the user language. Within this viewpoint the remainder of the system provides the functions that help define the virtual machine. Henceforth we refer to the first part of the CLP as the Command Language Compiler (CLC) and the second part as the Command Language Executor (CLE).

Command Language Compiler. The task of CLC is to take the corrected input store and produce a program executable by the CLE. CLC has as two of its goals to provide a consistent, system-wide flavor to the command language. In order to

achieve these goals, CLC takes a pseudo-natural-language approach to commands, with the command language defined in terms of a very simple English-like syntax. This is not to say that CLC attempts to interpret anything like general English input. Rather the CLC command syntax uses the same basic notations and categories (noun, verb, adjective, etc.) that native English speakers comprehend. All rules provided by each application module are defined in these terms: the application module author is required to categorize the words he uses to fit this framework. CLC uses this framework to provide horizontal consistency throughout the command language.

Command Language Executor. CLE's task is to perform the services requested by the user, which it accomplishes by making calls on other modules of the overall system.

The Editor

The design of the CONNECT Editor has proceeded from three basic maxims:

1) Whenever the user is typing, he is entering text and should have available to him as much of the Editor as is appropriate.

2) There should be as little difference as possible between the text as the user sees it while editing and the text that would appear in hard copy at that moment.

3) The Editor should provide all the basic capabilities of paper, pencil, scissors, paste, and typewriter while being as natural to use as possible.

A careful examination of the range of possible user environments also produces some boundaries defining what the CONNECT Editor should not attempt to do. In particular, since the Editor is not envisioned as a "program" editor, it will not seek to provide specific program-structure-oriented commands, nor will it support the writing of sophisticated editing programs such as might be required if its users were primarily programmers. It will also restrict itself to nongraphic editing, in deference to the limits of commonly available and inexpensive CRT terminals.

The design of the Editor may be viewed at three logical levels: the correction function, low-level editing, and the full editor.

The Corrector provides intraline editing of the user's input, whether commands or text. This must be carefully human-engineered to be as natural as possible (though this obviously varies depending on terminals). This level of the Editor requires absolutely minimal training and has the flavor of actually "making corrections" rather than "issuing commands" to an editor. As far as possible (dependent on the terminal's local capabilities and bandwidth considerations) the results of these corrections should be reflected back to the user.

Low-level editing is accomplished by simply generalizing the Corrector's intraline capabilities to extend to text in general. With greater "moving" capability, the user can perform any editing task without needing "linguistic" commands. If the Corrector interface is natural, this low-level editor requires almost no additional training. It does not, of course, provide the power of searches, block transfers, or replacing, but it does allow a user to start using the system to edit text almost as soon as he can log on. This level also includes text reading facilities.

Full editor capabilities include search, replace, restructuring, block manipulation, interfile, multi-author, and annotation. They attempt, however, to minimize the number of distinct commands without going to the opposite extreme of complex parameterization. They also attempt to optimize the tradeoffs between terseness and intelligibility, and between feedback and bandwidth requirements.

User Monitor

The main purpose of today's hardware/software measurement tools is to aid in the design or selection of new equipment or the reconfiguration of present equipment. One criterion or a combination of criteria is established as a metric (such as processing a specified job to maximize throughput or minimize cost). Our interest, however, in an instrumentation and adaptation package is to increase the user's performance (as oposed to the system's performance) as he uses a service of a time-sharing system. Thus, the User Monitor instruments, evaluates, and predicts user/system behavior patterns in order to regulate service activities and the user's practices to optimize the user's

performance. The following illustrate the nature of improvements to be made in real time.

1) Detect when the user is doing things "wrong" by means of elicited errors or repetitive operations. Here, we are not looking for errors that prohibit useful work, but rather those that contribute to poor performance. Corrective action involves interaction with the user.

2) Detect when the user has mastered elementary sequences of operations and then advance tutoring to the next level of language constructs useful to his work.

3) Model the system and user to determine when a different sequence of events can more effectively accomplish the same task.

To improve performance, adaptive features must be able to affect system operation in real time according to the fluctuations in system and user actions. Hence, measurement and regulatory aids must be designed as an integral part of the system in order to assist resource scheduling and to alter the user's practices. Given compumetrics as a fundamental component of the system, it should function as unobtrusively as possible. Much of the data related to performance cannot be accurately estimated by the user, since large inefficiencies can occur in small time frames that cannot be detected. Hence, measurements are integral and are taken in real time.

To succeed in real time, the User Monitor must employ a data extraction technique that does not significantly alter the system's operation. Hence, pseudo-random sampling rather than event-oriented monitoring is used primarily. Synchronization is avoided to insure that the data obtained is a function only of the number of samples, not the sampling frequency.

Operating within these prerequisites, the User Monitor has two very different functions. The first is connected with the experimental goals of the project. As experiments are run, the User Monitor makes measurements in real time and in background analyzes the acquired data. Inferences based on the results are made by the designers. Thus a user's performance is correlated with an off-line model of the user and the service. Given models of the user and service, they may be used in the future as predictors for choosing language forms yielding best

performance, best teaching methods, and so forth.

The second function is to aid the user (via the Tutor) in improving and personalizing elements of the language. Three cases arise. The first case is to recognize dialogue sequences which occur with significant regularity. Once identified, the Tutor will suggest a composite form to the user. The second case arises where the CLP is unable to recognize an input. The Monitor attempts to identify the spurious command and suggest (via the Tutor) some remedial action. The third case involves the isolation of those language constructs that do not prohibit useful work but do lead to poor performance. Again, alternate forms are recommended through the Tutor.

The Tutor

One of the prime aims of CONNECT is to be easy to learn and forgiving to use. A crucial aspect of both these goals is the provision of fully integrated help and teaching facilities.

A number of systems make use of a "?" or "Help" feature to allow the user to ask for assistance whenever he is confused. Typically, these use a minimal amount of the user's recent actions to make a guess at what the user wants to know, then provide him with a short description or list of options open to him at this point. A slightly more sophisticated approach is to give the user a set of choices and converse with him to find out what he wants to know, although this takes more time.

We envision this sort of capability (which ultimately provides interactive access to an on-line User's Manual) as the first step toward a helpful system. This "Help" facility must provide terse reminders for experienced users who have momentarily forgotten something, verbose explanations (e.g., selections from a User's Manual) for naive users, as well as an interface with the actual Tutorial segment described below. It must allow the user to interact to select what he wants to see, while recognizing enough of his state to make reasonably accurate first guesses to minimize such interactions. It must also allow easy ways for the user to ask for help at higher or lower levels: for example, a request for help when typing a command might yield a description of that command, whereas the user really wanted either

alternate commands or details of particular parameters to that command.

The Tutor provides help on request, introduces users to new aspects of the service, interprets service errors for the user, and provides User Manuals at various levels tailored to a user's knowledge of the service.

Additional issues applying both to Help and Tutorial concern whether the system should take the initiative in giving the use help when it "thinks" he needs it (e.g., when he repeats errors in using some command) or whether it should always wait until he asks for help.

Though the value of human classroom teaching and personal tutoring should not be underestimated, sooner or later a user is required to do something with the system for which he has not been trained. The only instructional aid that we can guarantee to be available whenever the user is on the system is the system itself. Therefore we consider it important to make the system as self-teaching as possible.

The majority of existing CAI systems either are oriented toward teaching some specific subject or provide languages in which instructional programs (or "lessons") can be written. The former approach is inappropriate for an open-ended system like CONNECT that must provide for arbitrary services. The latter is a feasible approach; however, the designer of a service module should not be required (or, more to the point, expected) to write CAI lessons, since this is tangential to his major goal. It may be possible to make the CONNECT Tutor table-driven by a description of the particular service, though this requires further research.

The Tutor is designed both as an extension of the Help system (so that if the User's Manual is insufficient to resolve a user's confusion, he can ask for a lesson on the subject in question) and as an introduction to the system as a whole or to a given module or command.

Essential to our concept of the Tutor is a recognition of the need for the user to get his hands on the system and try things. Ideally, the Tutor should provide the equivalent of an interactive classroom (or tutored) lesson, and then allow the user to experiment with the system as if a human tutor were looking over his shoulder.

Of the two apparent ways to implement this, the first--having the Tutor simulate the commands that the user tries--has been discarded in the belief that it can only lead to an ultimate disparity between the system's behavior as simulated by the Tutor and the actual system's behavior once the user leaves the Tutorial. In order to avoid the inevitable loss of user confidence that would result, we are seeking to provide the Tutor with a "monitor" mode in which it can experimentally pass the user off to some service, allow him to execute commands "safely," and insure that the Tutor can always maintain control. In a sense this allows the Tutor to simulate the system by using the system itself.

This approach greatly expands the possibilities for the Tutor: It can lead the user through trials and examples in which he is actually using the system. This capability may be able to greatly reduce the gap between classroom instruction and actual hands-on use, and will make the system behave far more helpfully.

In summary, the Tutor seeks to make the system intelligible to the user by (1) providing an on- or off-line User Manual for that part of the service which the user is interested in, (2) answering questions about the service, (3) teaching the user to do new things with the service, and (4) translating all service error messages into terms the user can understand.

PROJECT STATUS

The project, which consists of three phases, is currently in the latter stages of the first phase and entering the second. Each phase is scheduled to require about six months for completion.

The first phase is the design of a model system that will reflect the state of the art with respect to human-factors technology and the methodology described in this section, as well as incorporate the two goals for the resulting system: first, that it be a demonstration service to exhibit and evaluate that technology; second, that it be an experimentation vehicle to develop a new methodology.

Phase two is implementation. During the second phase the project will implement the model system on the PDP-10 under the TENEX operating system, using the BLISS-10 programming language. Fortran will also be used for much of the

statistical analysis done by the User Monitor. Because of a mild interest in transferability, all operating system calls will be restricted to one module.

Phase three is the experiment and evaluation phase. The currently planned experiments deal with different language forms which can be provided to the user. These include functional notations, computer-directed dialogue, multiple-choice commands, function keys, sub-commands (key word macros), etc. The goal, as described under the User Monitor, will be to match various user characteristics to the different language forms. The evaluation phase will address both the suitability of the model system as a production paradigm to be used in directing grand or production targets and its applicability as a human-factors testing facility. The expected future efforts should include both the more production-oriented goal of delivering complete command and control message services and the pursuit of research into user's needs.

SUMMARY

The main goal of the Information Automation project is to extend the benefits of computer technology and methodology to users of written information, delivering these benefits in a way that users find convenient and helpful. The primary emphasis of the proposed system design is people efficiency rather than machine efficiency. It is the conviction of the project personnel that general-purpose but nontunable services do not solve these types of problems. Hence, CONNECT makes it possible to efficiently generate specialized simple services for communications applications by means of continuous measurement and adaptation.

Though some of the above ideas are not new, they have not yet been successfully combined into an integrated service for use by people unfamiliar with computers. In fact, the attempt to design a human-engineered service is a relatively recent one. Attempts in this direction are prone to many pitfalls, particularly the failure to understand in sufficient depth and breadth the true needs of the prospective user population. CONNECT will attempt to avoid these pitfalls by careful, ongoing attention to the needs and requirements of specific user groups. A prototype CONNECT service should be available on the ARPANET early in 1975.

REFERENCES

1 Roberts, L. G., and B. D. Wessler, "Computer Network Development to Achieve Resource Sharing," AFIPS Conference Proceedings, 1970 Spring Joint Computer Conference, Vol. 36, AFIPS Press, Montvale, N. J., 1970, pp. 543-550.

2 Bobrow, D. G., J. D. Burchfiel, D. L. Murphy, and R. S. Tomlinson, "TENEX, a Paged Time Sharing System for the PDP-10," Communications of the ACM, Vol. 15, No. 3, March 1972, pp. 135-143.

NETWORK SECURE SPEECH

PROJECT LEADER: Dan Cohen

RESEARCH STAFF: Raymond L. Bates
E. Randolph Cole
Thomas N. Hibbard
Robert H. Parker
Paul K. Raveling
Guner Robinson

RESEARCH STAFF SUPPORT: Norma B. Johnston

STUDENT AIDE: Steven L. Casner
Jimmy T. Koda

INTRODUCTION

The development of a digital means for secure speech transmission is a very high priority military goal. The ISI network secure speech project is attempting to establish secure, high-quality, real-time, full-duplex voice communication using the ARPANET as a transmission channel. The resulting voice communications system will have the following advantages:

1) It can be secured (encrypted) to any desired level of complexity and security, since infinitely many coding schemes exist that can be readily applied to a digital signal.

2) If desired, extremely high quality and low error rates can be achieved (although possibly at the expense of substantially higher data rates).

3) Packet-switched communication can exploit the natural pauses and breaks in human speech in order to achieve a lower overall data rate. Because packet-switching methods used in the ARPANET are asynchronous, however, the received speech will be delayed slightly, since it must be assembled synchronously.

4) Such a voice communications method will be highly compatible with future communications systems (such as satellites, lasers, packet radio, etc.), all of which will be digital and many of which will be packet-switched.

PACKET-SWITCHING NETWORKS

Packet switching offers more efficient use of communication channels than circuit switching. It is suited for voice communication because of its intrinsic nature, i.e., of achieving any given rate by using short bursts of higher rate. This technique by its very nature takes advantage of the variable data rate in human speech (long vowels, silence periods, etc).

The philosophy behind packet switching is that the highest reliability can be achieved by error detection and retransmission when necessary, which allows reliability to be as high as desired at the possible cost of delays and bandwidth. Since voice communication cannot tolerate arbitrary gaps and delays, the ARPANET's present definition of reliable transmission must be modified and transmission procedures changed accordingly.

Special protocols are developed for voice communication independent of the specific vocoding technique being used. In fact, a part of the protocol is to make sure that both parties use the same vocoding technique. The same communication protocol will be used for LPC (Linear Prediction Coding), CVSD (Continuous Variable Slope Delta modulation), and any other vocoding technique.

VOCODING

In order to achieve digital voice communications over the ARPANET, it will be necessary for the data rate required to be as low as possible in order to minimize the load on the network; some type of bandwidth compression (called vocoding when applied to speech) will thus be required. An extensive study of available vocoders, both hardware implementations and software algorithms, determined that present hardware vocoders are inflexible, expensive, and of insufficient quality. It was therefore decided to implement the best available software vocoder using a powerful general-purpose signal-processing machine. Following the evaluation of several signal processors, a Signal Processing Systems SPS-41 was purchased, along with a PDP-11/45 to drive it and handle communications with the ARPANET. Software support efforts for these two machines will be described in the final part of this section.

The software vocoding method chosen for implementation was LPC (Linear

Predictive Coding), specifically the version developed at the Speech Communication Research Laboratory (SCRL) in Santa Barbara [1]. Off-line simulations have shown that relatively high-quality speech can be transmitted at a data rate of around 3kb/sec using LPC. LPC requires much less computation than other known vocoding methods, with superior quality and the same data rate. It is important to note, however, that the vocoding method and the network communications algorithms will be independent of each other. The latter will be optimized to take full advantage of the properties of speech, but not the properties of the particular vocoding method itself. Should the speech research community develop a vocoding method that is clearly superior to LPC (not likely in the near future), the vocoding method could be changed without changing the network communications software.

The real-time LPC vocoder will be implemented in the following steps:

1) An off-line floating-point simulation on TENEX, undertaken to familiarize project personnel with the LPC algorithm and its implementation, is almost complete.

2) An off-line integer simulation will be implemented on TENEX. This must be done in order to insure that problems with the 16-bit integer structure of the SPS-41 will be minimized and to verify that results from the SPS-41 are correct.

3) The resulting algorithm will be implemented to run in real time on the SPS-41/PDP-11 system. Because of the highly complex programming structure of the SPS-41, this step will probably require much more time and effort than the previous two steps.

SPECIAL PROTOCOLS FOR VOICE COMMUNICATION

The communication protocol to be used for voice transmission has two components: control and data transfer.

The control component of the protocol is responsible for performing the connection to the right party and generating the equivalent of a "busy" tone or a "ring" tone at the originator and the equivalent of a telephone bell at the answering party. It is also responsible for making sure that the vocoding techniques used are compatible, and for negotiating some options.

Later, when conferencing is implemented, the control component will also be responsible for "Conference Control" (the chairmanship). It will direct voice output from the speaker to all the conference participants and direct control signals from the chairman (a person or a program) to all participants to inform them when a participant will become the speaker. Other signals will tell the chairman who wants to speak at any time.

The data transfer component is responsible for transferring the data from the speaker to the listener at the best combination of minimum delay and highest bandwidth. This will be accomplished by continuous monitoring of the network performance on-line and by adapting the transmission parameters to the variable network characteristics. The transmitting party will measure the delays, bandwidth, and actual number of parallel links available to the destination, and adapt itself accordingly. The receiving party will self-impose a voluntary delay in order to maintain better continuity (smoothing) of the output. This delay will vary according to the network performance as measured on-line.

ELF AND SPS SUPPORTING EFFORTS

The software environment for implementation of voice protocols will be provided by ELF, an operating system created by SCRL that is undergoing continued development. User processes running under ELF will control the vocoding algorithm on the SPS-41 and interface to the ARPANET using the special voice protocols. An additional user process will handle debugging facilities for the SPS-41.

Minor modifications to ELF's network control program will support those aspects of voice protocols that differ from standard ARPA host-host protocol and will supply facilities for measuring network timing characteristics during voice transmission.

Both ELF and SPUD (Signal Processing Unit Debugger) have been modified to support ISI's PDP-11/45 configuration, and each is operational independently. Work currently in progress includes conversion of SPUD to run as an ELF process, addition of a process to measure network timing characteristics as perceived by the PDP/11, implementation of a CVSD vocoder on the SPS-41, and final checkout of hardware built at ISI (PDP-11/IMP

interface and A/D and D/A converters for the SPS-41).

Virtually all program support other than debugging is handled by programs under TENEX. Support programs in use by the speech project include an assembler for SPS-41 programs, Bliss-11, MACN11, and many utility programs. MACN11's facilities for assembling PDP-11 programs have been substantially augmented by ISI and SCRL, and its development is continuing.

SUMMARY

Digital voice transmission can be made as secure and as reliable as desired. Because of its inherent asynchronous nature, packet-switching technology is naturally suited to voice transmission. Therefore, developing the means to use a packet-switching network for digital voice transmission is an effort of great potential usefulness to the military community.

Our work in this area is performed in close collaboration with the ARPA-NSC (Network Secure Communication) group, particularly with the Speech Communication Research Laboratory of Santa Barbara, mentioned above.

Our major role in this effort is establishing the methodology of using the ARPANET for this project and implementing it with real-time vocoding to prove the feasibility of using packet-switched digital networks for secure voice communications. Presently, our efforts are aimed at (1) performing network measurements for the development of the protocols required for real-time voice communications and (2) implementing LPC, both on-line and off-line. The next stage of effort will be to optimize the communication protocols and improve the quality and reduce the bandwidth of the vocoding technique.

REFERENCE

1 Markel, J. D., and A. H. Gray Jr., "Documentation for SCRL Linear Prediction Analysis/Synthesis Programs", Speech Comunications Research Laboratory, Inc., November, 1973.

TECHNOLOGY SUPPORT

8

RESEARCH STAFF: Thomas O. Ellis
Louis Gallenson
Robert E. Hoffman
Robert H. Parker
John J. Vittal

RESEARCH SUPPORT STAFF: George W. Dietrich
Orallo E. Garza
Lloyd G. Jensen

STUDENT AIDE: Ronald L. Currier

INTRODUCTION

This section describes three of the major advanced hardware systems being developed at ISI. Each of these hardware efforts was undertaken in response to research support requirements and/or to demonstrate a capability for a recognized DoD application. As mentioned earlier, they are also being developed as necessary support items for the several software projects that compose the Network Communication Technology effort. The projects include the Xerox Graphics Printer (a high-quality document printing capability in the form of a network terminal), the Video Display System, and Portable Terminals.

XEROX GRAPHICS PRINTER

Two Xerox Graphics Printers (XGP's) are now in operation at ISI as terminal devices. Each XGP is attached through a PDP-11/40 to the PDP-10 via a 2400-baud data link. The PDP-11/40, with a modified version of the software designed at Carnegie-Mellon University, will drive the XGP and provide hard-copy output from files on the PDP-10. The PDP-11 acts as a data buffer and line "rasterizer," providing video data and synchronization to drive the XGP through an ISI-designed interface. The interface design, based on a design originally conceived for ARPA at the University of Utah, has been repackaged and improved, resulting in a factor-of-four reduction in package size and an increase in reliability and performance.

One of the XGP systems is being shipped to the ARPA office in Washington, D.C. and attached to a Terminal Interface Processor (TIP) there to provide high-quality on-line hard copy. The

system remaining at ISI will be used experimentally to develop a two-speed version of the copier capable of providing (1) a high-quality output at 300 lines per inch resolution and 0.5 inch/second paper speed and (2) a lower-quality output at 100 lines per inch resolution and approximately 2 inch/second paper speed. Presently, the system runs at 196 lines/inch resolution and 0.67 inch/second paper speed. When the experimental system is completed, hardware and software will be retrofitted to the system in the ARPA office; if the system modification provides a useful function, it will probably be incorporated into future XGP systems supplied by the Xerox Corporation. The schedule for completion of two-speed modification to the ISI machine is late fall of 1974.

VIDEO DISPLAY SYSTEM

The development of the Video Display System is continuing as previously reported [1]. The system design provides an inexpensive high-quality terminal for computer users (i.e., programmers, managers, and secretaries) who require significantly more capabilities than are currently available with low-cost terminals. These include a full page of text (more than 4000 characters), graphics, 256 characters of writable font, and a standard communications interface to facilitate computer connection. A modular design (part of the stated requirements) will make it possible to use components in a clustered environment as well as in a remote stand-alone unit, with a minimum cost differential. A contract for the system has been let to System Concepts, Inc. of Palo Alto. Internal problems of that firm have delayed completion of the system by one year--first deliveries are now expected in the first quarter of 1976. Alternative suppliers are being examined, and the question of how to provide the desired high-quality terminal system is being reconsidered.

As an interim measure, ISI has leased Beehive terminals, a video-based teletype replacement. These terminals are standard equipment in each full-time professional's and each secretary's office; in addition, several public terminals are available for graduate students, consultants, and other part-time personnel.

One major effort during the year was the design of the keyboard layout for the proposed terminal system. We required the keyboard to generate the 128 seven-bit

standard ASCII characters in an eight-bit field (to be compatible with the ARPANET). The American National Standards Institute has proposed two standards for ASCII keyboards (proposed standard X4A9/199B): one with "logical bit pairing" and the other with "typewriter pairing." But, in either case, the standard defines only the "inboard" area containing the alpha-numerics and punctuation symbols; the layout of the "outboard control area" (e.g., Return, Delete, Control) is left to the designer. Thus we had three topics on which further design was necessary: which of the two standard keyboards to choose, the design of the outboard control area, and the method by which the operator can produce the 128 other characters permitted by the eight-bit field generated by the keyboard.

Most computer terminals use the logical bit pairing principle, which specifically pairs characters on a key so that there is a single-bit difference between the internal ASCII code representations of the characters. This was an early design decision made to ease the construction of older electromechanical keyboards; current technology permits more flexibility, and so we have chosen the typewriter pairing. A number of our projects envision the use of the terminal in standard office environments as an individual's first personal contact with a computer. The typewriter pairing will ease the transition to the terminal and permit easy switching back and forth between typewriter and terminal.

Our design for the outboard control area is similar to that found on a number of ASCII terminals. Two exceptions are that the infrequently used Line Feed key was moved to the edge of the keyboard, and an Alpha Lock key (which locks only the alphabetic keys to upper case) was substituted for the normal Shift Lock key. Included in the outboard control area are Case II keys, a Control II key, and appropriate Lock keys. When activated, these keys shift the entire keyboard (with the exception of the formatting keys like Space and Return) into a "third" case, and permit the generation of the upper 128 ASCII codes (octal 200-377). Finally, the keyboard includes explicit Tab and Back Space keys, as well as a key labelled "Help," which will cause ISI-designed software to take tutorial action.

Another major effort during the year has been work on a standard protocol for controlling the proposed terminal. A

major start on an acceptable protocol has been made by the ARPANET Network Graphics Group, chaired by Jim Michener of Project MAC at the Massachusetts Institute of Technology. Institute efforts have been devoted to understanding this protocol, especially with a view to any hardware features it might suggest, and cooperating with Robert Sproull of Xerox Palo Alto Research Center and Charles Irby of the Augmentation Research Center of Stanford Research Institute in refining it. Irby especially is the major designer of the "positioned text" portion of the protocol, the part in which the Institute is most interested. The positioned text protocol permits the programmer to divide the display screen into "windows" and to manipulate text independently within each window. For example, in a text-editing application, four separate windows might be used for messages from the system, command entry and feedback, the file being edited, and a file of connections.

As mentioned, we have been especially interested in designing hardware features for our terminal system that would ease the implementation of the protocol. The most significant area in which hardware can help is with the windows, especially independent scrolling of side-by-side windows and automatic actions at window boundaries. Our most promising idea is to organize the display character buffer as a list, with pointers and lengths being separate from the characters. Independent window manipulations can then be accomplished by relatively quick manipulation of the pointers instead of time-consuming rewrites of major portions of the character buffer. Finally, we are working on including new character fonts in the protocol to take advantage of the hardware proposed for the ISI terminals.

PORTABLE TERMINALS

Since its inception last year, the goal of this project has been the development of a portable terminal device able to communicate from any telephone to any ARPANET site, permitting mobile users to easily take advantage of mail, message, and other ARPANET services. In July of 1973, a prototype portable terminal was delivered to ARPA and has been functioning since then free of additional maintenance. The unit weighs 20 pounds and is enclosed in a small briefcase (10" x 14" x 6") that can fit comfortably under an airplane seat (see Figure 8.1). In contrast, conventional portable display terminals

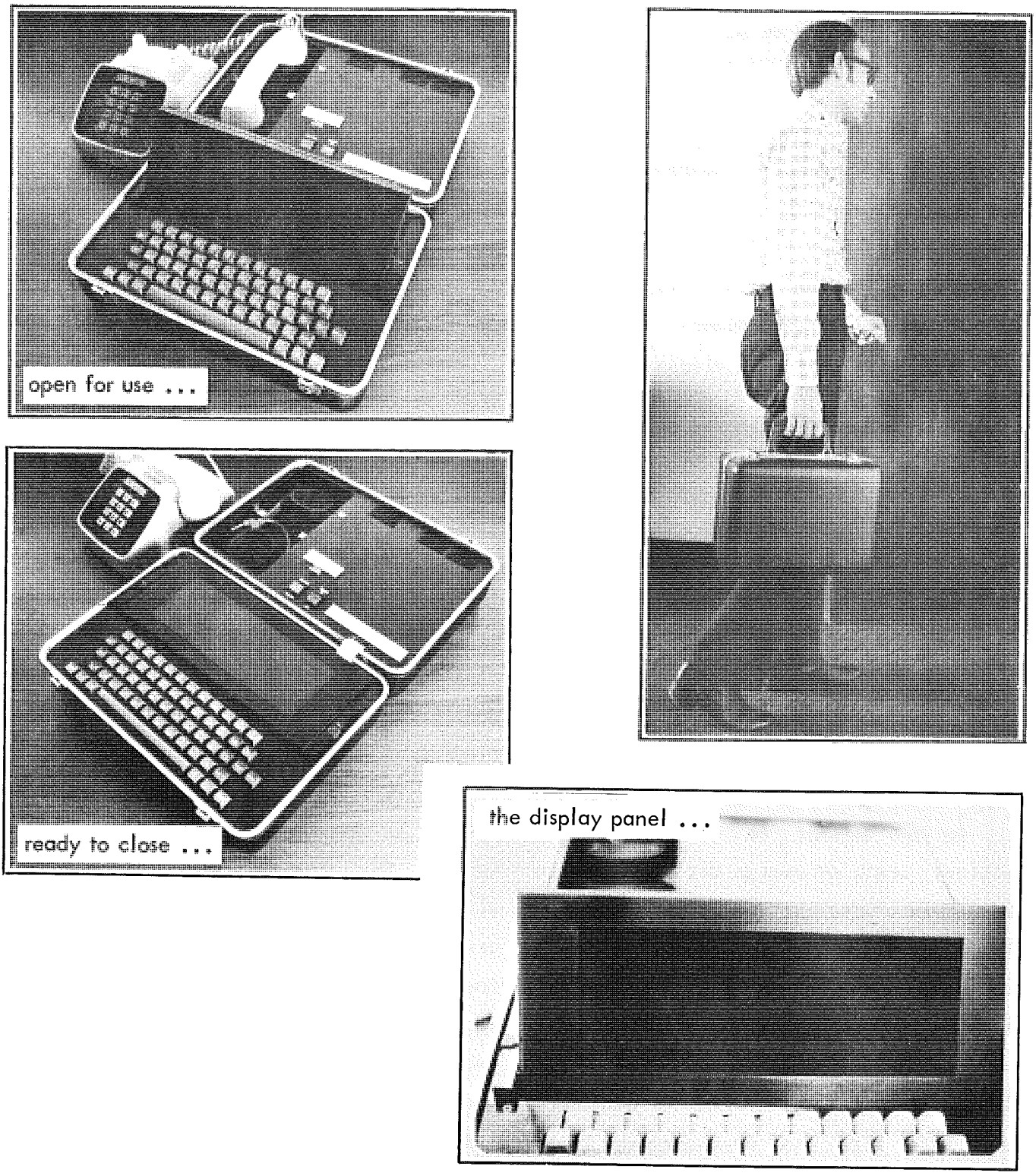

Figure 8.1 The ISI portable terminal

weigh nearly twice as much and are bulky and unwieldy to carry.

The ISI terminal is built entirely of standard off-the-shelf components. It uses a 53-key electronic keyboard, built-in acoustic coupler, and a Burroughs Corporation Self-Scan panel display unit with 8 lines of 32 alphanumeric upper-case characters (256-character display).

This terminal is intended primarily as an evaluation instrument to help in defining the functional capabilities required of second-generation portable terminals. The critical areas of the present design have been found to be its limited visual context and its size and weight parameters. The optimum tradeoff between minimum size and maximum display area is now being empirically defined.

In an attempt to improve these critical areas of design and also to gain insight into alternatives to the Burroughs display panel, two plasma panels have been purchased from Owens-Illinois Corporation. These are roughly comparable in size and weight to the Burroughs panel, but their 40-character-per-line format is much more compatible with the 80-character lines generated by TENEX (nearly doubling the line display capability of the terminal). The Owens-Illinois panels will be used in two further prototype portable terminals that will incorporate the following additional design refinements: upper- and lower-case characters, increased visual context, smaller power supply weight, smaller keyboard size and weight, and resulting smaller overall weight.

When these two prototype terminals are completed, the project will concentrate upon the most critical of the problem areas so far identified: the display screen. Liquid crystal displays, which could provide a lightweight, low-power, electromagnetic-radiation-free substitute for CRT terminals, look feasible for the portable terminal application. They seem to greatly reduce electrical power requirements, although with a proportional decrease in writing speed. An optimum tradeoff between power demands and writing speed requirements will need to be identified. The liquid crystal technology now existing in industry will probably be capable of providing a usable display panel within two years if this project can provide proper encouragement and direction by defining specific military requirements and perhaps providing financial assistance.

REFERENCE

1 Annual Technical Report, May 1972 - May 1973, USC/Information Sciences Institute, ISI/SR-73-1, 1973.

NETWORK MANAGEMENT INFORMATION CENTER

PROJECT LEADER: Stephen R. Kimbleton

PROJECT SUPPORT: Linda K. Tisnado

INTRODUCTION

The Network Management Information Center (NMIC) was established on March 4, 1974. Its objective is to provide a comprehensive base for network management, to develop policies and procedures for concurrent operation of Network Control Centers (NCCs), and to develop requirements and staffing characteristics for minimum-manpower, minimum-skill-level NCCs. To ensure the feasibility of the approaches developed, the initial objective will be to establish a backup NCC for the currently existing NCC at BBN which, on a scheduled basis, will be capable of assuming responsibility for detecting outages and performing appropriate notifications to accomplish their repair.

Accomplishing this initial objective will require a significant level of effort. The potential reward to be achieved is large when measured in terms of reliability, efficient resource use, and ability to accommodate rapidly changing workloads such as occur in stress situations for command and control installations. A framework for developing the total requirements of NCC(s) is needed. The following remarks provide one approach to the development of such a framework. It is to be anticipated that this approach will be critically examined during the implementation of the initial phase discussed above.

The ARPANET interconnects equipment provided by most of the major mainframe vendors, permits effective sharing of these resources by a widely varying user population, and serves as a research vehicle for directing and evaluating potential network technologies such as packet-switched radio and satellite communication. As a result of strong user demands for the latest advances in computing technology, high reliability, and ease of interface, management of the ARPANET is an exceedingly difficult task.

The difficulty of ARPANET management is increased by the nearly exponential growth in network traffic [1], the

increasing demand within the DoD community for access to the network, and--because of budgetary limitations--the requirement that ARPA-supported hosts operate near or above the utilization levels at which effective service can be rendered. Computing economies achievable through resource sharing, coupled with the opportunities resource sharing affords to provide services previously unachievable, e.g., the National Software Works [2,3], render it unlikely that this trend in usage will reverse. Thus, the requirement for efficient network management is of increasing importance. The need for this capability is increased by its relevance to all organizations possessing large collections of computer systems. Indeed, within the DoD several multimillion-dollar examples of such procurements exist, including the World Wide Military Command and Control System (WWMCCS), the Air Force Advanced Logistics System (ALS), and the Air Force Base Logistics System.

Effective network management requires rapid determination of existing chokepoints, timely identification of new configurations designed to eliminate these chokepoints, and the ability to determine the performance impact of proposed configurations upon both users and hosts. Estimation of the performance impact of new technologies must be available to provide for their smooth integration into an ongoing network. Determination of the performance impact of security requirements is necessary to permit effective utilization of network technology by other DoD or domestic agencies. Both existing and proposed configurations must provide sufficient reliability to encourage acceptance and continued usage.

NMIC has been established to provide both tools for the effective management of the ARPANET and a vehicle for research in the management of computer communication networks. The objective of this center is to afford a means for effective (management) decisionmaking among technological issues in network management. The resulting tools will also assist network management in evaluating organizational issues by determining their technological implications.

These capabilities will be achieved through a coordinated approach involving the establishment of an on-line data base containing network information, development of a management-oriented information display permitting rapid pinpointing of network chokepoints and easy fixing of their determinants and implications, and

implementation of programs designed to permit rapid examination of the performance impact of alternative network configurations.

The existence of such capabilities will be of significant use in determining the kinds, types, and locations of network traffic--a subject on which little is currently known--and will provide advance planning information to meet evolving network needs. In addition, these capabilities will provide a factual basis for resolving potential conflicts among users, hosts, and budgetary constraints through estimation of the total impact of demands or changes and provision of information to pinpoint reliability problems and other difficulties impeding effective usage.

In addition to the direct role they will play in network management, these tools will also be of significant interest in several network-related issues, e.g., configuration control, sizing and tuning of command and control systems, dynamic approaches to reliability, and peak workload processing through dynamic job migration. Further, the organized approach provided for displaying and manipulating of network performance information, as well as for determining the systems impact of network modifications, promises to be of assistance to other network-based research projects.

The remainder of this section is devoted to a fuller discussion of the ultimate objectives and approach of NMIC. To place this discussion in perspective and to demonstrate that the present confused state of computer system performance analysis does not preclude an effective capability for network management, we begin by reviewing some of the major factors responsible for the current somewhat unsatisfactory state of computer systems performance analysis.

PROBLEMS IN COMPUTER SYSTEMS PERFORMANCE ANALYSIS

Computer systems performance has been the object of careful analysis by a significant number of individuals for approximately the last five years. Nevertheless, the current state of the art is generally regarded as unsatisfactory. In large part, this appears to be due to three factors: (1) the ad hoc nature of much of the effort, (2) the failure to distinguish between organizational and technological issues in performance, and (3) a lack of coordination between information required for decisionmaking, data

required to generate this information, and tools used to aid in decisionmaking.

The ad hoc nature of prevailing performance efforts was strongly influenced by management expectations that (1) decreasing hardware costs would gradually eliminate the need for performance efforts and (2) a "good" system configuration could be found which, once obtained, would need relatively little adjustment. Time has shown, however, that decreasing hardware costs increase the complexity of the applications that can usefully be supported; thus the demand rate continues to outstrip available capacity. In addition, the computer system workload varies significantly with time because of changes in the organization, its product, and its objectives. Thus, performance must be viewed as a continuing rather than a sporadic activity.

Effective utilization of computer systems performance methodologies by management requires distinguishing between organizational and technological issues in performance [4]; some of the topics relevant to each category are indicated in Table 1. Organizational issues have two primary characteristics: the inclination of management to satisfice rather than optimize on these variables, and the dependency of reasonable values for these variables upon factors specific to a given site. Thus, although centralized guidance in the resolution of organizational issues can be provided, a centralized solution is precluded.

TABLE 1

ORGANIZATIONAL AND TECHNOLOGICAL ISSUES IN COMPUTER SYSTEMS PERFORMANCE

Organizational Issues	Technological Issues
Unnecessary Jobs	Extended Instruction set
Insufficiently trained programmers	More host memory
Priority structure requirements	Device/channel ratio
Acceptable reliability	Data set location
System overhead	Hardware-compatible upgrades
Programming language support	Load leveling
Data base capabilities	Performance impact of file backup

Technological issues, by contrast, are capable of centralized investigation and solution. Such an approach requires coordination of data gathering, the information to be derived from this data, the decisions to be made by management, and the means used to assess these decisions. The cost of not having a centralized approach is reflected in the fact that a typical simulation study of a computer system costs $50,000 and requires one man-year of effort over six man-months, while the cost of renting the simulator for six man-months would usually be less than $5,000. The difference reflects manpower costs to gather the data and the cost of execution of the simulator.

The distinction between organizational and technological issues is not completely invariant. As an example, if the number of users in a proposed message switching system [5] were to increase by a factor of five, both categories of issues would be affected. However, a coordinated approach to technological issues would significantly enhance the capability of management to project the implications of such an increase.

Effective decisions among technological issues in network management must be based on suitable information to permit (1) timely detection of usage trends and their determinants, (2) better assessment of the performance impact of modifications and enhancements designed to provide cost-effective service, and (3) factual resolution of the performance impact of requests for additional services. Thus, the gathering of data and the transformation of data into information and information display must be coordinated. One dimension of the cost implicit in not having a coordinated approach is reflected in the cost of present computer system simulation studies as discussed earlier.

The need for a coordinated information-based approach to performance has been well recognized [6]. The available information on both management requirements and performance technologies is sufficient to permit such an approach. It has been argued [4] that a major reason for the absence of such an approach for individual computer systems is the lack of leverage in comparing the cost of its implementation against the probable accrued benefits. Networks, however, provide an immense leverage potential. Further, their geographical dispersion, coupled with the magnitude of the resources involved, serve to increase this leverage. Let us discuss the requirements and implications of such an approach.

TECHNOLOGICAL ISSUES AND NETWORK MANAGEMENT

In the preceding subsection we identified the distinction between technological and organizational issues in performance; it is evident that a similar dichotomy exists for networks. A major objective of network management should be an effective capability to resolve technological issues and thereby make it possible to determine the technological implications of organizational issues.

Management of technological issues requires three functional capabilities: (1) a monitor function that helps to identify the relative "health" of the network, (2) a detection function that helps to identify factors responsible for poor performance, and (3) a correction function that helps to identify appropriate modifications to achieve satisfactory performance.

The first function requires identification of display variables of use to management in monitoring the quality of the network environment. In addition, it requires instrumentation of users to track their evolving requirements and instrumentation of the network to track its performance. To avoid inundating management with unnecessary information, a management by exception approach is required that provides information on a few key factors to determine if network performance is acceptable.

The second function requires a hierarchically organized data base on network performance that permits rapid identification of the determinants and implications of network chokepoints. Careful control of the amount and types of information presented is necessary to permit quick identification of major factors.

The third function requires a capability to predict network performance as a function of resource modifications that might be performed. Although a network is very complex, basic network research suggests that, in an unsaturated network, a structured consideration of hosts, TIPs/IMPs and transmission links may be used to determine or predict overall network performance in an effective manner [7,8]. Thus the requirement to predict network performance as an entity can be replaced by a requirement to predict the performance capabilities of major resources and interrelate the resulting predictions to determine network performance in an unsaturated network. For a saturated network, by contrast,

performance degrades so rapidly [9] that a major objective is to reduce load or increase capacity until it is below the point of saturation.

The preceding observation suggests that management of technological issues requires two basic capabilities: a report/display capability and an alternatives assessment capability. The first would permit quick inspection of existing or projected performance and determination of the nature of desired modifications to obtain additional improvement through careful hierarchical organization of performance information. The second would permit projection of network performance for a proposed configuration.

Determination of the precise information to be displayed, as well as the manner in which it is to be organized and stored, requires further investigation. However, it is apparent that at least four categories of information must be provided: (1) current performance, (2) present network resource availability, (3) projected resource availability (to coordinate preventive maintenance and other scheduled resource reductions), and (4) detailed performance information for evaluating the effects of proposed changes and determining desirable directions of future changes.

The dimensions of the network performance problem as reflected in the large number of potential alternatives available, the probable variation in the metrics used to compare alternatives, the constraints that must be observed in comparing alternatives (e.g., the connectivity of an IMP in a network topology) must be at least two, and the necessity to consider alternatives to satisfy organizational issues (requirements) argues against a straightforward (single-stage) approach to achieving networks with good performance. Instead, an interactive approach permitting rapid management investigation of alternative configurations is required.

To be effective, this approach must stress speed in achieving performance projections, perhaps at the cost of some slight amount of accuracy. Current research [10] suggests that very fast performance prediction for computer systems can be achieved through analytically driving simulators. Thus, a key element for rapid prediction of network performance is potentially at hand.

Rapid network performance prediction also requires an efficient means of

inputting data descriptive of host capabilities, host workloads, IMP workloads, and traffic among IMPs. This can be achieved through (1) automatic data gathering, (2) maintenance of performance information on-line, and (3) inputting of only change data in assessing alternative configurations. No major technical obstacles preclude such an approach, although organization of the performance data base will require careful consideration. Thus, these capabilities can easily be achieved within the research environment provided by the ARPANET. It should be noted that the use of artificial intelligence technology could permit more effective management interaction at a relatively low net cost.

Our discussion of network management has been primarily focused on the management of resources to meet evolving workload needs. However, the capabilities represented by NMIC also promise to be of substantial use in other significant management concerns directly driven by the existence of the network. These include (1) prediction of network capability to handle a new research project and determine the "home" host for this project, (2) determination of network impact of reliability problems and the nature of appropriate changes, (3) provision of basic performance and capability information to other research projects predicated on network existence (e.g., COTCO [5], National Software Works [2,3], and Network Secure Speech) and (4) investigation of the user impact of emergent network technology.

RELATED BENEFITS

Configuration control, i.e., the assurance of compatibility among programs, files, and hosts, is a topic of continuing concern to network management. This problem is of particular significance to organizations in which the network is centrally funded and is intended as a centralized means of investigation of particular problems (e.g., large-scale command and control systems). Existence of NMIC will provide a means for semi-automating configuration control through establishment of data bases describing salient properties of hosts, files, and program resource requirements.

The existence of these data bases will also be of interest in resolving two other problems of concern in command and control systems: (1) workload reallocation to provide for continuing effective processing in the presence of outages

originating during a stress situation, and (2) reallocation of required, but not essential, workload from sites suffering a processing overload to less heavily loaded sites during periods of peak processing. Thus, the first problem effectively requires a fail-soft capability in the face of equipment malfunction or destruction, while the second reflects the fact that proper functioning of an installation requires processing of both the stress-induced workload and other workload components during lengthy stress situations.

Network reliability is a major topic of concern to users faced with deadlines and increased manpower costs occurring because of an inability to access network resources. The number of distinct resources in a computer network both make it highly probable that one or more resources will be unavailable during a given time interval and simultaneously preclude a straightforward redundancy approach to reliability (because of budgetary limitations). However, the network and the global sharing of resources which it affords provide an opportunity to achieve reliability through dynamic migration of jobs and files. The requirements and costs for such a software approach to reliability constitute an interesting subject for future study.

Sizing individual hosts to handle peak loads is a problem of continuing concern. If sufficient resources are required to handle peak loads while providing satisfactory resources, significant average excess capacity may result; conversely, if sizing is performed to permit good service for the average workload, peak load processing may suffer.

Workload migration to permit effective peak load processing could be a viable and cost-effective solution within homogeneous (sub)networks. In a crude sense, remote job entry capabilities provide one form of such migrational capabilities. The extent to which these capabilities can be realized dynamically is a significant research topic whose resolution requires information concerning network capabilities, user loads, and resource requirements. Much of this information is potentially available in the data base maintained by NMIC.

The diversity of funding sources and organizations represented on the ARPANET limit widespread application of a centralized approach to sizing and selecting network resources. However, organizations possessing large collections of computer systems, e.g., command and control systems or logistics systems, are often con-

strained by a fixed allocation of funds for procurement of the entire system. Thus, dynamic workload migration promises significantly increased system capabilities within fixed budgetary constraints.

SUMMARY

A trial-and-error approach to network performance promises to be disastrous. Network management requires consideration of both technological and organizational issues, and the dimensions of a network demand the development of tools to facilitate management assessment of these issues. Such tools can be achieved through a coordinated approach involving data gathering, transformation of data into information, information display, and alternatives assessment. Subsaturation behavior of a network permits a hierarchical approach to performance projection which, in turn, permits development of appropriate managerial tools. In addition to assisting in network management, these tools will also be of significant use to other projects requiring networking capabilities.

REFERENCES

1 Kleinrock, L., "On Measured Behavior of the ARPA Network," Proceedings of the 1974 National Computer Conference, pp. 767-780.

2 Balzer, R. M., T. E. Cheatham, and S. Crocker, National Software Works Design, USC/Information Sciences Institute, ISI/RR-73-16 (in progress).

3 ---, The National Software Works, USC/Information Sciences Institute, ISI/RR-73-18 (in progress).

4 Kimbleton, S. R., "An Analytic Framework for Computer System Sizing and Tuning," Proceedings of the NBS/ACM Computer Performance Evaluation Workshop, San Diego, California, March 1973. Also appeared as P-5162, The Rand Corporation, Santa Monica, California, January 1974.

5 Ellis, T. O., L. Gallenson, J. F. Heafner, and J. T. Melvin, A Plan for Consolidation and Automation of Military Telecommunications on Oahu, USC/Information Sciences Institute, ISI/RR-73-12, May 1973.

6 Boehm, B. W., T. E. Bell, and S. Jeffery, eds., "Introductory Comments" in Proceedings of the NBS/ACM Computer Performance Evaluation Workshop, San Diego, California, March 1973.

7 Kleinrock, L., "Analytic and Simulation Methods in Computer Network Design," AFIPS Conference Proceedings, 1970 Spring Joint Computer Conference, pp. 569-579.

8 Frank, H., I. T. Frisch, and W. Chou, "Topological Considerations in the Design of the ARPA Computer Network," AFIPS Conference Proceedings, 1970 Spring Joint Computer Conference, pp. 581-587.

9 Frank, H., R. E. Kahn, and L. Kleinrock, "Computer Communication Network Design--Experience with Theory and Practice," AFIPS Conference Proceedings, 1972 Spring Joint Computer Conference, pp. 255-270.

10 Kimbleton, S. R., A Fast Approach to Computer System Performance Prediction, USC/Information Sciences Institute, ISI/RR-74-20 (in progress).

RESEARCH RESOURCES

PROJECT LEADER: Thomas L. Boynton

RESEARCH STAFF: Robert E. Hoffman
Jeff Jacobs
John J. Vittal

CONTRIBUTING STAFF: Louis Gallenson
Robert H. Parker
Leroy C. Richardson

RESEARCH STAFF SUPPORT: Jerry Pipes
Deborah E. Williams

STUDENT AIDES: Danny L. Charlesworth
Dale Chase
Ronald L. Currier
Mark Horton
Jimmy T. Koda
Kyle Lemmons
Margaret Mathers

INTRODUCTION

The ISI time-sharing facility is operated as a research and service center in support of a broad set of ARPA projects. It currently services 400 users, 85 percent of whom access the facilities via the ARPANET from locations extending from London, England to Hawaii. All facilities of the operating system are available to all users, whether they are connected through the ARPANET locally or remotely.

The system consists of two Digital Equipment Corporation PDP-10 CPUs, Bolt Beranek and Newman paging box, large-capacity memory, high-performance paging drum, on-line file storage, and associated peripherals (see Figure 10.1). In conjunction with the large core and drum memory, the BBN TENEX operating system provides considerable computing capacity and is capable of supporting a wide variety of simultaneous users.

GENERAL SYSTEM UPGRADING

In a continuing effort to improve system availability, capacity, and efficiency, additions have been made in hardware and software and in support personnel.

Hardware

The hardware acquired in the past year as part of the general upgrading

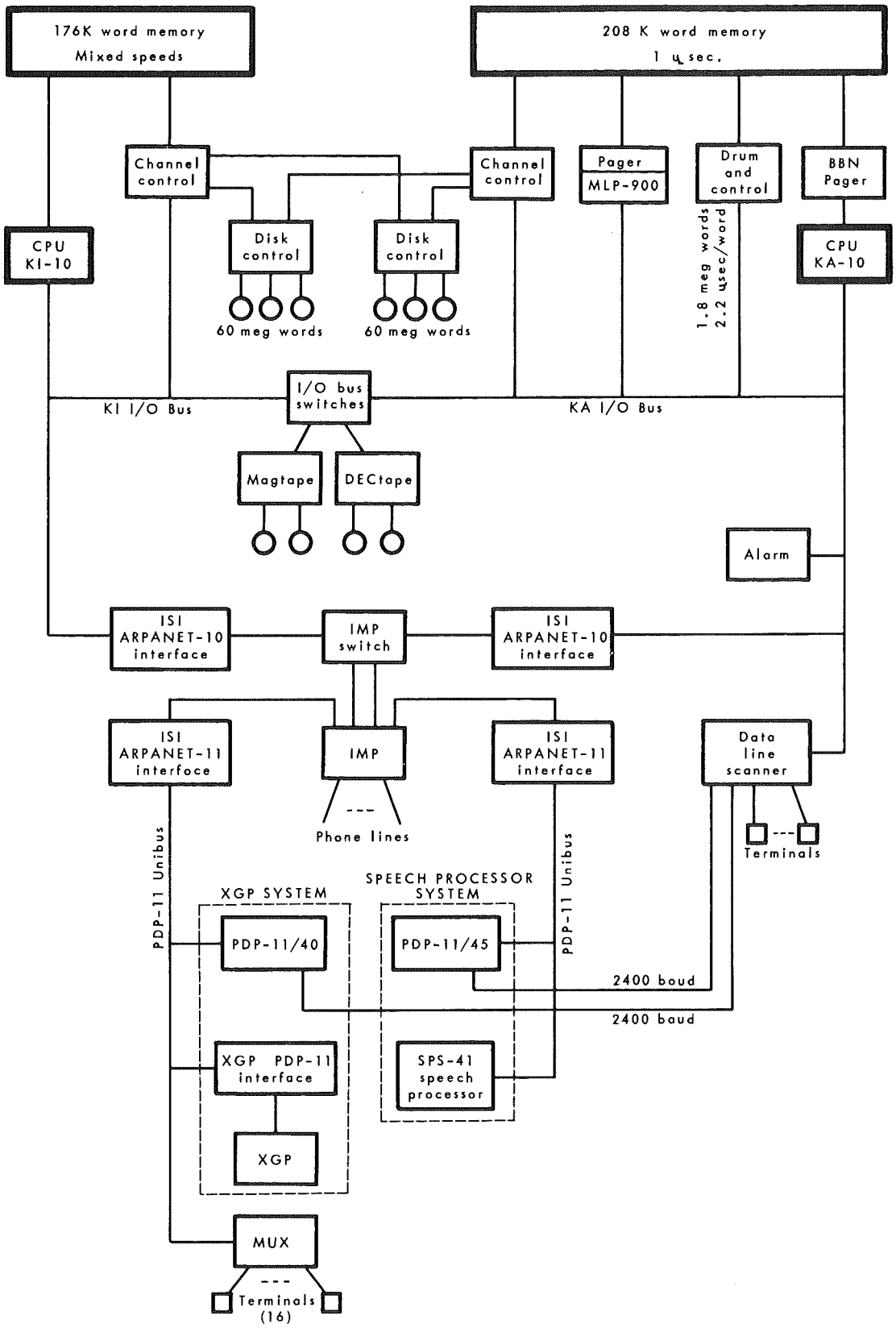

Figure 10.1 ISI research resources facility

effort includes a second PDP-10 CPU, an additional 128K of high-speed (850-nsec) memory, six CALCOMP 230 disk drives that will approximately double the present on-line file storage, and several interfaces and switches designed and built at ISI. Figure 10.1 shows the current ISI PDP-10 configuration. Note that the two CPU's, the KA-10 and the new KI-10, do not operate in a dual CPU mode. Instead, the main goal of having the two CPU's and the switches is to provide a significant increase in the availability of the ISI "primary" machine. Thus if the CPU acting as the primary machine breaks down or is needed for preventive hardware maintenance, system software maintenance, or development, then the other CPU may be started as the primary machine and service continued after a brief (15-30 minutes) interruption to switch machines. This ability to switch machines is made possible by the several components described below.

A second copy of the ISI ARPANET-10 interface produced for the KI-10 is now operational. This interface, together with the first ISI ARPANET-10 interface running on the KA-10, is attached to an ISI-designed electronic IMP switch that allows the interfaces to be easily switched from one distant IMP port to the other. The purpose of this switch is to allow either of the two PDP-10's to assume the identity of the primary outside user machine on the ARPANET and to allow rapid, error-free changeover from one primary machine to the other.

Electronic PDP-10 I/O bus switches have also been designed and developed to allow the two PDP-10's to share peripheral equipment (see Figure 10.2). The two PDP-10 buses can be looped through each switch. A single stub slot is provided to

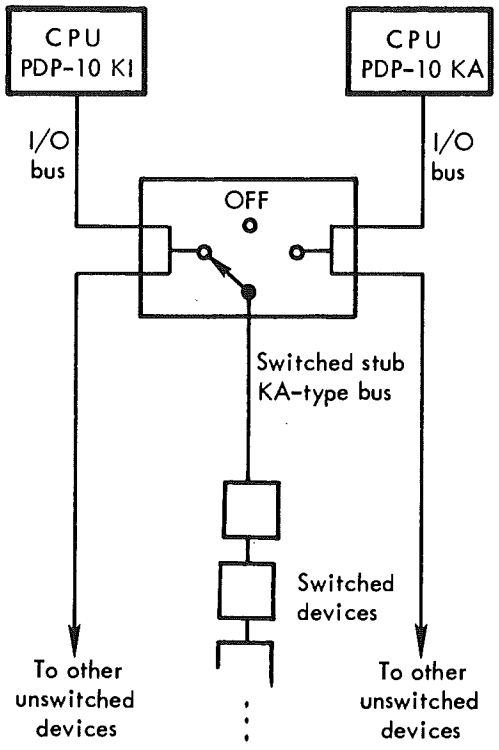

Figure 10.2 I/O bus switch

allow "daisy-chaining" of peripheral devices. The switch accommodates either KA or KI I/O buses, but the stub slot accommodates a KA bus only. The switch also has a center off position to detach the switched peripheral equipment from both machines for preventive maintenance or service. When the switch is thrown, a half-second timer resets the peripheral device and then engages the device onto the main computer bus to provide trouble-free switching. Two complete switches are now operational, one switching the DECtapes and the other the magnetic tapes between the KA-10 and KI-10.

In order to alert operators to system malfunctions more quickly, one interface not previously reported has been implemented for the primary machine. A "watchdog" or "deadman" timer monitors the CPU, tripping both a visual alarm at every terminal and an audible alarm on each floor of ISI if the CPU goes down. The alarm has an override feature and a silence feature that overrides either the two types of alarms or only the audible alarm, respectively.

The new CALCOMP disk system also will contribute to the ability to switch primary machines quickly by making it possible to switch the disk controller from one machine to the other using its two-channel switch feature.

Software

The demand for ISI's computing capacity far exceeded supply for most of the year, and a way was needed to reduce the load on the system and to restrict access to designated users at scheduled times, as specified by ARPA and ISI managements. In March, ISI installed a modified version of the group allocation scheme developed at Stanford Research Institute for controlling user access to the PDP-10. The group allocation scheme, as modified by ISI, divides the total number of job slots available on the system into two categories: general-access slots permitting full use of the system and limited-access slots restricting the user to certain facilities, principally message and text editing activities. These access slots are further subdivided into groups, with each group defined by the names of group members and by a quota (which may vary in size throughout the day) that specifies the number in that group guaranteed access to the system for general use. For most groups, the number

of group members is much greater than the quota, and a user may therefore not be able to log in on-quota. If the system as a whole is still under quota (because some other group is not at quota), then the user will be permitted to log in off-quota. If the slot taken up by an off-quota user is later needed to serve a new on-quota user, then the off-quota user is given a five-minute warning and logged off.

Management designates the group members and the quotas for each group throughout the day. By setting the system's total available quota to a sufficiently low level, management can reduce the load on the system so that those who can log on will find the system responsive and effective. Although the group allocation system does not directly allocate the scarce central processing capacity, the ISI system supports enough users that their requests usually make an acceptable average central processor demand. In all, the initial experience with group allocation has been good.

Before group allocation was installed, a program was developed to monitor the system load and to warn the operator when it exceeded a tolerable level. Selected individuals were then asked to temporarily stop their computing. The program is now being used to monitor the working of group allocation and to provide statistics on system usage. In order to understand load problems, several analysis programs were developed that display the system use data for management.

Support Personnel

ISI has hired and trained student operators to provide twenty-four-hour machine service. The primary responsibility of the operator is to assist the system staff in maintaining the operating system and assuring the continuous functioning of the overall system. The on-duty operator is principally engaged in running system maintenance programs and monitoring machine-room equipment. The latter function includes detecting failures on the computer or its support equipment, taking corrective action, and notifying appropriate system personnel in the event of a nonrecoverable failure.

LOCAL PROJECT SUPPORT

The TENEX facility has been utilized extensively in support of local projects. The staff makes use of all of the available standard subsystems (e. g., editors, compilers, assemblers, and utilities). Additionally, staff members have written subsystems and utilities in support of their own projects (e. g., a program for copying PDP-11 programs to the PDP-11's and TENEX accounting package extensions). The facility has supported less frequently used subsystems at the special request of users (e. g., PDP-11 cross assemblers and the DECUS Scientific Subroutine Package).

ISI has also taken delivery of three DEC PDP-11's. One of these, an 11/45, is described in Section 7. The other two are 11/40's intended to support the XGP's (see Section 8). One 11/40 is being used until new IMP ports are provided in two modes: during the day the PDP-11 serves as many as 16 local terminal users as a local terminal support processor to the ARPANET, and at night it supports the XGP.

An ISI-designed interface, the ISI ARPANET-11 interface, provides a local host connection to an ARPANET IMP for any model PDP-11 minicomputer. The interface operates in full-duplex mode to the IMP, employing four-way handshaking of serial data. A direct memory access capability allows the interface to operate from PDP-11 memory directly for data transfers without the intervention of the PDP-11 CPU. The dialogue with PDP-11 memory is handled on a starting address and byte count basis, where each half of the duplex interface operates into or out of a unique buffer area in core.

Hardware features of the interface include a capability for byte swapping on a per-word basis, four diagnostic modes (LOCK, LOOP, TEST INC, and LOCAL), extended memory capability, complete control panel, separate power supply, odd output starting address and odd byte count capability, output buffer chaining, a factor-of-four size reduction over other existing designs, modularity (each half of the interface is one unique card), low power consumption (27 watts), and a total front panel space (including control panel) of only 10-1/2 inches. Two ISI ARPANET-11 interfaces are now operational, one supported by the PDP-11/45 running ELF (part of the speech processing effort) and the other supported on a PDP-11/40 running ELF with up to sixteen ARPANET users. ELF is a multiuser PDP-11 operating system

developed by Speech Communications Research Laboratory, Santa Barbara, specifically designed to allow PDP-11's to be attached to the ARPANET.

MLP-900 PROCESSOR

Monitor modifications to support the initial installation and checkout of the MLP-900 have been developed and verified. These modifications allow basic processor-to-processor communication through both the I/O bus and memory. This is the first step in a multistep plan for fully integrating the MLP into the TENEX operating system (see Section 3).

PROGRAMMED AUTOMATION

PROJECT LEADER: Robert H. Anderson

RESEARCH STAFF: Ernest M. Hinds
Gopal K. Kadekodi
Nake M. Kamrany
Bennet P. Lientz
Elliott A. Ponchick

CONSULTANTS: Michael Boretsky
Alexander F. Brewer
Vernon Edwards
Jack Rosenberg

RESEARCH STAFF SUPPORT: Sandra F. Whitaker

INTRODUCTION

The project goals from July 1973 to December 1973 have been to 1) evaluate the technological feasibility of significant advancements in computer-based manufacturing systems for discrete product manufacture; 2) evaluate the economic impact on the DoD and the U. S. economy resulting from the implementation of those advancements; and 3) define the specific development program and resources required to achieve those advancements. The evaluations and recommendations of the project's staff were provided to ARPA in a final report on the study phase of the project [1], consisting of a major economic analysis of the costs, labor, and industry-structure factors characterizing DoD procurement. In addition, the report presented several detailed case analyses of particular DoD-related manufacturing operations in order to facilitate the understanding of the manufacturing enterprise as a system.

The following selection describing the project's achievements and expected impact is taken from the Preface to that report.

> In October 1971, a research effort was begun at the request of the Advanced Research Projects Agency of the Department of Defense to investigate the feasibility of significant production advancements using computer-based manufacturing technology and to evaluate the impact such advancements might have on DOD procurement of manufactured goods.
>
> This report presents specific recommendations for a major high-impact ARPA-sponsored research and development program in advanced computer-based manufacturing technology, including the objectives, required level of effort, milestones, and duration of such a program. The economic analysis in this report also provides a major source of information on many key characteristics of DOD-related manufacturing industries. The data obtained in the course of our study will be of great importance to other governmental agencies and to research contractors in formulating and evaluating complementary research and development programs in computer-based manufacturing.

PROJECT STATUS

The Project was terminated at the close of the study phase by mutual agreement of ARPA-IPTO and ISI. At the time it was concluded that it would not be possible to adequately staff a combined research and development effort to explore fully the conclusions of the final report.

REFERENCE

1 Anderson, R. H., and N. M. Kamrany, Advanced Computer-Based Manufacturing Systems for Defense Needs, USC/Information Sciences Institute, ISI/RR-73-10, September 1973.

COLLOQUIA

May 1973 to May 1974

MAY

Bob Balzer, ISI
Report on Pajaro Dunes Problem Solving Conference
18 May 1973.

Dan Cohen, Harvard University
Squeezing high bandwidth for graphics from a network
24 May 1973.

JUNE

Ellis Cohen, ISI
Object schematology: pride and protection
15 June 1973.

Joe Olive, Bell Telephone Laboratory
Speech work
22 June 1973.

Bill Wulf, Carnegie-Mellon University
Some tentative thoughts on another approach to high-speed computation
22 June 1973.

Larry Ragland, ISI
A verified program verifier
28 June 1973.

JULY

Jim King, IBM
Effigy: A symbolic executor
3 July 1973.

Roger Schank, Stanford University
The fourteen primitive actions and their inferences
5 July 1973.

Bill Wulf, Carnegie-Mellon University
Some tentative thoughts on another approach to structured programming
5 July 1973.

Peter Deutsch, Xerox PARC
An interactive program verifier
6 July 1973.

Neil Miller, Stanford University
Artificial Intelligence Laboratory
Speech enhancement
10 July 1973.

Bill Wulf, Carnegie-Mellon University
Informal presentation on HYDRA and its protection
12 July 1973.

Bill Mark, MIT
Artificial Intelligence Laboratory
MAPL language
16 July 1973.

Ellis Cohen, ISI
Formal semantic approach to protection
27 July 1973.

John Barr, UCLA
Interactive man-machine communications
31 July 1973.

Don Waterman, Carnegie-Mellon University
Summary of the AI-Psychology workshops at Carnegie
31 July 1973.

AUGUST

Ben Leintz, USC
A simulation program for the automation project
3 August 1973.

Bill Mark, MIT
Artificial Intelligence Laboratory
Model debugging in automatic programming
7 August 1973.

Dan Cohen, Harvard University
Network graphics and flight simulation
7 August 1973.

Terry Winograd, MIT
Artificial Intelligence Laboratory
Breaking the complexity barrier (again)
9 August 1973.

Ron Tugender, ISI
Interface between microprogramming and language design
10 August 1973.

Donald Good, ISI
Axiomatic models of programming languages
10 August 1973.

Andee Rubin, ISI
A quick look at time
14 August 1973.

Beau Shell, ISI
Experiments with a mixed declarative, assertive and procedural language
16 August 1973.

SEPTEMBER

Jay Moore, University of Edinburgh
Proving theorems about Lisp functions
3 September 1973.

Peter Lauer, University of Newcastle upon Tyne
Consistent and complementary formal definitions of programming languages
14 September 1973.

Peter Christy, University of California at Berkeley
Application of information technology to health care delivery
24 September 1973.

OCTOBER

Randy Cole, University of Utah
The removal of unknown image blurs by homomorphic filtering
4 October 1973.

Michael J. Flynn, Palyn Associates, Inc.
Towards more efficient organizations
9 October 1973.

D. R. Musser, University of Texas
Algebraic aspects of program verification
10 October 1973.

W. W. Bledsoe, University of Texas
Interactive theorem proving
10 October 1973.

D. C. Luckham, Stanford University
Program and theorem proving activities at Stanford
10 October 1973.

Lou Gallenson, ISI
Video display systems for ISI
18 October 1973.

Bob Balzer, ISI
Software production facility project: status and opportunities
18 October 1973.

Dan Cohen, ISI
Computer-aided instruction
25 October 1973.

Jean-Marie Cadiou, IRIA, France
Mechanizable proofs about parallel processes
26 October 1973.

NOVEMBER

Chino Srinivason, Rutgers University
The architecture of coherent information systems: a general problem-solving system
16 November 1973.

Dick Mandell, Compata Inc.
Computer performance estimation
20 November 1973.

DECEMBER

Jim Moore, Carnegie-Mellon University
Knowledge representation in the Merlin System
17 December 1973.

Nake Kamrany, ISI
MIT's Sub-Sahara Project: a system dynamics approach
20 December 1973.

Martin Kay and Bill Mann, ISI
Human communications with humans and machines
26 December 1973.

JANUARY

Warren Teitelman, Xerox Corporation
New capabilities in Lisp
9 January 1974.

E. M. Greenawalt, University of Texas
Automatic program partitioning
14 January 1974.

Nadine Malcolm, ISI
Data bases
29 January 1974.

FEBRUARY

John S. Brown, Bolt Beranek and Newman, Inc.
SOPHIE: An intelligent CAI system for electronic troubleshooting
19 February 1974.

MARCH

Larry Yelowitz, New Mexico Institute of Mining and Technology
Joint program/proof development
7 March 1974.

David Rumelhart, University of California at San Diego
SOL: Semantic Operating Language
8 March 1974.

George Heidorn, IBM
English as a very high level language for simulation programming
27 March 1974.

APRIL

Tom Martin, Annenberg School of Communications
Comparative analysis of interactive retrieval systems: questions and answers
9 April 1974.

Steve Kimbleton, ISI
Structures and solution methodologies for network performance
24 April 1974.

PUBLICATIONS

1 Anderson, Robert H., Programmable Automation: The Future of Computers In Manufacturing, ISI/RR-73-2, March 1973; also appeared in Datamation, Vol. 18, No. 12, December 1972, pp. 46-52.

2 ---, and Nake M. Kamrany, Advanced Computer-based Manufacturing Systems for Defense Needs, ISI/RR-73-10, September 1973.

3 Balzer, Robert M., Automatic Programming, ISI/RR-73-1 (draft only).

4 ---,CASAP: A Testbed for Program Flexibility, ISI/RR-73-5 (In progress).

5 ---, Another Look at Program Organization, ISI/RR-73-6 (In progress).

6 ---, Human Use of World Knowledge, ISI/RR-73-7, March 1974.

7 ---, A Global View of Automatic Programming, ISI/RR-73-9 (In progress).

8 ---, AP/1 - A Language for Automatic Programming, ISI/RR-73-13 (In progress).

9 ---, Language-Independent Programmer's Interface, ISI/RR-73-15, March 1974; AFIPS Conference Proceedings, Vol. 43, AFIPS Press, Montvale, N. J., 1974.

10 ---, Norton R. Greenfeld, Martin J. Kay, William C. Mann, Walter R. Ryder, David Wilczynski, and Albert L. Zobrist, Domain-Independent Automatic Programming, ISI/RR-73-14, March 1974; also to appear in Proceedings of the International Federation of Information Processing Congress, 1974.

11 ---,Thomas E. Cheatham, Stephen D. Crocker, and Stephen Warshall, National Software Works Design, ISI/RR-73-16 (In progress).

12 ---, Thomas E. Cheatham, Stephen D. Crocker, and Stephen Warshall, The National Software Works, ISI/RR-73-18 (In progress).

13 Bisbey, Richard L., and Gerald J. Popek, Encapsulation: An Approach to Operating System Security, ISI/RR-73-17, October 1973.

14 Carlstedt, Jim, Toward Explicit Policy: A Structured Approach to Decision and Evaluation of Protection Systems (In progress).

15 Ellis, Thomas O., Louis Gallenson, John F. Heafner, and John T. Melvin, A Plan for Consolidation and Automation of Military Telecommunications on Oahu, ISI/RR-73-12, June 1973.

16 Goldberg, Joel, and Leroy Richardson, PRIM User's Guide (In progress).

17 Kamrany, Nake M., A Preliminary Analysis of the Economic Impact of Programmable Automation Upon Discrete Manufacturing Products, ISI/RR-73-4, October 1973.

18 London, Ralph L., Shigeru Igarashi, and David C. Luckham, Automatic Program Verification I: A Logical Basis and Its Implementation, ISI/RR-73-11, May 1973; also appeared in Artifical Intelligence Memo 2000, Stanford University, May 1973.

19 Oestreicher, Donald R., A Microprogramming Language for the MLP-900, ISI/RR-73-8, June 1973; also appeared in the Proceedings of the ACM Sigplan Sigmicro Interface Meeting, New York, May 30-June 1, 1973.

20 ---, General Purpose Microprogramming Language Reference Manual (In progress).

21 ---, and Joel Goldberg, MLP-900 Reference Manual (In progress).

22 Richardson, Leroy, PRIM Overview, ISI/RR-74-19, February 1974.

23 Annual Technical Report, May 1972 - May 1973, ISI/SR-73-1, 1973.

DOCTORAL THESES IN PROGRESS AT ISI

AUTOMATIC PROGRAMMING

Richard Hale, Optimizing of Searching in Pseudo-augmented-transition-network Models.

Robert W. Lingard, A Representation for Semantic Information within an Inference-making Computer Program.

Nadine Malcolm, Methods for Improved Access Time in a Relational Data Base.

David Wilczynski, Interactive Domain Acquisition for Automatic Programming.

PRIM/MLP-900

John M. Malcolm, Efficient Language-specified Instruction Sets for a Microprocessor.

Martin D. Yonke, A Semantic Approach for On-line Program Development.

PROGRAM VERIFICATION

Donald S. Lynn, Automatic Program Verification: Compiler Proofs.

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ISI/SR-74-2	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* A Research Program in the field of Computer Technology		5. TYPE OF REPORT & PERIOD COVERED Annual Technical Report May 1973 - May 1974 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR*(s)* Keith W. Uncapher (Principal Investigator)		8. CONTRACT OR GRANT NUMBER*(s)* DAHC 15 72 C 0308
9. PERFORMING ORGANIZATION NAME AND ADDRESS USC Information Sciences Institute 4676 Admiralty Way Marina Del Rey, California 90291		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS ARPA Order #2223
11. CONTROLLING OFFICE NAME AND ADDRESS Advanced Research Projects Agency 1400 Wilson Blvd. Arlington, Virginia 22209		12. REPORT DATE May 1974 13. NUMBER OF PAGES 139
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*		15. SECURITY CLASS. *(of this report)* 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Distribution unlimited. Available from National Technical Information Service, Springfield, Virginia 22151.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

1: automatic programming, domain-independent interactive system, natural language, nonprocedural language, nonprofessional computer users, problem solving, problem specification, process transformation, world knowledge

2: interactive theorem proving, lemma generator, Pascal, program correctness, program verification, Reduce, symbolic executor, verification condition

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

This report summarizes the research performed by USC/Information Sciences Institute from 17 May 1973 to 16 May 1974. The research is aimed at applying computer science and technology to problem areas of high DoD/military impact.

The ISI program consists of ten research areas: Automatic Programming- the study of acquiring and using problem knowledge for program generation; Program Verification- logical proof of program validity; Programming Research Instrument- development of a major time-shared microprogramming facility; Protection

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601 |

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

19. Key Words (continued)
generator
3: ARPANET, control memory, microprogrammed processor, microprogramming, microprogramming language, microvisor, MLP-900, operating systems, resource sharing, TENEX, time sharing, writable control memory
4: access control, computer security, encapsulation, error analysis, error-driven evaluation, error patterns, evaluation methods, protection mechanisms, software security, verification
5: computer security, COTCO, interactive message service, message service, reliability, terminal based message service
6: computer terminals, interactive message service, office automation, CONNECT, nonprofessional computer users, terminal-based message service
7: computer network, digital voice communication, network conferencing, packet-switched networks, secure voice transmission, signal processing, speech processing, vocoding
8: liquid crystal displays, minimum communications terminal, plasma displays, portable terminals, video display system, Xerox Graphics Printer
9: computer networks, configuration control, decisionmaking, information display, load leveling, network data base, network management, network performance, performance analysis, performance measurement
10: TENEX, KA/KI, PDP-10, PDP-11/40, resource allocation, computer network, user quotas, ARPANET interface

---

20. Abstract (continued)
Analysis- methods of assessing the viability of security mechanisms of operating systems; Command and Control Message Technology- study of advanced computer-based techniques for military message handling; Information Automation- development of a user-oriented message service for large scale military requirements; Network Secure Speech- work on low-bandwidth, secure voice transmission using an asynchronous packet-switched network; Technology Support- development of Xerox Graphics Printer facilities, portable terminals, and military office terminal system; Network Management Information Center- development of a network performance-measurement methodology; and Research Resources- operation of TENEX service and continuing development of advanced support equipment.

USC / INFORMATION SCIENCES INSTITUTE
4676 Admiralty Way Marina del Rey California 90291