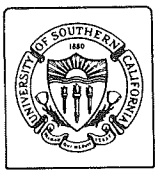
ISI/SR-84-138



University of Southern California

1983

ANNUAL TECHNICAL REPORT

July 1982 – June 1983

A Research Program in Computer Technology

INFORMATION SCIENCES INSTITUTE

213/822-1511

4676 Admiralty Way/Marina del Rey/California 90292-6695

Unclassified

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

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ISI/SR-84-138	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* 1983 Annual Technical Report: A Research Program in Computer Technology		5. TYPE OF REPORT & PERIOD COVERED Annual Technical Report July 1982 - June 1983 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) ISI Research Staff		8. CONTRACT OR GRANT NUMBER(s) F30602-81-K-0056 F49620-79-C-0181 MDA903 81 C 0335 MCS-7918792
9. PERFORMING ORGANIZATION NAME AND ADDRESS USC/Information Sciences Institute 4676 Admiralty Way Marina del Rey, CA 90292-6695		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wilson Blvd. Arlington, VA 22209		12. REPORT DATE July 1984 13. NUMBER OF PAGES 155
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)* ----------		15. SECURITY CLASS. *(of this report)* Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

This document is approved for public release and sale; distribution is unlimited.

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. implementation of interactive systems, knowledge base, knowledge-based inference, natural interface, online services, process script, service building, tool building, user interface
2. bit-mapped graphics, consistent underlying environment, COUSIN, extensible environment, integration, interactive devices, knowlege-based approach, MENUNIX, menus, natural language, semantic model, service building, service execution, windows

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 July 1, 1982, to June 30, 1983, for the Defense Advanced Research Projects Agency, the Air Force Office of Scientific Research, the National Science Foundation, and the Air Force Systems Command, Rome Air Development Center. The research is focused on the development of computer science and technology, which is expected to have a of high DoD/military impact.

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

Unclassified

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

11. CONTROLLING OFFICE NAME AND ADDRESS (continued)

National Science Foundation
1800 G Street NW
Washington, DC 20550

Air Force Office of Scientific Research
Building 410, Bolling Air Force Base
Washington, DC 20332

Air Force Systems Command,
Rome Air Development Center
Griffiss Air Force Base, NY 13441

19. KEY WORDS (continued)

3. automation–based paradigm, GIST specification language, granularity, high–level editing, locally formal notations, mappings, reimplementations, software development, software maintenance, software specification, Will programming language
4. artificial intelligence, computing environments, constraints, knowledge representation, programming environments, rapid prototyping, software modeling, specification languages
5. computer mail, gateways, interconnection, internetwork protocol, multimedia mail, networks, protocol design, protocols, protocol verification, simple mail transfer protocol, transmission control protocol, type–of–service
6. briefing aid application program, command graphics, computer graphics, high–level graphics language, network–based graphics, on–line map display
7. computer network, digital voice communication, network conferencing, packet satellite network, packet–switched networks, packet video, packet voice, secure voice transmission, signal processing, speech processing, vocoding
8. design rules, device fabrication service, device testing, integrated circuit oriented language, MOSIS–MOS Implementation System, silicon compilation, standard pad frames, VLSI design, VLSI design library, wafer testing
9. design development, electronic office, environment, human dimension, optimum workspace, productivity, storage, user control, workstation
10. artificial intelligence, discourse modeling, functional linguistics, grammar development, human–computer interfaces, natural language, Nigel, Penman, systemic linguistics, text generation, text planner, text structure
11. debugging, documentation, explanation, formal specifications, GIST, paraphraser, symbolic evaluator, syntax, transformational implementation
12. formal development structure, grammar–based editing, GIST, Paddle, Popart, reimplementation, replay, structure comparison, transformation–based maintenance
13. address space limitations, C, Interlisp, LISP dialects, UNIX, VAX, VMS
14. application software, ARPANET, computer network, hardware, Interlisp, KA/KI, KL10/KL20, network services, operations, PDP11/45, QLISP, resource allocation, system software, TOPS–20, UNIX, upgrades, VAX 11/750–80, VMS
15. computer communication networks, packet radio, survivable networks, system support
16. internetwork protocol, transmission control protocol, protocol implementation
17. distributed processing, portable workstations, survivable networks
18. distributed processing, local networks, personal computers, workstation environment

20. ABSTRACT (continued)

The ISI program consists of eighteen research areas: *Consul* – development of a knowledge–based interactive system that can provide services and explanations in response to user requests, including natural language requests from users with little computing experience; *CUE* – development of a methodology to automate the process of generating a consistent command–, editor–, and menu–based interface to integrated software applications using a knowledge–based approach; *Mappings* – development of transformations for converting high–level specifications of programs into implementations in software or VLSI; *Information Management* – combining artificial intelligence, software technology, and database techniques to build a more highly integrated, uniform, and user evolvable computing environment for both software development and information services; *Internet Concepts Research* – exploring aspects of protocols for the interconnections of computer communication networks, specifically the design and prototype implementation of an internetwork computer message system and the design of internetwork host and gateway protocols; *Command Graphics* – development of a device–independent graphics system and graphics–oriented command and control applications programs; *Wideband Communications* – exploration of the technology needed to support satellite–based wideband computer communication with dynamic intercommunication of many ground stations via packet–switched sharing of a single satellite channel, and investigation of methodologies for supporting various media of information in this environment; *VLSI* – providing a low–cost, fast turnaround LSI/VLSI device fabrication service to support a geographically distributed VLSI research community with no direct access to VLSI fabrication but with access to a computer communication network, and conducting research on the VLSI design problem, from algorithms to "silicon" compilation; *Office Environments* – development of a plan for an optimal office space which offers physical ease and comfort, and allows for a high degree of performance; *Knowledge Delivery* – development of new methods for autonomous creation of text by machine, with the focus on fluent, easily controlled sentence and paragraph production; *Specification Validation* – determining whether a specification meets the end user's intent by paraphrasing it and/or all of its possible behaviors in natural language; *Supervisory Control of Transformational Implementation Systems* – creating a framework to aid a programmer to formally develop software from specifications via transformations by automatically supplying sequences of low–level transformations which accomplish or facilitate high–level transformations; *Interlisp* – development and maintenance of portable, large address–space Interlisp implementations; *Computer Research Support* – operation of reliable computing facilities and continuing development of advanced support equipment; *Strategic C3 System Experiment Support* – participation in a Strategic Command, Control, and Communication systems experiment demonstrating and evaluating the use of new technologies (such as the ARPANET, packet radio, network security, and distributed knowledge–based techniques); *TCP/IP Implementation Support* – implementing TCP/IP protocols into active use in the operational ARPANET and MILNET by installing and debugging host software to support the ISI user community; *Exportable Workstation Systems* – development of a remote testbed environment of advanced workstations and servers; *New Computing Environment* – exploring, determining, and implementing the next generation of computers and computing facilities for the ISI research environment.

ISI/SR-84-138

University of Southern California

1983

ANNUAL TECHNICAL REPORT

July 1982 - June 1983

A Research Program in Computer Technology

Principal Investigator and Executive Director:
Keith W. Uncapher

Deputy Director:
Thomas O. Ellis

Prepared for the Defense Advanced Research Projects Agency

Effective date of contract 1 July 1981
Contract expiration date 30 June 1984
Contract # MDA 903 81 C 0335
ARPA order 4242

INFORMATION SCIENCES INSTITUTE

213/822-1511

4676 Admiralty Way/Marina del Rey/California 90292-6695

This research is supported by contracts with DARPA (MDA903 81 C 0335), NSF (MCS-7918792), AFOSR (F49620-79-C-0181), and RADC (F30602-81-K-0056). Views and conclusions contained in this report are the authors' and should not be interpreted as representing the official opinion or policy of DARPA, NSF, AFOSR, RADC, the U.S. Government, or any person or agency connected with them.

PROJECT FUNDING

The following projects were funded by the Defense Advanced Research Projects Agency, under contract no. MDA903 81 C 0335: Consul, CUE, Mappings, Information Management, Internet Concepts Research, Command Graphics, Wideband Communications, VLSI, Office Environments, Computer Research Support, Strategtic C3 System Experiment Support, TCP/IP Implementation Support, Exportable Workstation Systems, and New Computing Environment.

The Knowledge Delivery project was funded by the Air Force Office of Scientific Research, under contract no. F49620–79–C–0181.

The Specification Validation project was funded by the Air Force Systems Command, Rome Air Development Center, under contract no. F30602–81–K–0056.

The Supervisory Control of Transformational Implementation Systems project was funded by the National Science Foundation, under contract no. MCS–7918792.

The Interlisp project is internally funded by the University of Southern California.

CONTENTS

Summary *iv*

Executive Overview *v*

1. Consul *1*
2. CUE *11*
3. Mappings *19*
4. Information Management *29*
5. Internet Concepts Research *39*
6. Command Graphics *55*
7. Wideband Communications *59*
8. VLSI *73*
9. Office Environments *83*
10. Knowledge Delivery *91*
11. Specification Validation *97*
12. Supervisory Control of Transformational Implementation Systems *105*
13. Interlisp *115*
14. Computer Research Support *119*
15. Strategic C3 System Experiment Support *125*
16. TCP/IP Implementation Support *127*
17. Exportable Workstation Systems *131*
18. New Computing Environment *135*
19. Publications *141*

SUMMARY

This report summarizes the research performed by USC/Information Sciences Institute from July 1, 1982, to June 30, 1983, for the Defense Advanced Research Projects Agency, the Air Force Office of Scientific Research, the National Science Foundation, and the Air Force Systems Command, Rome Air Development Center. The research is focused on the development of computer science and technology, which is expected to have a high DoD/military impact.

The ISI program consists of eighteen research areas: *Consul* – development of a knowledge–based interactive system that can provide services and explanations in response to user requests, including natural language requests from users with little computing experience; *CUE* – development of a methodology to automate the process of generating a consistent command–, editor–, and menu–based interface to integrated software applications using a knowledge–based approach; *Mappings* – development of transformations for converting high–level specifications of programs into implementations in software or VLSI; *Information Management* – combining artificial intelligence, software technology, and database techniques to build a more highly integrated, uniform, and user evolvable computing environment for both software development and information services; *Internet Concepts Research* – exploring aspects of protocols for the interconnections of computer communication networks, specifically the design and prototype implementation of an internetwork computer message system and the design of internetwork host and gateway protocols; *Command Graphics* – development of a device–independent graphics system and graphics–oriented command and control applications programs; *Wideband Communications* – exploration of the technology needed to support satellite–based wideband computer communication with dynamic intercommunication of many ground stations via packet–switched sharing of a single satellite channel, and investigation of methodologies for supporting various media of information in this environment; *VLSI* – providing a low–cost, fast turnaround LSI/VLSI device fabrication service to support a geographically distributed VLSI research community with no direct access to VLSI fabrication but with access to a computer communication network, and conducting research on the VLSI design problem, from algorithms to "silicon" compilation; *Office Environments* – development of a plan for an optimal office space which offers physical ease and comfort, and allows for a high degree of performance; *Knowledge Delivery* – development of new methods for autonomous creation of text by machine, with the focus on fluent, easily controlled sentence and paragraph production; *Specification Validation* – determining whether a specification meets the end user's intent by paraphrasing it and/or all of its possible behaviors in natural language; *Supervisory Control of Transformational Implementation Systems* – creating a framework to aid a programmer to formally develop software from specifications via transformations by automatically supplying sequences of low–level transformations which accomplish or facilitate high–level transformations; *Interlisp* – development and maintenance of portable, large address–space Interlisp implementations; *Computer Research Support* – operation of reliable computing facilities and continuing development of advanced support equipment; *Strategic C3 System Experiment Support* – participation in a Strategic Command, Control, and Communication systems experiment demonstrating and evaluating the use of new technologies (such as the ARPANET, packet radio, network security, and distributed knowledge–based techniques); *TCP/IP Implementation Support* – implementing TCP/IP protocols into active use in the operational ARPANET and MILNET by installing and debugging host software to support the ISI user community; *Exportable Workstation Systems* – development of a remote testbed environment of advanced workstations and servers; *New Computing Environment* – exploring, determining, and implementing the next generation of computers and computing facilities for the ISI research environment.

EXECUTIVE OVERVIEW

USC/Information Sciences Institute (ISI) is a large, university-based information sciences research and development center, with programs embracing a blend of basic research, applied research, and systems development.

ISI's broad spectrum of research covers the areas of communication technology, VLSI, software production technology, and user-friendly systems. During the past year, there has been a sustained effort to enhance the quantity, quality, and focus of ISI's research in order to anticipate and meet emerging national needs and for DARPA-supported work, there is special emphasis on emerging DoD/military needs. A continuing effort is underway to enhance the research base to track the growing impact of the information sciences on DoD and the country. High-bandwidth communications, knowledge-based and expert systems, and advanced extensions of the MOSIS system, related to major changes in manufacturing technology, represent new or augmented research programs.

ISI plays a major role in supporting the military's technology transfer programs. ISI has continued its commitment to large segments of the DARPA research community in supplying high-quality shared computer support over national networks, and in developing the technology necessary for the transition of selected military sites from remote shared-cycle systems to distributed workstation-based environments.

A highlight of this reporting period has been the emergence of DARPA's MOSIS as a truly national resource. Over 40 universities and hundreds of designers now submit VLSI designs in electronic form via any network to MOSIS. MOSIS delivers chips and will soon deliver user-specified printed circuit boards to designers 30 to 35 days after receipt of a design.

ISI remains committed as a major technology transfer center to the military and, as such, a considerable portion of its resources are dedicated to the following:

- discovering critical areas of military C3 in which information processing can allow dramatic improvements in mission performance; and
- identifying required research areas and objectives leading to resultant technology in usable form.

The DARPA-sponsored research projects at ISI are as follows: Consul, CUE, Mappings, Information Management, Internet Concepts Research, Command Graphics, Wideband Communications, VLSI, Office Environments, Computer Research Support, Strategic C3 System Experiment Support, TCP/IP Implementation Support, Exportable Workstation Systems, and New Computing Environment. The non-DARPA-sponsored research projects at ISI are as follows: Interlisp (funded internally by the University of Southern California), Knowledge Delivery (sponsored by the Air Force Office of Scientific Research), Specification Validation (sponsored by the Air Force Systems Command, Rome Air Development Center), and Supervisory Control of Transformational Implementation Systems (sponsored by the National Science Foundation). The diversity of research interests at ISI provides a broad base for a healthy interchange and amplification of ideas and information.

Consul. The Consul project is exploring the use of knowledge-based technology to allow a natural interface between users and computer services. We conduct our research in the framework of a demonstrable user interface environment that includes facilities for understanding natural language

requests and producing natural language help and explanations. The system contains detailed models of users and services and a set of inference mechanisms to transform descriptions from one model into an appropriate description in the other. Consul research therefore focuses on representation and inference in the interactive systems domain. Consul's interface is designed to be used in conjunction with CUE, which provides a menu/command-based interface that interacts with Consul's knowledge-based components. The interface is provided in a distributed environment, with Consul residing on a server machine that can be shared by one or more CUE workstations on the same Ethernet.

CUE. The goal of the CUE project is to provide an extensible environment for building and using integrated interactive computer services. CUE will develop and deliver a working system in three years. In CUE, there are no boundaries between, say, the electronic mail service and the automatic calendar service. This type of automatic inter-service interaction requires an environment in which individual services can make assumptions about the properties and behavior of the other services in the system. These assumptions must relate to data structures, functional capability, and state information. The objective of the CUE project is to produce such a consistent underlying environment (CUE), in which users can interact with the system without regard to service boundaries.

Mappings. The primary goal of the Mappings project is to capture program specification, development, and implementation knowledge in formal transformations: high-level editing commands, high-level transformations, simplification steps, and optimization strategies. Since its inception the Mappings project has addressed the problem of representing programming knowledge in terms of our specification language, Gist, by discovering correctness-preserving transformations for translating Gist's high-level constructs into the lower level constructs used in more conventional programming languages. Hence, the Mappings project addresses the problem of encapsulating programming knowledge in terms of concise mappings from Gist into alternative implementations. In the future, more emphasis will be given to mappings for optimization of these implementations and to mappings which aid our incremental understanding of the design and development of specifications. These latter mappings we call "high-level editing commands" to emphasize that they change the specification in well understood ways.

Information Management. The Information Management project is designing and implementing a computing environment that significantly reduces the effort required to create, integrate, and evolve computing services, and enables users to customize these services without detailed knowledge of how they were specified. The basis for such improvements lies in raising the level at which these services are specified and modified. The primary task of this project is to implement a testbed software construction and maintenance environment. The testbed must also provide an execution environment for software constructed within it. This testbed must make a higher level of specification directly available to system builders and users without seriously compromising their ability to produce efficient software.

Internet Concepts Research. The Internet Concepts Research project extends and enhances computer communications. This work is based on the ARPA Internet, a system of interconnected computer communication packet networks. The ARPA Internet has working protocols for communication between heterogeneous computers via an interconnected collection of heterogeneous packet networks. This research involves work at several levels, host-to-host and gateway-to-gateway protocols and applications protocols. The basic protocols are largely complete now, but a number of extensions are being explored to integrate and extend the packet network technology. In the gateway and host level protocols, these extensions include mechanisms and

procedures for monitoring and maintaining performance and survivability, for allowing growth of the Internet, and for dynamic control of the Internet. In the applications level protocols, the focus will be on new uses of the Internet such as multimedia mail, and new capabilities in the gateway protocols. Another important aspect of the development of protocols is to investigate their correctness. This research studies the Transmission Control Protocol (TCP) using several protocol verification tools. The long-term goals of this project are to provide appropriate and effective designs for the primary user service applications in the internetwork communication environment.

Command Graphics. A major issue for the military is improved utilization of available data to enhance the Command and Control decision-making process. The military, like its private sector counterpart, currently finds itself in the midst of an information explosion. More computers and computer-controlled systems are being acquired, generating information in ever-increasing quantity and detail. For this information to be useful in decision making, it is necessary that computers take more active roles in storing, retrieving, analyzing, integrating, and presenting data. The man-machine interface is a critical link when computers are used to aid the decision maker. Information must be presented to the decision maker in ways that enhance and facilitate the decision process. Two-dimensional graphics can play an important role in improving this interface. Where spatial relationships exist, plotting the information on a graph, bar, or pie chart or positioning the information on a map can aid in the rapid assimilation of the information by the decision maker. Such two-dimensional displays can even disclose perspectives (e.g., a trend on a graph or a clustering of forces on a map) that would not be readily apparent from a table or list of numbers. Finally, two-dimensional displays provide a natural medium for integrating and fusing information. To address the above needs, ISI designed a graphics system based on the premise of distributing the processing load across hosts in a computer network. The architecture supports a wide variety of configurations ranging from clustering all functions on a single host to distributing each to a different host. ISI has also developed a set of generic graphics primitives (Graphics Language) by which pictures can be described and interacted with at the application level. The Graphics System is now being adapted for use in DARPA's Strategic C3 Program. Here the computing architecture consists of a network of VAXes running the UNIX operating system. This adaptation required reimplementing the Graphics System in the "C" programming language and interfacing it to the UNIX operating system.

Wideband Communications. The Wideband Communications project at ISI is one of several groups involved in the joint DARPA/DCA Wideband Packet Satellite Program. The objective of the Wideband Program is to explore the technical and economic feasibility of packet voice on a large scale, and to begin investigations of other media, such as packet video. ISI's role is to conduct experiments using the Wideband Network as it becomes available for regular, active use. ISI has been closely involved in efforts to make the network as reliable and useful as possible, and has established a schedule of regular tests and experiments in an effort to encourage real use of the network, thus gaining as much practical experience as possible. The ISI WBC project has expended considerable effort in developing an interface which provides voice access to the Wideband Network from ordinary telephones, in order to promote everyday use of the network and gain realistic user experience as early as possible. The Wideband Network is currently being actively used for packet voice experiments. Later, attention will be focused in other directions, including experiments with high-rate transmission of more conventional non-real-time data and the application of packet-switching techniques to media such as packet video for the first time. The ISI WBC project is conducting experiments to demonstrate the utility of packet video in the same way that the NSC program demonstrated the utility of packet voice.

VLSI. The VLSI design communities of DARPA and NSF require fabrication capabilities in order to investigate the design methodologies and architectures appropriate to VLSI where gate-count will exceed one million gates per device. Recognizing this need, DARPA established the MOSIS (MOS Implementation Service) system at ISI in January 1981. MOSIS has met its design objectives: it has reduced the cost of VLSI prototyping, shortened turnaround time for VLSI prototyping, freed designers from fabrication idiosyncracies, and made design less dependent on specific fabrication lines. Future services of MOSIS will include fabrication of printed circuit boards (PCBs), using a tooling preparation methodology common to both PCBs and integrated circuits. In addition, MOSIS will rapidly expand its vendor base for 3 micron CMOS/Bulk, judged to be the work horse of the design community for the next ten years. MOSIS is also taking steps to position its design community to exploit the 1.2 micron CMOS/Bulk technology that is being developed at several commercial and industrial laboratories. The idea is to develop the rules and techniques in parallel with the development of the technology, allowing the entire MOSIS community to use this technology upon the completion of its development. This strategy will narrow the gap between the availabilty of a new technology and the ability to use it.

Office Environments. As the man-machine relationship becomes an integral part of our lives, and the computer an extension of our minds, our environment must facilitate the use of hardware while offering a human dimension. The Office Environments project developed a plan for a workspace which offers physical ease and comfort, and allows for a high degree of performance. Current research in related areas was analyzed to help identify major issues and to avoid duplication. Combining research, design development, construction, and testing, the project produced a scaled prototype of an electronic office based on an existing ISI office area.

Knowledge Delivery. The usefulness of computers is often limited by their very poor ability to communicate with people. Computer users—and potential users—are often prevented from using information because that information is in an obscure, computer-internal notation. Many of these limits could be removed by a general technique for expressing computer information in English. A new technology of text generation is needed, one in which techniques can be refined and moved from one application system to another. Working toward this new technology, the Knowledge Delivery project is developing a theory and programs of text generation based on the Systemic Linguistics tradition. Significant progress has been made, both in grammar development and discourse modeling, within a general design framework which encompasses both domain-dependent and domain-independent parts. A text generation system called Penman has been designed to embody these developments, including a large computational grammar of English named Nigel. The Nigel grammar has been tested extensively, and a corresponding semantic notation for systemic grammars has been defined and tested. Future work on Knowledge Delivery is aimed at completing and integrating the existing developments in grammar and discourse, and on applying them in experimental text generators.

Specification Validation. Research at ISI has indicated that two major impediments to understandability of specifications are the unfamiliar syntactic constructs of specification languages and dynamic interactions between parts of the specification—parts that are often widely separated. These interactions may cause the specification to denote behaviors that were *not* intended by the original specifier, or not to denote behaviors that *were* intended. The Specification Validation project, now completed, has attempted to overcome the impediments of unfamiliar syntax and non-local interactions by constructing computer tools to make specifications more understandable, both to specifiers and to those unfamiliar with formal specification languages. One tool, the Gist paraphraser, addresses the syntax problem by directly translating a Gist specification (Gist is a

high-level specification language being developed at ISI) into English. Another pair of tools, a symbolic evaluator and a trace explainer, address the more difficult problem of making non-local specification interactions apparent by simulating the dynamic behavior implied by the specification and explaining the results of that simulation.

Supervisory Control of Transformational Implementation Systems. The primary goal of the Supervisory Control of Transformational Implementation Systems project was to design and develop a framework for understanding, reusing, and maintaining previous specifications and optimizations. The project developed the system support needed to facilitate the automated implementation of specifications using transformations. The SCTI project made considerable progress toward this goal, including the creation of a language-independent development system called Popart, with facilities for grammar-based editing, analysis, and transformation; a formal representation of program development; a compilable subset of our (wide-spectrum) formal specification language to act as a target language for transformation; a goal-directed transformation selection and application mechanism; and a system for recording and supporting the evolution of real systems (to improve our understanding of this evolutionary process). Our primary research goal for the future is to support reimplementations, in a follow-on project called Transformation-Based Maintenance. This support will take three forms: strengthening the formal representation of developments (especially the rationale behind each step), supporting the evolution of specifications through the same mechanisms that support maintenance (and in fact integrating the two processes), and developing a formal language of modification for both specifications and developments.

Interlisp. This project is creating a self-sustaining support center for Interlisp. The goal of the project is to provide large-address-space portable versions of Interlisp for the VAX and for other hardware architectures. Currently, ISI-Interlisp is implemented on the VAX and runs under the VMS and UNIX operating systems. In addition to ongoing maintenance tasks, development efforts are planned for the future, including the porting of Interlisp to other machines.

Computer Research Support. The Computer Research Support project is responsible for providing reliable computing facilities on a 24-hour, 7-day schedule to the ARPANET research and development community. At the same time, the project makes available to ARPANET users the latest upgrades and revisions of hardware and software. The project provides continuous computer center supervision and operation, and a full-time customer-service staff that is responsive to user inquiries. This project supports three computer installations, the largest at ISI's main facility in Marina del Rey. The other supported facilities are at the Naval Ocean Systems Center (NOSC) in San Diego and at Gunter Air Force Base. The Computer Research Support project provides support in four interrelated, though distinct, areas: Hardware, System Software, Operations, and Network Services.

Strategic C3 System Experiment Support. DARPA has defined an experiment in Strategic C3 systems to be conducted in cooperation with the World Wide Military Command Control System (WWMCCS) System Engineer (WSE) and the Strategic Air Command (SAC). The concept of the experiment is to demonstrate and evaluate the use of new technologies (such as the ARPANET, packet radio, network security, and distributed knowledge-base techniques) for strategic command, control, and communication. The DARPA experiment is defined as having three phases. Phase I is planned to demonstrate air-to-surface packet radio links and gateways into the ARPANET as a first step in evaluating the feasibility of a truly survivable strategic network. Phase II is directed toward creating a survivable message system and data bases through multiple copies of the critical components and data across the ARPANET. Phase III will address the feasibility of rapid reconstitution of a strategic network by deployment of packet radio networks to reconnect surviving

elements of the network. ISI's major portion of the above plan is to provide an initial core of necessary facilities (ARPANET/MILNET access, host systems, various software tools, Network Services support, etc.) to allow SAC personnel to gain experience with this technology and to ensure the success of the experiment. Installation of modems, installation of 30 or more interactive CRT terminals, user training, system software training and support, and on–site maintenance of equipment are part of the continuing program.

TCP/IP Implementation Support. The Department of Defense has adopted the Internet concept, and the IP and TCP protocols in particular, as DoD–wide standards for all DoD packet networks. The DoD will be converting to this architecture over the next several years. The role of the TCP/IP Implementation Support project is to assist in placing these protocols into active use in the operational ARPANET and MILNET by installing and debugging host software to support the ISI user community. This effort involves programmers and researchers from ISI, Bolt Beranek and Newman, Inc., Digital Equipment Corporation, UC Berkeley, SRI International, Stanford, the Massachusetts Institute of Technology, and other institutions.

Exportable Workstation Systems. The ISI Exportable Workstation Systems project has proposed a testbed workstation system for installation at the DARPA–IPTO office in Arlington. This workstation, the SMI SUN (a Motorola 68010–based system with graphics hardware and display), runs UNIX–based software. ISI will install fourteen workstations, which will be connected to a local Ethernet and a VAX file server. TOPS–20 systems will be maintained at ISI in Marina del Rey as remote servers and database hosts. This workstation system will provide a model for a distributed environment, and will serve as a test site for new hardware and software developed in DARPA–IPTO programs. The system will also serve as a proving ground for software developed by ISI and other DARPA–IPTO contractors.

New Computing Environment. The New Computing Environment project's goal is to adapt developing computer technologies to serve the research and military user communities. The resulting model computing environment will serve several purposes. It will provide a very large improvement in languages, system support, and additional computing cycles to the ongoing ISI research effort; it will serve as a testbed and eventual existence proof for the implemented technology; and it will serve as a proving ground for local computer environments targeted for DARPA and the military community, as well as a device for investigating new research ideas that will eventually be adapted by those communities. In addition, the small size, portability, and local computing capability of personal computers will allow for experimentation and realization of command and control requirements for an environment that can exist in mobile command centers.

1. CONSUL

Research Staff:
William Mark
Thomas Kaczmarek
Thomas Lipkis
William Swartout
David Wilczynski

Research Assistants:
Nitsan Har-Gil
Gabriel Robins
Richard Stokey

Support Staff:
Sharyn Brache
Kathie Patten

The Consul project is exploring the use of knowledge–based technology to allow a natural interface between users and computer services. We conduct our research in the framework of a demonstrable user interface environment that includes facilities for understanding natural language requests and producing natural language help and explanations. The system contains detailed models of users and services and a set of inference mechanisms to transform descriptions from one model into an appropriate description in the other. Consul research therefore focuses on representation and inference in the interactive systems domain.

1.1 PROBLEM BEING SOLVED

Although computers have become a part of the everyday office environment, their use has not expanded much beyond that of a typewriter with spreadsheets. In particular, most computer "users" still do not utilize the many interactive services that could help them do their jobs better. The reason for this is clear: it is still very difficult for the average user to get these services to do what he wants, especially if he has to combine more than one service to do his job.

If interactive services are to be useful to a wide audience, they must be accessible through an interface that makes them truly easy to use. We believe that this interface must *understand* what the user wants to do (or wants to know) and let him describe it in terms of menu selection, commands, or natural language—in whatever combination is most natural for him for that particular task. This accessibility has been designed into the Consul system. Once Consul understands the user's request, it either formulates the set of function calls that will do what the user wants or generates an English explanation response to tell him what he wants to know.

This interface must be constructed over a body of service functionality that is integrated at the semantic level. That is, the service functions and data structures must be written so that programs can be executed in any combination that makes semantic sense. This requires a careful model of desired service functionality and a mechanism for establishing the connections between the model and the actual data structures and code written by individual service builders.

1.2 GOALS AND APPROACH

Our research on the Consul project is directed toward producing a knowledge–based interface for the users of interactive office services such as electronic mail, appointment calendars, and document preparation. The interface includes natural language request understanding and explanation facilities to be used by a wide class of users ranging from novice to experienced.

Consul's interface is designed to be used in conjunction with CUE (see Chapter 2), which provides a menu/command–based interface that interacts with Consul's knowledge–based components. The interface is provided in a distributed environment, with Consul residing on a server machine that can be shared by one or more CUE workstations on the same Ethernet, as shown in Figure 1-1.

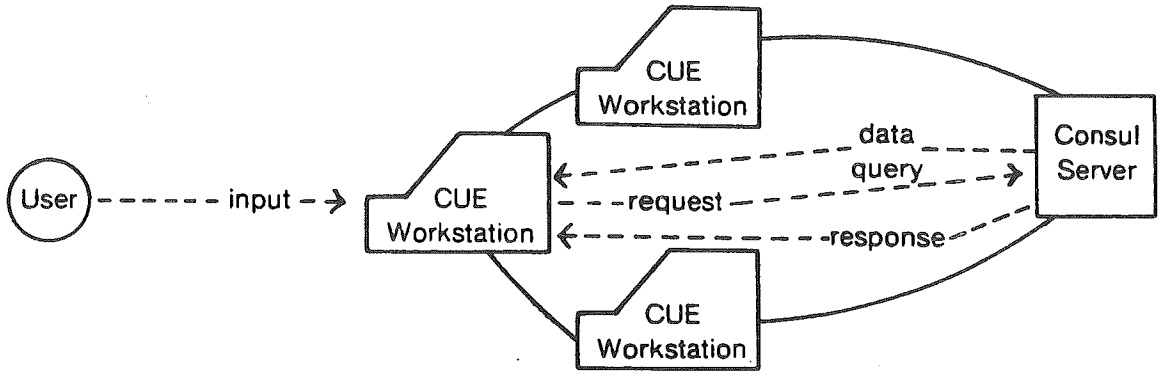


Figure 1-1: Consul/CUE System Configuration

Consul's approach is based on representation of the characteristics and behavior of users and services. The resulting knowledge base of user and service actions, objects, and events provides a foundation for the inferential activities that are necessary to map the user's description of his needs into the system's descriptions of service capabilities and requirements (and back again to the user's domain, to produce explanation responses). Using the knowledge base to provide interface facilities is a process of inferring the appropriate system behavior for a given input.

When a user produces input at a CUE workstation, the input can be a menu selection (via mouse), a typed–in command, or a typed–in natural language request. Any input that CUE cannot deal with in the workstation command processing environment is sent on to Consul as a request that requires knowledge–based processing. Consul expresses the request in terms of the system's knowledge representation and then uses inference to try to describe the representation as either a service operation to be executed or a natural language explanation response to be generated. This processing may require Consul to query databases on the CUE workstation about the details of data structures (e.g., to find out if there is some user named Jones, or if the logged–on user has scheduled a meeting for tomorrow on his calendar).

Consul is also examining the problem of knowledge acquisition in the interactive service domain. The approach here is to allow a service builder to describe a function or data structure to the system, and to use inference to determine how it fits into Consul's knowledge base (what Consul already knows about the user and system environment).

1.2.1 Knowledge Representation and Modeling

Consul's current knowledge base is implemented in the KL–ONE representation formalism [3]. Although KL–ONE is adequate for constructing our demonstration, its limitations in terms of efficiency and expressive power make it the wrong vehicle for future work. We are therefore currently involved (jointly with Bolt Beranek and Newman, Inc.) in a major new implementation of KL–ONE (NIKL), on which we will base our future work.

NIKL provides more consistent and compact data structures for representing knowledge, along with

NIKL provides more consistent and compact data structures for representing knowledge, along with a much-needed overhaul of the function call interface. Even more important is NIKL's improved expressive power, including the ability to represent disjunction (which aids certain aspects of semantic interpretation in the natural language system and allows all of the system's inference mechanisms to reduce their search times), a clean representation of roles (which aids in the process of concept definition and improves the efficiency of the classifier), and a clean interface to the assertional mechanism (which allows consistent usage of the knowledge base by external reasoning processes). Some of the Consul system has already been brought up in NIKL; conversion of the remaining part is underway.

A good representational foundation is extremely important for the knowledge base, but it does not address the crucial question of what should be in the knowledge base. Our modeling efforts maintain a distinction between the ***user model***, a consistent description of a "user's view" of the system, and the ***systems model***, a representation of the operations found in interactive systems along with the data structures these operations work on. The systems model is further divided into two categories: general knowledge of interactive computing that is pre-built in the Consul knowledge base, and specific knowledge of the particular operations and data structures of each service that is added by the acquisition mechanism.

These distinctions are important for the inferential processes that use the knowledge base to produce the behavior of the system. For example, the user model provides the target for the natural language system (anything the user says must be translated into some user model concepts) and the filter for producing explanations (anything said to the user must be in terms of user model concepts). The systems model similarly provides a filter for the acquisition process, allowing it to produce valid service-specific descriptions as its output.

We have recently begun to tie together these various aspects of the model to form a "safety net" that will be able to catch virtually any user input in order to allow further processing. For example, if the user says:

Get this message to Jones.

Consul can recognize the input as a request for sending, because the parser will interpret it as a request for the user model concept of "information transfer."[1] "Information transfer" on an object that can be seen as a "message" can then be recognized as the system's concept of "send."

1.2.2 Inference

Consul's reasoning activity is based on the fundamental inference activities of recognition and redescription. Recognition is the process of determining how each piece of incoming information fits into the system's current knowledge. Redescription is the process of determining how each recognized piece of knowledge can be viewed as applicable to the system's problem-solving goals.

Recognition consists of **classification**, finding the appropriate place for each new piece of information in Consul's taxonomy of knowledge, and **realization**, finding the real-world objects described by each new piece of information. Classification involves the determination of the relationship between the terms of a new description and the terms already known to Consul. For example, a concept like "Smith's reply" must be related to known terms, such as "structured data,"

---

[1] The safety net structure of the model is reflected in the semantic interpretation of natural language as well—see Section 1.2.3.

"message," "user," and "Smith." Realization is the association of each new term with the real-world objects it describes. For example, "Smith's reply" must be seen to describe a particular real-world entity if Consul is to be able to determine useful information like "Smith's reply is the same message as Message 17."

The classification process is currently being reimplemented in NIKL to take advantage of the new representation technology and the new, more efficient data structures. Preliminary tests indicate that it will be several times faster than the existing Consul classifier. The realization mechanism is also being redesigned in order to take advantage of NIKL's clean interface to the structure of assertions that represents the real world.

Consul's redescription mechanism is also being revamped to take advantage of NIKL and of the new structure of the model. Rules in Consul are still represented in the knowledge base as ordinary concept structures consisting of a condition and conclusion linked by asserted correspondences between parts. Interpretation of the rule is, however, somewhat different in the new scheme.

When the description of an entity instantiates the condition of a rule (as determined by classification), the redescription mechanism knows that, under certain conditions, that description can be reformulated according to the conclusion of the rule. These conditions express necessary redescribability constraints between the rule's condition and conclusion. For example, a rule might state, "a user reply can be redescribed as the creation of a system message whose 'replier' is the sender of the new message and was an addressee of the original message."[2] The redescription mechanism must then check to see if the 'replier' can be redescribed as a valid sender, and so on—a recursive process that terminates with known primitive redescription relationships (e.g., any of the known ways to describe a person on the system—name, nickname, system ID, etc.—can be interchanged). This redescription process continues until a description is created that corresponds to one of Consul's goal behaviors: responding to a request via function execution or explanation.

1.2.3 Handling Natural Requests

A major Consul accomplishment has been the demonstration of the advantages of a workstation environment that allows natural language input and output to be integrated into a state-of-the-art, window-based menu and command interface. Consul has demonstrated an interface that includes understanding of natural English requests (e.g., "Forward this message to everybody in my 3:30 meeting."), natural English responses to questions about the system (e.g., "What has to be in a forwarded message?" leading to the answer "A forwarded message must have an addressee and a message to be forwarded."), and natural English error responses (e.g., "You can't forward a draft message, you can release it or send it for review.").

The natural language features of the interface make it easy to use, easy to learn, and nonthreatening; provide natural access to problem-specific help information; and give users easy access to functionality across applications.

The natural language understanding system is based on a large coverage English parser [1] and a first-level semantic system [2], both originally developed at Bolt Beranek and Newman, Inc. We have interfaced the parser to the Consul knowledge base and extended and adapted the semantic system to our workstation environment.

---

[2] The actual rule is much more detailed.

The semantic system is structured in terms of ***phrasal frames***. Each frame is a pattern for distinguishing a semantically distinct phrase. Semantic interpretation consists of matching the input sentence to the appropriate phrasal frames and producing the meaning representation from the matched frames. These frames are organized into a safety net to support the structure of the model. Natural language input that does not fit directly into an interpretation in the user model is "caught" in the phrasal frame safety net for special processing. Thus,

Get the specs to Jones.

can be interpreted as some kind of request for sending, even if the system does not know what "specs" means, because the phrasal frame safety net is prepared to accept a more general noun phrase in what is usually the "message" position. Consul's response would be to put up a message-sending form with "To: Jones" filled in, along with the explanation "I don't know what you're trying to send...Could you move it into the message window?" In addition, a new semantic interpretation algorithm, MIFIKL [10, 13], is being implemented to efficiently access this greatly enlarged body of frames.

We are also exploring techniques for robust natural language interaction based on the ideas of relaxations coded in meta-rules [5, 9]. These rules apply when normal interpretation of input fails. Each rule identifies a type of "error" and the relaxation necessary for complete interpretation of the phrase. This work has been implemented for ungrammatical input with respect to our current grammar system.

1.2.4 Acquiring Services

Execution and explanation of individual service capabilities require detailed models of services. It would be virtually impossible for us to build in all of these models by hand. It would also be impossible for the service programmer to build them by hand, since he is presumably unfamiliar with our model and, indeed, the entire modeling process. It is therefore essential that Consul provide a computer-aided methodology for incorporating detailed models of services into the overall system.

The acquisition procedure receives new services into Consul by asking the service programmer to describe the relevant features of the functions and data he is implementing. "Relevant" features are those that, according to Consul's systems model, affect the user's view of the system. The service programmer's descriptions are in terms of ***process scripts***, a formalism especially designed to provide an adequate descriptive mechanism for allowing the automatic acquisition procedure to examine the system's knowledge base and make appropriate connections to the service programs and data. The acquisition mechanism therefore consists of a structured dialogue between the system and the service programmer that results in forming a relationship between Consul's general model of a given function or data structure and its implementation in terms of the function or data structure the service programmer is actually coding.

The first acquisition scenarios were always in terms of process atoms, process scripts with only a single call to application code. The acquisition of more complex process scripts was an open issue; acquisition was never intended to read and analyze arbitrary process script code. Now we are writing process scripts that establish an implementation for the functions represented in the knowledge base. Acquisition will direct the service builder in specializing these scripts. The process script is constructed so that its instantiation produces a specialization of the KL-ONE model as well. The process script thus becomes a bridge between the KL-ONE models and the CUE execution environment.

As an example, consider the following abstract process script that deletes a reference to a generic object from its directory:

```
Process Script: DeleteObjectReference;
Input: o:ObjectReference
Body:
    (Edit o ChangeHilite Deleted)
  * (MarkObjectDeletedAtom o)
    (IF (m ← (Openp (ObjectFromDirectoryReference o)))
        then (CloseWindow (GetWindow m)))
```

This process script, which is attached to the KL-ONE model of "directory item deletion," is a skeleton to be instantiated for real objects like messages. The functions Edit, Openp, CloseWindow, and GetWindow are system utilities for use with any data structure in the CUE environment. ObjectFromDirectoryReference is applicable only to objects defined to be a kind of ObjectReference. The only code the service builder must write when he specializes this process script is MarkObjectDeletedAtom—the "*" indicates the need for a service atom.

1.3 TECHNICAL ISSUES

The primary problem for Consul research is deciding how to choose, express, and implement the many knowledge base concepts necessary to model the events, actions, and objects of the interactive computing world. Deciding what to model is a matter of deciding both what to include in the knowledge base and what to say about it. This is essentially the problem of describing something so that someone else (or in this case, *something* else—Consul's inference mechanisms) can understand it. We are gaining experience with this process and have made considerable progress, thanks in great part to NIKL's greater expressive power and its cleaner split between the descriptive and assertional worlds.

Once the choice of concept description is made, the concept must be expressed in a consistent, general formalism that allows automatic recognition and rule-based reformulation procedures. This is a problem of epistemology applied to the domain of mechanized inference. Our work, in combination with that of our colleagues [4, 7], has been directed toward achieving clarity and expressive power in our representation language through careful preservation of the epistemological structure inherent in the knowledge we are trying to represent. This work has resulted in KL-ONE, NIKL, and related formalisms; we see it as an area of continuing research interest and importance.

Finally, the representation formalism must be implemented efficiently on the machine so that the knowledge base can be quickly searched, cleanly interfaced to external databases and procedures, and easily maintained. Actually using the knowledge base for Consul activities requires efficient inference and search procedures. We need algorithms for rapidly determining the relationships among terms in the formalism, using terms to make statements about the world, and using inferential meta-knowledge to pursue the problem-solving goals of our system. The classification, realization, and redescription mechanisms probably represent the state of the art in this area. Nonetheless, they are only a first-cut solution to the overall problem. We are currently investigating new algorithms based on our latest thinking about the representation formalism and on the probable future availability of parallel computation methods.

The knowledge base must also be interfaced to the outside world—not only to provide input/output, but also to allow connection to information that is part of the system but not represented in the knowledge base. Explicitly modeling information in the knowledge base increases the system's capability, but is costly in terms of execution speed and memory requirements. Choosing to represent information outside of the knowledge base can be more efficient (and is in fact necessary in some environments), but brings up issues of what the system does and does not know about the information it contains. We are currently investigating ways to connect the knowledge base with the rest of the system so that it provides clean semantic distinctions to guide Consul's various dealings with the information it contains.

Building and maintaining the knowledge base is a matter of providing the appropriate tools for creating, examining, and changing knowledge structures. We have produced useful tools for each of these activities, but we still have much to do in order to make our entry and maintenance procedures smooth and efficient.

The Consul project is also investigating technical issues not directly related to the knowledge base: how to present information on the display screen so that the user can understand it easily, and how to build service software with sufficient flexibility and consistency to meet the demands of responding to requests that are natural to the user. We have been experimenting with various designs for the user's screen interface, which must wait for use by real users for any sort of validation. In terms of service software design, we have been concentrating on the abstract datatype formalism as a mechanism for enforcing consistent programming practice to meet Consul's overall service design requirements.

1.4 SCIENTIFIC PROGRESS

- We have implemented a combined Consul/CUE demonstration interface that runs as a distributed system across the Ethernet. The system is written in Interlisp and has been demonstrated on all machines in the Xerox 1100 series. It is a demonstration vehicle that works only on a limited number of test cases. The system consists of

  - a KL-ONE knowledge base of about 1000 concepts;
  - the natural language understanding system;
  - a first-cut explanation and language generation facility;
  - a real-time interface to the CUE workstation environment.

  In order to receive feedback on our interface design from a wide variety of experts and potential users, we have provided numerous demonstrations of our combined Consul/CUE system.
- We are cooperating with Bolt Beranek and Newman, Inc., in the KL-ONE/NIKL knowledge representation effort.
- We are continuing our cooperation in basic knowledge representation research with groups at Fairchild Research, Xerox Palo Alto Research Center, and Bolt Beranek and Newman, Inc., and are conducting a knowledge representation workshop this fall to provide a forum for the discussion of recent progress in knowledge representation research.
- Our publications and presentations include the following:

  - 1983 International Joint Conference on Artificial Intelligence, Karlsruhe, Germany:

    - presentations and demonstrations of the Consul/CUE interface;
    - publication of a paper on the KL-ONE classifier [8].

- Presentations and demonstrations of the Consul/CUE interface at the National Artificial Intelligence Conference, Washington, D.C.
- A tutorial on natural language interfaces at the ACL Applied Natural Language Conference, Santa Monica, California [11].
- Presentation at the Workshop on Automated Explanation Production, Idyllwild, California [6].

- We have designed and are implementing a major new semantic interpretation system (MIFIKL) for natural language understanding in the NIKL environment [10, 13].
- We are continuing our investigation of parallel control structures for our knowledge base algorithms, specifically in the context of MIT's Connection Machine architecture [12].

1.5 IMPACT

Personal workstations are emerging as one of the most significant tools available to the office environment. The commercially developed workstations now entering the workplace present a standard for the development of workstation software: workstations must provide an excellent user interface to an integrated set of application programs. However, the software technology of current workstation offerings has fundamental limitations:

- The interface can be provided only in the limited framework of predetermined (vendor-provided) application programs. When the system is extended to new applications, only a much lower level of integration and user interface uniformity can be provided.
- The interface is inherently limited in terms of flexibility and naturalness.
- It will be extremely difficult to extend these systems to take advantage of advances in knowledge-based processing technology (e.g., speech, expert systems).

Consul shows a way to progress beyond these limitations, an approach in which the machine has knowledge both of what the user is trying to do with the workstation, and of what the workstation can do for the user. It allows the development of the workstation software to be controlled so that application programs—and their data—interact whenever it makes *semantic* sense to do so (not just in the cases that were "wired in" by the original programming team). The use of semantic models also allows the workstation user to interact with the system at the semantic level, making requests and receiving help responses in everyday English. In fact, the model can provide access to inferential routines that allow the system to react usefully to requests and commands that are ill-formed from the standpoint of absolute system requirements. Finally, the semantic models provide a basis for extension of the software into the realm of speech understanding and expert system technology.

User interfaces based on natural communication open up computer systems to a much wider audience—eventually to anyone with a need for their services. The natural interface accepts requests from users with absolutely no previous training with the system. It can explain how to use the system, what the system did, or what has gone wrong in terms that the user can understand and learn from. This kind of interaction is virtually a necessity for acceptance by users in many aspects of society where the need for computer service is greatest. Natural user interface technology is fundamental to continued progress in the delivery of computer services to society at large.

1.6 FUTURE WORK

Our major goal is to use the combined Consul/CUE approach to produce an interface that is efficient enough (in terms of both user and system execution time) to be useful with current technology and is robust enough to be useful in an actual user environment.

Efficiency will come from a thoroughgoing use of our distributed approach to interface management. By concentrating the high-frequency window, menu, and command-oriented operations on the user's own workstation, and by providing smooth access to the knowledge-based server for the rarer requests requiring this service, we will achieve a total system that is efficient enough for real-time use with currently available technology.

Robustness in the Consul system will mean that, where possible, the system will provide the service execution or explanation response that the user has requested. Where this is not possible, the response generated will always be a useful suggestion of a way the user can accomplish his intent.

For the natural language system, this robustness requires the construction of a large set of phrasal frames. We have already begun to build the phrasal frames needed to handle a wide variety of message transmission requests. In the future, studies will be undertaken on filing messages, updating calendars, querying the contents of directories of files, etc. Because of the interrelationships, each completed study will ease the next domain study. For example, our phrasal frames for message descriptions used in the transmission task will largely serve for the filing task. Further, we have begun to identify classes of activities and semantic phenomena that can be allowed for automatically or can be easily added to phrasal frames in new domains. Example results include a tool for machine-assisted lexical acquisition and a fairly general treatment of time modification. The product of this research will therefore be not only a large set of phrasal frames, but also tools and guidance on developing these structures.

For the rest of the Consul system, increased efficiency and robustness require better inference mechanisms and more complete models of users and systems. We are therefore deeply involved in inference and modeling research, expecting to provide efficient inference on a knowledge base an order of magnitude larger than our current one (i.e., on the order of 10,000 concepts). This increase in inference efficiency and model size and sophistication in turn depends on continuing improvements in our underlying knowledge representation formalism and the tools we have developed to use it. Through our work on NIKL, we have already begun the process of improving our representation technology; we plan to push forward in this area, in cooperation with others in the AI community.

REFERENCES

1. Bobrow, R. J., "The RUS System," in B. L. Webber and R. J. Bobrow (eds.), *Research in Natural Language Understanding*, Bolt Beranek and Newman, Inc., Cambridge, MA, 1978. BBN Technical Report 3878.

2. Bobrow, R. J., and B. L. Webber, "Knowledge representation for syntactic/semantic processing," in *Proceedings of the National Conference on Artificial Intelligence*, AAAI, August 1980.

3. Brachman, R., *A Structural Paradigm for Representing Knowledge*, Bolt Beranek and Newman, Inc., Technical Report, 1978.

4. Brachman, R., R. Fikes, and H. Levesque, "KRYPTON: A functional approach to knowledge representation," *IEEE Computer*, September 1983.

5. Kwasny, S. C., and N. K. Sondheimer, "Relaxation techniques for parsing grammatically ill–formed input in natural language understanding systems," *American Journal of Computational Linguistics* 7, (2), 1981, 99–108.

6. Lipkis, T., "Descriptive mapping for explanation production [Abstract]," *SIGART Newsletter*, (85), 1983.

7. Moser, M. G., "An overview of NIKL, the New Implementation of KL–ONE," in *Research in Natural Language Understanding*, Bolt Beranek and Newman, Inc., Cambridge, MA, 1983. BBN Technical Report 5421.

8. Schmolze, J., and T. Lipkis, "Classification in the KL–ONE knowledge representation system," in *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, IJCAI, 1983.

9. Sondheimer, N. K., and R. M. Weischedel, "A rule–based approach to ill–formed input," in *Proceedings of the Eighth International Conference on Computational Linguistics*, pp. 46–53, International Committee on Computational Linguistics, October 1980. Tokyo.

10. Sondheimer, N. K., and R. M. Weischedel, *Consul Note 22: "Towards Semantic Processing with Phrasal Frames Using Structured Inheritance Networks"*, USC/Information Sciences Institute, 1983.

11. N. K. Sondheimer (ed.), *Tutorial on Natural Language Interfaces,* Association for Computational Linguistics, 1983.

12. Stokey, R., *Consul Note 21: "Implementation of a KL–ONE Network on the Connection Machine"*, USC/Information Sciences Institute, 1983.

13. Weischedel, R. M., and N. K. Sondheimer, *Consul Note 23: "Relaxing Constraints in MIFIKL"*, USC/Information Sciences Institute, 1983.

2. CUE

Research Staff:
Thomas S. Kaczmarek

Research Assistants:
Haym Hirsh
Gabriel Robins

Support Staff:
Kathie Patten

2.1 PROBLEM BEING SOLVED

The goal of the CUE project is to provide an extensible environment for building and using integrated interactive computer services. In CUE, there are no boundaries between, say, the electronic mail service and the automatic calendar service. Consider the following task:

sending a message to everyone in my 3:30 meeting telling them I can't make it

Accomplishing this task requires a common understanding of the services involved: essentially, one must know how to find a particular meeting in the calendar system framework, access its attendees, and convert them to valid addressees in the message system framework. Unless the basis for such understanding has been designed into the system from the very beginning, automatic interservice interaction at this level is impossible. In current systems, almost all of which consist of totally separate service programs or "subsystems," the necessary interservice understanding is completely outside of the design framework. If a user wants to perform a task like the one above, he must interact with each service program separately, doing his own data conversions (usually via a text editor). In short, given current software environments, the problems of such a priori program planning and implementation to allow for interservice interaction are intractable. Tasks like the one above cannot be performed without a great deal of painful user involvement.

Automatic interservice interaction requires an environment in which individual services can make assumptions about the properties and behavior of the other services in the system. These assumptions must relate to data structures, functional capability, and state information. The objective of the CUE project is to produce such a consistent underlying environment (CUE), in which users can interact with the system without regard to service boundaries.

2.2 GOALS AND APPROACH

The key element of the CUE strategy is to have a semantic model of how each program element (function or data structure) fits in with the other program elements in the system. That is, when a "send message" function is defined, CUE must have a model of how the elements of a message (e.g., the "user" in the addressee field) are already being used in the system. It must also know that "sending" involves transmitting information from one place to another, that the information must be made accessible at the destination, and so on. CUE can then use this knowledge to ensure that each new program element fits in appropriately with, and can be used as a part of, overall system functionality. CUE must also act as an execution environment which supports the sharing (or coercion) of data structures and the uniformity of function invocation necessary for interservice interaction.

To take full advantage of CUE's underlying consistency, the user interface must facilitate two activities: (1) sharing data from various services, and (2) combining functions from various services. All interaction with CUE will be through a multi-windowed full-screen editor. Such an interface is ideal for encouraging data sharing, because it allows the user (through movement of a graphic cursor) to select any displayed information and to insert a copy of it at any position on the display. Functional combination will be encouraged by allowing arbitrary functions to be applied to any selected region of a window and by supporting a functional style of computing in the command language.

2.2.1 Research Issues

The CUE project must address the following research issues:

- specifying a common data definition substrate;
- representing the relationships among structures and functions of different services;
- defining the user interface;
- achieving the correct level of modularity (grain size) for service functions;
- finding the correct level of interaction with a knowledge-based user interface (the Consul system); and
- providing transportability to maximize the impact of CUE on personal workstations.

2.2.2 Approaches to the Solution

Although there are currently no adequate solutions to these problems, we believe that there are fruitful approaches for all of them, based in part on complementary research at ISI and elsewhere.

2.2.2.1 Data definition

The problem of common data definition is well known in computer science; several kinds of solutions have been or are being investigated (e.g., [4, 12]). One of these techniques or a combination of them must be adapted by CUE to its service definition formalism. Work on abstract data types [2] will have a very strong influence on CUE. This technique is appealing because of its compatibility with the requirements of knowledge-based modeling and because it provides a well-defined and consistent view of data structures.

2.2.2.2 Modeling service relationships

A few current systems (e.g., UNIX) contain primitive mechanisms for combining the functional elements and data structures of different interactive services. However, there is never a semantic model of the relationships between different service entities. Thus, the systems have no representation of the basis for commonality between, say, a message service and a calendar service. Existing systems can therefore never go very far toward ensuring integration: they must rely on each service builder to think of the necessary interrelationships among his service and others, and to design in enough flexibility to handle any interactions that might occur.

CUE will avoid this total reliance on the service builder by providing a semantic model that links the functions and data structures of one service to those of others. CUE will provide an acquisition mechanism that associates the program elements of new services with the constructs of this model. Research on knowledge representation and acquisition in the Consul project provides already well-developed techniques for modeling the characteristics of interactive systems [7, 11]. The CUE

project will use and extend Consul's model to provide the independent semantic description that allows the system to relate the data and functions of different services.

2.2.2.3 The user interface

CUE's user interface will focus on techniques which facilitate interservice sharing, promote a consistent user view of the system, and guide the user through interactions. A number of existing techniques and extensions to them will influence the design of the user interface.

Multi-windowed full-screen editors are a proven technique for coordinating and sharing information between files. The benefits of supplying an omnipresent editor were demonstrated in the SIGMA message system [10]. The extension of this approach to a multi-service environment will encourage a consistent user view by providing a uniform mechanism for the examination and modification of data. A functional style of computing [1] will be used to further encourage the combination of functions from different services.

Extensive use of forms and menus will be used to help the user specify his task. The value of such an approach has been demonstrated in commercial systems like the Xerox Star as well as in research efforts such as COUSIN [3] and MENUNIX [8]. The main difference between the CUE approach and that of COUSIN or MENUNIX is the integration of services and the resulting pervasiveness of the user interface. All data, not just command lines, will be handled through the same user interface in CUE. Furthermore, while the physical appearances of the CUE interface and the COUSIN interface may resemble each other, they are generated by very different processes. In CUE, after acquisition of a function, the user interface is automatically updated to make that function available and to make the invocation of it consistent with similar functions.

CUE offers multiple levels of help. Menus and forms are a first level, because they make it clear to the user what functionality is available. User inquiries about valid parameterizations of functions will also be handled through menus. For example, if the user wants to delete a file, the request for valid parameterizations will list all of the files he can delete. Explanation text will be an integral part of the user interface. Since CUE and Consul use the same knowledge base, CUE can use Consul's explanation facility to generate this text for service constructs automatically. In addition, an integral part of the CUE project is provision for communication between the CUE and Consul systems, thus giving the CUE user access to Consul's natural language help and explanation facility for very specific help requests.

CUE research on the user interface must also include experiments that investigate the most user-efficient ways of using bit-mapped graphics, electronic pointing devices, speech, and other state-of-the-art interaction media. The CUE user interface will be designed to be very tailorable and device independent. The user will be given great flexibility in modifying the interaction protocols, allowing experiments with different methods to be conducted and providing the user with modifiability to meet individual needs and preferences. Device independence will be supported with multiple input channels available to the user at all times. The user will decide which input device is most convenient and simply use it. Mixed-media input will be supported, so that parts of a command may be spoken while other parts may be typed and still other parts pointed to on the screen. Multi-media interfaces are a new development, and it will take some amount of experience to learn how to best use them.

2.2.2.4 Grain size

Ensuring the appropriate functional grain size for each service requires checking with respect to the semantic model and with respect to the other functions in the system. CUE will use the model to ensure that the size of each new function makes *semantic* sense (i.e., that the function works on the right kind of objects—for example, that a function purporting to be a "send" puts message data structures into mailbox data structures). In addition, CUE will use its knowledge of the other functions in the system to ensure that each new function makes *structural* sense (e.g., that there are other functions for accessing the mailbox data structure and for composing the messages).

2.2.2.5 Consul interaction

CUE's user interface is designed to interact with Consul to provide natural language request handling and to support the generation of specific help for the user's problems. Practical application of Consul is at least several years off; however, CUE is tied into Consul research, and when the technology becomes practical CUE will offer it to the user. When Consul comes on line, the CUE user will be able to type sentences such as, "Allow anyone to read this file," or "How can I recover from this error?" CUE will notice that these are not command requests, realize that they might be natural language requests, and pass them to Consul. Consul will be able to process sentences such as these only if it has a model of the current environment. CUE must supply Consul with enough information to build this model. The CUE and Consul projects must together determine how much of the state should be modeled in Consul and what mechanism Consul will use to reference or access information that is not explicitly modeled. The projects must also determine an appropriate formalism for efficient communication at two levels: knowledge base operations and procedure invocation (inference procedures in Consul and service procedures in CUE). From the operating system viewpoint, these are both presumably interprocess communications that will have to be covered by the appropriate protocols (see [9]).

2.3 SCIENTIFIC PROGRESS

The CUE project has decided to rely heavily on the formalism of abstract data types. This formalism provides the basis for the common data definition substrate as well as the modeling of data structures. Furthermore, reliance on abstract data types provides some of the answers with respect to the user interface, because they provide a model of data that is an abstraction of the way users think about it. Combined with generic functions, abstract data types simplify both the modeling in Consul and the communication required between CUE and Consul. Finally, abstract data types and interprocess communication combine to give CUE a methodology for both integrating existing services into the CUE environment and distributing functionality across a network.

The strategy of using abstract data types as the basis for modeling, data definition, and the user interface has been tested by generating a demonstration system. The demonstration system portrays the interaction between calendar and mail services. The execution environment of the demonstration has been built by hand, using the principles that CUE is seeking to automate. This effort has verified the soundness of the approach.

The demonstration system has also been useful in terms of gaining experience with different strategies in the graphic characteristics of the user interface. As a result of this experience, the CUE project has defined a generalized forms package and a strategy for employing it in the system. All information in CUE will be presented via a generalized form. The following generalizations are made in the CUE forms package:

- Forms are hierarchically organized—the fields of a form are generally other forms, not simply text strings.
- Classes of forms can be defined to encourage consistency across forms and to simplify their definition.
- Form generation and placement protocols are user modifiable.
- Forms define the interaction between the user and the form as well as the graphic attributes of the form.
- Forms may have functions attached to them which perform actions such as testing data, controlling the flow of the interaction, and computing values.
- Forms may use multi-media communications.

The demonstration system also gave us an opportunity to experiment with CUE/Consul interaction. CUE and Consul were separated so that they ran on different processors. A strategy was developed for deciding which information CUE must supply to Consul to maintain its record of an interactive session. We discovered that the granularity of the information exchange between CUE and Consul could be quite coarse, since in most cases Consul did not need to know the details about the data being manipulated. Consul simply needed to know that the data existed and that some operation had been performed on it. This discovery is important because it allows Consul to do "less modeling" in the sense that it does not have to build an explicit semantic model of all the objects in the user's world.

The publications and presentations of the CUE project include the following:

- SoftFair '83 [5];
- demonstration of CUE/Consul at IJCAI '83;
- demonstration of CUE/Consul at AAAI 1983.

2.4 IMPACT

Many current applications are underutilized because they are not well integrated with other systems with which they naturally interact. Recent successes in integrating just three services (spread sheets, graphics, and data bases)[1] have demonstrated the impact that integration can have. However, integration beyond a few services is impractical without a methodology to oversee the installation of new services into the environment. The knowledge-based approach of CUE can provide the basis for enforcing integration.

CUE can also affect two of the major problems faced in the development of applications:

- designers would like to minimize inconsistencies in the interface as the user moves from one application to another, and
- roughly 60 percent of current development efforts are committed to the generation of the user interface.

With the advent of more sophisticated interactive devices, the time spent on the user interface and the inconsistencies found across applications will only increase. A methodology is needed to automate much of the responsibility for the generation of the user interface. This problem can also be solved by CUE's knowledge-based approach.

---

[1] The commercially available systems, Context MBA and Lotus 1-2-3.

Personal workstations are the way of the future, and CUE can influence the development of this important technology in the short term. Personal workstations will support significant local processing to avoid delays seen in current time-sharing systems, and will use bit-mapped graphics to improve the presentation of information to the user. The hardware for these workstations is now coming on the market from several sources. However, the software available on current workstations, while advanced in some respects, is inadequate for the long term because of its user interface and lack of support for service integration.

Software which intelligently uses the advanced graphics capabilities of current workstations, provides a consistent and understandable interface, and allows interservice interaction (i.e., data sharing and functional cooperation), must be developed to prevent the entrenchment of the current software in this new and important computing environment. The consequence of not attending to these problems is a system which will only exacerbate the problems now faced. As part of the CUE research plan, we will produce such software for use within ISI and for export to external user communities.

2.5 FUTURE WORK

CUE will develop and deliver a working system in three years. This system will include several services (e.g., document preparation, mail, calendar, data analysis), integrated into the CUE user interface. The user interface will support multiple forms (windows) and an all-present full-screen editor for these forms. A CUE workstation will be able to communicate with Consul for handling natural language requests. An important part of the working system will be a set of tools that allow new services to be added to the system. While CUE is a single system, it has two aspects to it: the service building activity and the service execution activity.

2.5.1 Service Building

Service building is an off-line activity. The service builder will use the Process Script language [6] to describe his service. Process Scripts are not intended to be used to actually write service functions; rather, they are to be used to declare information about functions and to configure calls to service functions. The executable code for the service functions may be written in any language, but the service builder must describe the data as abstract data types using the Process Script data definition capabilities (i.e., identify the structure and operators for the data) and must wrap the operators in Process Scripts. By using interprocess communication, CUE will be able to operate in an environment that allows service constructs to reside in independent processes or even independent machines.

After describing a service using Process Scripts, the service builder will introduce it to the system through an acquisition dialog, which will build a model of the service construct in the KL-ONE knowledge base and add it to the execution environment. Adding a new service to the environment will cause the new functions to appear in the correct menus, add command table entries for the functions, and generate explanation text for new constructs and include these in the run-time support.

The following major tasks must be accomplished to support service building:

- updating the definition of the Process Script language to reflect current ideas about modeling service constructs, including the addition of a facility to define abstract data types; and
- designing and developing the acquisition process, including building the execution environment.

2.5.2 Service Execution

The major features of the service execution environment include an omnipresent full-screen editor, integrated services, and Consul access.

The following major tasks must be accomplished to provide this execution environment:

- implementing a forms package to describe the presentation of user data;
- implementing a full-screen editor to interact with the forms package to allow the entry and modification of data;
- implementing a Process Script interpreter including support to inform Consul about user actions and error handling capabilities;
- placing CUE on top of some existing operating system to make use of its file system, process mechanism, I/O drivers, etc.;
- modeling the execution environment including general service areas; and
- selecting, building, and acquiring several useful services.

REFERENCES

1. Backus, J., "Function-level computing," *IEEE Spectrum* 19, (8), August 1982, 22-28.

2. Dahl, O. J., E. W. Dijkstra, and C. A. R. Hoare, *A.P.I.C. Studies in Data Processing.* Volume 8: *Structured Programming,* Academic Press, 1972.

3. Hayes, P. J., "Cooperative command interaction through the COUSIN system," in *International Conference on Man/Machine Systems*, Manchester, July 1982.

4. NASA, *IPAD: Integrated Programs For Aerospace-Vehicle Design*, NASA, 1980.

5. Kaczmarek, T., W. Mark, and D. Wilczynski, "The CUE Project," in *Proceedings of SoftFair*, July 1983.

6. Lingard, R., "A software methodology for building interactive tools," in *Proceedings of the Fifth International Conference on Software Engineering*, 1981.

7. Mark, W., "Representation and inference in the Consul system," in *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, IJCAI, 1981.

8. Perlman, G., *Two Papers in Cognitive Engineering: The Design of an Interface to a Programming System, and MENUNIX: A Menu-Based Interface to UNIX (User Manual)*, National Technical Information Service, AD/A108 929, November 1981.

9. Rashid, R., *An Inter-Process Communication Facility for UNIX*, Canegie-Mellon University, Technical Report, March 1980.

10. Stotz, R., et al., *SIGMA Final Report*, USC/Information Sciences Institute, RR-82-94 and RR-82-95, 1982.

11. Wilczynski, D., "Knowledge acquisition in the Consul system," in *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, IJCAI, 1981.

12. Yntema, D. B., *The Cambridge Project: Computer Methods for Analysis and Modeling of Complex Systems*, Rome Air Development Center, Technical Report, 1973.

3. MAPPINGS

Research Staff:
Robert M. Balzer
David S. Wile
Martin S. Feather
David J. Mostow

Support Staff:
Audree Beal

3.1 PROBLEM BEING SOLVED

Software specification, development, and maintenance continue to present an enormous problem to everyone involved with computers. We believe that the computer itself must play a far more significant role in the software development process than it does presently. The software designer's role should be streamlined to require only decision making and guidance, while the computer's is expanded to include manipulation, analysis, and documentation. The key to such support is to capture in the machine all crucial information about the processes of specification, design, implementation, and maintenance.

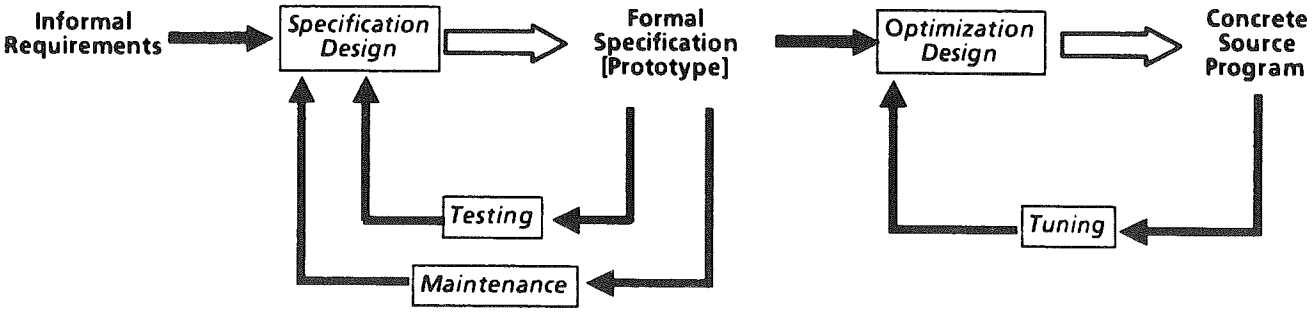


Figure 3-1: Alternative software development paradigm

For several years we have been developing an alternative software development paradigm based on this tenet [2].[1] We envision a future user developing a high-level specification of what he or she wants a program to do, and transforming the specification into an efficient program, using a catalog of (proven) correctness-preserving transformations (see Figure 3-1). Most debugging and all maintenance will be performed on the specification, which will have an operational interpretation suitable for testing. Reimplementation will be necessary to tune implementations exhibiting unacceptable performance characteristics.

---

[1] Several researchers led by Robert M. Balzer, working on projects funded by DARPA, NSF, and RADC, have provided the foundations for this paradigm.

This automation-based paradigm will rely on an integrated set of tools that directly support human developers in the processes of requirements specification, system design, implementation, and maintenance. It will be characterized by the fact that the entire evolution of a system—the history of all four of these processes, as directed by the developers—will occur and be recorded within the integrated environment. This history will provide the "corporate memory" of each system's evolution. The software development system will be an active participant in the development process, using the history to determine how these four processes interact: what assumptions they make about each other, what the rationale behind each evolution step was, how the developed system satisfies its requirements, and how to explain all of these to its developers.

We have developed a specification language, called Gist, which formalizes some of the expressive constructs commonly used in natural language specifications. These constructs include nondeterminism ("a message from another site"), descriptive reference ("the longest message"), historical reference ("the last message the user looked at"), constraints ("never send multiple copies of a message to the same person"), and demons ("if a week passes without a reply to a request, inform the sender"). A Gist specification describes the behavior of both the program and its environment; this provides a natural way to specify embedded programs that interact in complicated ways with their environment. Gist's expressive power makes it possible to specify ***what*** behavior a program should exhibit without saying ***how*** it should be implemented [4].

Three major problem areas require research and development before our transformation-based paradigm can succeed:

1. Tools, methods, and support facilities for ***acquiring***, ***designing***, and ***developing*** the specification consistent with a user's intent;
2. Tools, methods, and support facilities for ***implementing*** and ***optimizing*** the specification;
3. A framework for ***understanding***, ***reusing***, and ***maintaining*** previous specifications and optimizations.

The primary goal of the Mappings project is to capture program specification, development, and implementation knowledge in formal transformations: high-level editing commands, high-level transformations, simplification steps, and optimization strategies.

3.2 GOALS AND APPROACH

3.2.1 Overview

Over the years, the group has developed several tools to support the specification ***process***, i.e., to help bridge the gap between the informal intent inside the specifier's head and its formal expression in Gist. Specifications in any formal language tend to be difficult to understand, because they are usually concise and lack the redundancy found in natural language descriptions of the same behavior. Hence, we have designed a program that exposes static aspects of Gist specifications by paraphrasing them into English [14]. To expose dynamic implications and remote interactions of a specification, we have developed a symbolic evaluator that looks for interesting consequences of a Gist specification [5], and a program that explains the symbolic trace in English [15]. By clarifying what a specification really means, such tools help ***validate*** the specification, i.e., increase the specifiers' confidence that the specification matches their informal intent and the users' confidence that the resulting system will meet their needs. (For more information on the Gist paraphraser and the symbolic evaluator, see Chapter 11.)

We have discovered that when people specify programs to other people, they use an ***incremental*** technique for conveying the information necessary to develop the program. This technique is built on a very stylized set of methods, such as ***refinement*** of modeling detail, ***generalization*** of functionality, ***introduction of exception*** into idealized specification, and ***restriction*** of scope or functionality. In the future, our Mappings project will address the problem of capturing these methods as ***formal*** methods, thus permitting the specification development to appear much more natural to future readers of the specification. We have called these methods ***high-level editing*** commands, to emphasize the fact that they do indeed change the specification, yet at the same time convey more information to a reader than conventional keystroke editing commands. The use of high-level editing commands to design specifications is analogous to the use of transformations to develop implementations (see Figure 3-1).

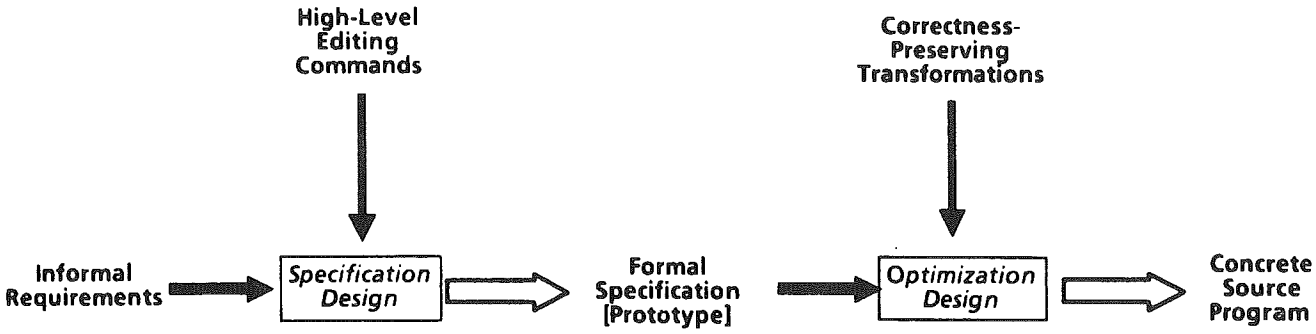


Figure 3-2: Design process mappings

Since its inception the Mappings project has addressed the problem of representing programming knowledge in terms of our specification language, Gist, by discovering correctness-preserving transformations for translating Gist's high-level constructs into the lower level constructs used in more conventional programming languages. Hence, the Mappings project addresses the second problem mentioned above: encapsulating programming knowledge in terms of concise mappings from Gist into alternative ***implementations***. In the future, more emphasis will be given to mappings for ***optimization*** of these implementations. (See Section 3.2.2.)

The Supervisory Control of Transformational Implementation Systems project (see Chapter 12) developed the system support needed to facilitate the automated implementation of specifications using transformations. It addressed the third problem mentioned above: the framework for describing the design activities of specification and implementation. This paradigm is based on capturing the ***processes*** of specification, design, implementation, and maintenance within the computer and supporting them via automated tools. We have developed an experimental system in which the user develops not only the implementation, but also a formal explanation of how it arose; the design decisions, assumptions, optimization steps, and their structural relationships are recorded in a formal design document suitable for automatic reuse by the transformation system (see Figure 3-2); using this ***formal development*** the system can replay optimizations on a changed specification to produce a new implementation automatically [16]. Naturally, all future mappings and transformations

will be designed to fit into this framework. A primary research goal for the future is to support *reimplementations*, in a follow–on project called Transformation–Based Maintenance.

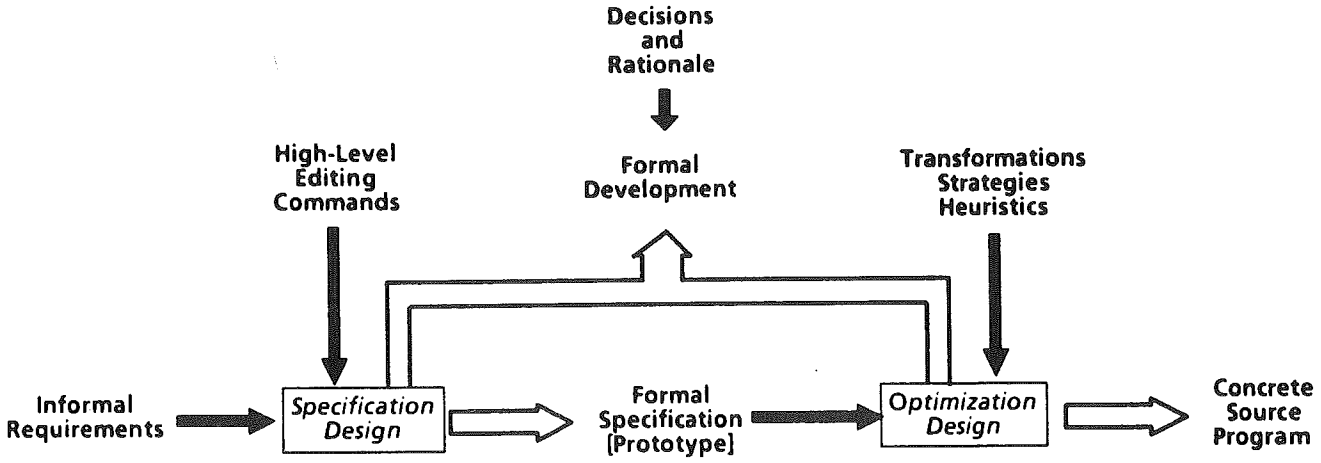


Figure 3-3: Formal developments

3.2.2 Mappings for Optimization

During the implementation phase, Gist's high–level constructs (nondeterminism, descriptive reference, historical reference, constraints, and demons) must be replaced by efficient realizations of these constructs. The Mappings project has concentrated on discovering and implementing the general transformations, or *mappings*, required. More precisely, for each construct, our research aims to accumulate the following:

- *implementation options* for converting an instance of the specification construct into a more efficient expression of the same behavior in terms of lower level constructs;
- *selection criteria* for deciding among these options;
- *mappings* to achieve the implementation options via sequences of correctness–preserving transformations.

We have also begun to implement a language called Will, an automatically compilable subset of Gist. (This subset was formed by omitting Gist's highest level constructs.) The compilation process is directed by user–supplied annotations to the program describing what implementations to use for specific data structures. We intend to use this language as the target language for our Gist mappings and as our high–level programming language for system development. We want to begin developing the system in itself, using the transformational programming paradigm. Although we have laid the groundwork for some of the high–level mappings from Gist into Will, we have found that the lack of low–level transformations presents an enormous bottleneck to the development of the system in itself. Hence our attention is currently focused on some quite utilitarian goals to provide the Gist– and Will–specific mappings and mapping technology needed to produce a usable system in the near term.

3.2.3 Mappings for Understanding

We have found that Gist specifications are often difficult to read, despite the high-level nature of the constructs used. The essence of the problem is that the modes of communication normally open between people are considerably richer than those between people and machines. We have several ideas on how this communication gap can be narrowed using ***high-level mappings for understanding***. Unlike the ***mappings for optimization*** these mappings often ***change*** one specification into one with observably different behavior.[2]

Our first idea is that one's understanding of a specification should ***evolve*** rather than arise all at once. The power of prose descriptions derives, in part, from reliance on a reader's ***incremental*** understanding as he goes through a document sentence by sentence, paragraph by paragraph. We believe we can convey such incremental understanding of specifications in terms of ***high-level editing*** commands that formally describe the evolution of Gist specifications. We would like to support two basic kinds of changes: explicit changes to the specification itself and implicit changes to the specification expressed as explicit changes to the behavior exhibited.

A second means for enhancing understandability is to allow much more ***expressive*** modes of specification than are presently used. Generally, the formal languages used to describe program behavior to machines (i.e., programming and specification languages) are extremely restrictive and primitive. For example, most journal articles contain tables, grammars, equations, graphs, and other specialized notations to communicate how a program works or what it does. There is no apparent reason why computers cannot begin to understand these richer, higher level modes of expression. We have coined the phrase ***locally formal notations*** to describe the specification technique we espouse, to emphasize that although the notation is self-consistent, it may not be interpreted out of context. That context establishes an interpretation and allows the notation to be concise. Of course, these notations must be mapped onto conventional Gist semantics internally.

The understanding conveyed by the high-level editing commands and locally formal notations—to both human readers and automatic analysis tools—will form a basis for ***evolving and adapting*** software. In fact, within the next two years, such ***high-level mappings for understanding*** will begin to dominate the research goals of the Mappings project.

3.3 SCIENTIFIC PROGRESS: ACCOMPLISHMENTS

Gist area:

Considerable progress along the lines established in previous years was made over this past fiscal year:

- **Transformations to remove Gist high-level constructs.** We have developed some high-level transformations which allow us to remove nondeterminism, to eliminate special cases of recursively derived relations, and to convert references to historical information into code which first saves the "literal" information while it is available (i.e., current) and later retrieves it when needed [9].
- **Transformations for rapid prototyping.** As an aid to understanding the specification, i.e., to ***validate*** that its behavior is what was intended, expedient implementations can be constructed by choosing adequate implementations for Gist high-level constructs.

---

[2] Notable exceptions are the mappings for "rapid prototyping" described in the Accomplishments section below.

These choices need not be optimal, but merely fast enough to examine behavior on a small number of test cases. We have called the use of transformations for such a purpose, ***transformations for rapid prototyping*** [6], although they differ only in that their selection criteria need not be developed fully.

- **Transformations to remove perfect knowledge assumption.** Gist's perfect knowledge assumption allows universal and existential quantification over a virtual global database of arbitrarily complex relations. Implementation of such relations requires a theorem-proving mechanism for testing suitability of proposed realizations. We have begun to experiment with a small set of implementing transformations and have constructed a preliminary theorem proving mechanism to test their feasibility [10].

Although these accomplishments are interesting technically, the details of the mechanisms are quite beyond the scope of this report.

However, in the last year the Mappings project has produced several results of more general interest at a ***methodological*** level. Over the past several years we have gained considerable experience with the design of medium sized specifications in Gist. The Mappings project has played a major role in the design of Gist: in order to understand the validity of transformations the Mappings project has attempted to formalize Gist's semantics. This has led to greater understanding of the specification process and its relationship with both the domain being modeled and the intended implementation environment.

We are basically happy with the design of the formal Gist language. However, we have discovered many factors that tend to make the processes of specification, testing, and implementation quite interdependent. In fact, it is the identification of these factors that constitutes a major result of the Mappings project and leads directly to the future direction the research will take.

3.3.1 Locally Formal Notations

First and foremost, specifications are difficult to construct because they are formal. Formalism is both concise and precise: its construction places demands on a writer which are orthogonal to the creative design decisions that are being made. Hence, details of form often interfere with the specification process. Such diversion of attention causes errors. We have dealt with this aspect of specification in the past by attempting to automatically paraphrase the specification into English. In the future, we hope to ameliorate this aspect by allowing locally formal notations to be specified, in which nonessential detail can be left implicit because it can be derived from the surrounding context.

3.3.2 Specification Evolution

Another reason specifications are hard to read is because the ***evolution*** of the specification is often the only way to understand what is happening. In particular, we have recognized for some time that numerous implementation decisions enter into the actual functionality specified for programs [1]. For example, any portion of a specification that is present to report error conditions to the user, or take actions based on error conditions, could not possibly be a part of the specification of the ***ideal*** behavior of the system—why would an ideal system make errors? Hence, at some point in the evolution of the system, someone realized that the particular implementation that would be used would necessitate dealing with such imperfect behavior. Of course, systems are never ideal: finite resources of both space and time force less than perfect implementations (e.g., finite stacks, heuristic game playing programs, and out-of-paper status flags). We believe that the best way to explain such a specification is as an ***incremental*** modification to the original ideal specification.

In fact, we have identified several categories of high-level editing steps to describe how specifications actually come about, such as refinement, generalization, specialization, exception introduction, implementation decision, and redundancy introduction. We have experimented with these categories as *informal* indications of the development of actual programs using the Develop System [3]. It records design decisions made in the course of developing a program. Develop encourages the developer to classify each design decision into one of these categories and document it in English. We expect to formalize these notions in the future as a part of the Mappings project.

3.3.3 Specification Granularity

Another important discovery we made during the last year concerns the domain being modeled in the specification [7]. An especially confusing aspect of specifications concerns the *granularity* of the model. We have discovered that this confusion arises because there are three different axes of granularity of modeling normally involved:

- ***Structural granularity*** deals with the amount of detail the specification reveals about each individual state of a process: what basic objects and relationships between them are to be modeled. Modeling always involves abstraction away from some details. The choice of which details to include as objects and their relationships is the essence of this type of granularity.
- ***Temporal granularity*** concerns the amount of detail modeled about activities in the original domain. For example, do we model each keystroke of a typist (full duplex), an edited version of an input activity (with rubouts and the corresponding erroneous entry activity invisible), or perhaps the entry of a whole line or file as a single activity in the domain?
- ***Coverage*** deals with the range of possible behaviors permitted by a specification. For example, in the real world some set of simultaneous events may be impossible, yet the modeled world may permit the behavior. For example, in the real world elevator doors may not close when a heavy person is jumping, yet the modeled world does not contain the constraint (or the information on mass sensitivity needed to discover the fact). This involves restricting combinatoric effects of the other two types of granularity.

Typically, specification refinements require dropping to a more detailed level of granularity. If a user is unable to record the evolution of the specification in terms of these layers of granularity, specifications will be difficult to read and understand by future developers and implementors of the specification. Hence, once again, we understand directions for our future work in the Mappings project: to permit an evolutionary description of the specification along these three axes of granularity.

3.3.4 Implementation Specification

Gist specifications are unusual in that the behavior of an entire system and its environment is specified: every independent agent who could affect the behavior of the system must be modeled to the extent that he can affect it. This means that every person and every piece of hardware in the imagined system must be modeled. It is then the task of the implementor to implement a portion of the specification as a computer program.

This past fiscal year we have developed a formal definition of the meaning of "implementation" of a Gist specification, in which a "specification of the implementation" must be provided separate from the Gist specification of the functionality of the system. Such a specification partitions the world into the "system to be implemented" and the "environment in which it will be implemented." This partition

also describes the interface between the two portions; i.e., the limits on one portion regarding control or information availability in the other portion. Any implementation which obeys the interface limitations is said to be *compliant*.

A *correct implementation* imposes different requirements on these two domains. The environment is generally autonomous: the system must react gracefully to *all* nondeterministic activity from the environment. On the other hand, the nondeterminism of the system portion of the specification can be exploited for efficient implementation: *any* behavior from the nondeterministically specified set is acceptable.

The only problematical issue remaining concerning the implementation specification is in establishing the appropriate set of interface limitations. The basic issue is transforming a global data base and globally visible process activity into local data bases of processes with appropriate communication protocols. Hence, we envision that a portion of our development process will consist entirely of a paraphrasing activity attempting to make the factoring into environment and system feasible.

VLSI area:

The Mappings project has attempted to apply the same transformational technology to VLSI design as to software design. It is clear that many of the complexity concerns are shared between the disciplines. In some sense, the software dilemma has been bequeathed to hardware designers through VLSI: they can now conceive of building into hardware, systems that are so complex they could only have been implemented in software before VLSI! The Mappings project had considerable success in capturing real hardware designs in terms of the methodology. These designs were confined to paraphrasing designs previously worked out by VLSI experts. This fiscal year our accomplishments include the following:

- **VLSI as programming constructs.** VLSI primitives have been modeled as constructs in conventional programming languages; hence, they have been made amenable to program transformation. A germinal transformation system was implemented to illustrate the applicability of the model to a real example [11, 12].
- **Systolic array transformations.** A set of symbolic transformations was designed to translate specifications (written in a canonical style) into systolic array processing primitives (a widely used subtechnology of VLSI) [8].

These VLSI accomplishments demonstrate the applicability of this transformation technology to the VLSI domain; in fact, others are beginning to use this technology [13]. No further work on applying transformational technology to hardware design is planned.

3.4 IMPACT

The specification mappings project is supportive of the transformational implementation software development paradigm [2]. Initially, this paradigm will dramatically improve programmers' ability to maintain software systems; ultimately the entire lifecycle from design through specification, implementation, debugging, and maintenance will be streamlined. This will allow programmers to more rapidly produce and maintain software systems and will make feasible the construction of larger, more complex systems with enhanced functionality and flexibility. Of course, a major goal of the Mappings project is to facilitate system maintenance by providing system designers with modes of expression that make specifications more easily understood and hence more modifiable. In addition,

this project will codify the knowledge needed by programmers to convert high-level specification concepts into efficient implementations via mappings. This knowledge would also be useful for programmer education independent of its mechanized application via transformations.

3.5 FUTURE WORK

As was mentioned in Section 3.3, we have developed a set of high-level Gist mappings but have not installed them into the system. We feel their installation should be preceded by developing a large set of medium- to low-level transformations to provide a sufficient underpinning for their use. Hence, the goals of the "optimization portion" of the Mappings project are quite utilitarian; they focus on providing the Gist- and Will-specific mappings and mapping technology for producing a usable system in the near term. In particular, we intend to reimplement or convert our existing facilities into the system under development by the Information Management project (see Chapter 4). Of course, the "mappings for understanding" represent much longer term research goals, with consequently less specific objectives. Specific future goals include the following:

- **Implement annotations as transformations.**
  - To use transformations as the basis for describing annotation methods to the compiler. These transformations will be added to the set of low-level transformations which may be explicitly applied and or used by the simplifier.
  - To perform both Will implementations and IM coordinations[3] via such mapping-based annotations.
- **Integrate software development technology into the Kernel System.** The kernel system being developed for the Information Management project will be used to house all of our software development technology in the near future. In particular, the transformation framework [16] is now being converted. The Gist Mappings project's role in this conversion is twofold; it will be necessary
  - to support the efficient execution of Will, which will be used both as the service-building language for IM and as our systems programming language. This support consists of mappings to implement Will annotations (see above) and general manipulations of Will constructs needed for realistic usage.
  - to design a more complete set of mappings to allow human-guided efficient implementation of IM coordination rules (beyond the simple implementation provided by the annotations mechanism).
- **Real system usage.**
  - To use the transformation system in the development of a substantial piece of itself. Medium- and low-level Gist/Will mappings must be in place for such a goal to be met.
- **Mappings for Understanding.**
  - To develop formal high-level editing mappings to express both changes to behavior and changes to specifications, thus facilitating incremental understanding by humans and incremental analysis and explanation by machines.
  - To develop a framework for locally formal notations, in which context information selects a semantic backdrop to facilitate concise expression of the important aspects of the specification.

---

[3] Rules that maintain the integrity of the system's global database.

REFERENCES

1. Balzer, R., and W. Swartout, "On the inevitable intertwining of specification and implementation," *Communications of the ACM* 25, (7), July 1982.

2. Balzer, R., C. Green, and T. Cheatham, "Software technology in the 1990's: Using a new paradigm," *IEEE Computer*, November 1983.

3. Balzer, R., The Develop system, 1983. In progress.

4. Balzer, R., N. Goldman, and D. S. Wile, "Operational specification as the basis for rapid prototyping," in *Proceedings of the Second Software Engineering Symposium: Workshop on Rapid Prototyping*, ACM SIGSOFT, April 1982.

5. Cohen, D., "Symbolic execution of the Gist specification language," in *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pp. 17–20, Karlsruhe, Germany, 1983.

6. Feather, M., "Mappings for rapid prototyping," in *Proceedings of the Second Software Engineering Symposium: Workshop on Rapid Prototyping*, ACM SIGSOFT, April 1982.

7. Goldman, N., "Three dimensions of design development," in *Proceedings of the National Conference on Artificial Intelligence*, American Association for Artificial Intelligence, Washington, D.C., August 1983.

8. Lam, M., and J. Mostow, "A transformational model of VLSI systolic design," in *IFIP 6th International Symposium on Computer Hardware Description Languages and their Applications*, pp. 65–77, Carnegie–Mellon University, May 1983.

9. London, P. E., and M. S. Feather, "Implementing specification freedoms," *Science of Computer Programming* 2, (2), August 1982, 91–131.

10. Mostow, J., "A problem–solver for making advice operational," in *Proceedings of the National Conference on Artificial Intelligence*, American Association for Artificial Intelligence, Washington, D.C., August 1983.

11. Mostow, J., "Program transformations for VLSI," in *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pp. 40–43, Karlsruhe, Germany, 1983.

12. Mostow, J., "A decision–based framework for comparing hardware compilers," *Journal of Systems and Software*, (4), 1984.

13. Subrahmanyam, P. A., Abstraction to silicon: A new design paradigm for special purpose VLSI systems, 1981. Submitted for publication.

14. Swartout, W., "GIST English generator," in *Proceedings of the National Conference on Artificial Intelligence*, pp. 404–409, American Association for Artificial Intelligence, Pittsburgh, 1982.

15. Swartout, W., "The GIST behavior explainer," in *Proceedings of the National Conference on Artificial Intelligence*, pp. 402–407, American Association for Artificial Intelligence, Washington, D.C., 1983.

16. Wile, D. S., "Program developments: Formal explanations of implementations," *Communications of the ACM* 26, (11), November 1983. Also available as USC/Information Sciences Institute, RR–82–99, August 1982.

4. INFORMATION MANAGEMENT

Research Staff:
Robert Balzer
Dave Dyer
Michael Fehling
Neil Goldman
Matthew Morgenstern
Robert Neches

Support Staff:
Audree Beal

4.1 PROBLEM BEING SOLVED

The Information Management (IM) project is designing and implementing a computing environment that significantly reduces the effort required to create, integrate, and evolve computing services, and enables users to customize these services without detailed knowledge of how they were specified. The basis for such improvements lies in raising the level at which these services are specified and modified.

Current computational services are functionally limited by several factors. They provide only the functionality that

- was anticipated by the service designer,
- had a cost–of–provision/benefit–to–user ratio below a certain threshold, and
- could be provided with minimal, and usually no, reliance on the functionality of other services.

The impact of a designer's foresight is further magnified by the very high cost of modifying that functionality in response to experience, and by the absence of any significant mechanisms for individual users to customize the functionality to their individual needs.

This situation is the inevitable result of the state of the art of software production, maintenance, and execution environments. Disparate services must be implemented with virtually no reliance on one another, both because their execution environments make communication prohibitively expensive and because no mechanisms exist for keeping the implementations mutually consistent across the software life cycle. To significantly customize the behavior of a service, a user requires access to tools that enable him to find and modify relevant parts of the system definition. Both the tools available and the level of language used for definition make this impractical for all but the most sophisticated and persistent users.

The primary task of this project is to implement a testbed software construction and maintenance environment. The testbed must also provide an execution environment for software constructed within it. This testbed must make a higher level of specification directly available to system builders and users without seriously compromising their ability to produce efficient software. Several office–oriented services will be constructed within the testbed. The testbed must be built within constraints defined by the following:

- processing speed and data volume required by the testbed services;
- capabilities of the class of hardware we estimate will be reasonably affordable to a wide range of professionals in the near future;
- the state of the art of compilation technology.

A collection of generic information and process modeling concepts suitable for specifying a wide range of computing services have been identified. Many of these concepts were developed within the framework of the Gist specification language [2]. The IM research project must find a means of delivering these concepts to system designers in a way that supports design, implementation, and maintenance of practical software. In particular, this goal requires the ability to blend specification and implementation concepts in a single language.

To produce the desired testbed, we must blend existing database and artificial intelligence (AI) technologies. The proposed execution environment consists of a single database of interrelated objects that are potentially sharable and modifiable by all services within the execution environment. The database must be self–describing, that is, the declarations that are traditionally considered a database "schema" must themselves be represented within the database. These declarations include semantic constraints on the database. The execution environment must ensure the validity of the database with respect to these constraints at all times.

Services will be specified and modified largely through the use of loosely coupled rules. Techniques for specifying, compiling, and debugging large collections of rules are being actively explored in the AI research community. We expect both to use the results of this research and to contribute to it. Furthermore, since one of our primary goals is to enable users to modify software behavior, the problem of making rule–based software understandable to people other than its authors must be addressed.

4.2 GOALS AND APPROACH

The IM project's research aims to increase the productivity of users of a variety of computer–based services by making it possible to deliver those services in an execution environment that eliminates service boundaries, regularizes basic capabilities across services, and permits personal customization of service functionality. At the same time, it aims to improve the productivity of those who design, implement, and maintain these services. Part of this improvement will come from raising the conceptual level at which these activities are communicated between user and machine, that is, shifting the software developer's activities more toward *specification* and away from *implementation*. Further improvement will arise from the recognition that the development and maintenance of software is itself an information management task. Application of the generic techniques being developed by this project to that specific task will transfer much of the bookkeeping activity inherent in the development and maintenance of large software systems from designers and implementors to programs.

Our approach to this task is to construct a testbed system populated with several useful services. The testbed will be used daily by members of the project and, later, by other researchers at ISI. It will serve three primary roles:

1. It will provide a means of demonstrating progress to individuals outside the project.
2. It will provide feedback from its regular users on the value and limitations of the mechanisms it incorporates.
3. It will provide an experimental framework for measuring the viability of proposed mechanisms relative to the bounds imposed by compilation technology and hardware capacity.

In addition, we expect the testbed to relatively soon reach a state that actually improves our own productivity in implementing and maintaining the services embedded in it.

This testbed kernel will be implemented in LISP. We believe that LISP-based personal workstations provide the best available tool for carrying out the construction of large and rapidly changing software systems. This development will be done using Symbolics 3600 hardware. However, all software will be written within the confines of the "Interlisp compatible" subset of LISP supported on this hardware. The purpose of this decision is to simplify the anticipated porting of the testbed to run on Xerox 1100 (Dandelion) workstations when they become the ISI "standard" workstation.

The programming of services for the testbed will be carried out not in the LISP programming environment, but in IM's own programming environment. A few major pieces of this environment, such as text editing and LISP source code compilation, will be inherited directly from the host system. The remainder of this environment will be written as an IM service. Service designers and implementors will view individual elements of their specifications and implementations as interrelated objects in a database. In this form, all the generic capabilities for managing these objects, as well as composite operations built from the generic ones, will be available.

We believe that we will obtain maximal benefit from this testbed by using it to the exclusion of alternative computing tools for our daily work. For this reason, we also plan to make the testbed available, at least for administrative computing, from whatever class of remote terminals our users find necessary. Initially, this means providing access to the system from HP2640-class terminals via a communications server on one of ISI's VAX 11/780 time-sharing systems.

To promote usage of the system, we have selected for our initial services those which currently dominate our computer usage: electronic mail, document preparation, and LISP software development. We will provide rudimentary, but useful, versions of these services very early in the course of the project. The members of the project will then begin to use these services as part of their standard computing environment, and to customize them using the generic capabilities provided by the IM software environment. Those customizations deemed to be of sufficiently general utility will be migrated into the "standard release" versions of the services.

4.3 SCIENTIFIC PROGRESS

Progress during the first year of this project consisted of laying out a number of design criteria for the envisioned testbed and implementing a prototype version.

4.3.1 Specification-based Computing: Generic Capabilities

The specification basis for the IM system will provide a number of generic capabilities with default implementations and, at least in some cases, alternative implementations that may be chosen by the specifier. All these generic capabilities revolve around the use of a *data model* that can be likened to the "semantic schemas" of database systems like McLeod's SDM [10] or the "concept network" of an artificial intelligence representation language like KL-ONE [3]. The data model provides the following capabilities to our framework:

- It provides a collection of *constraints* on potential states of the actual data environment. It thus provides *protection* against certain kinds of errors that could be made in creating or changing the data.

- It can be used to find potential problems in the software that manipulates the data. Any program, such as a compiler, that analyzes software can point out discrepancies between the data model and assumptions implicit in the software.
- It can be used to select efficient default representations for different classes of data.

The IM computing environment permits people to create and modify data models, to perform operations that map from one instantiation of a model to another, and to explore a given instantiation of a model.

Several essential generic facilities will be provided for the IM user. One such facility is a uniform ability to retrieve information by *description* rather than pointer following. Descriptions, constructed from the terms defined in the data model, make it possible to generate collections of data units that jointly satisfy some condition. For example, the expression

an EMPLOYEE || EMPLOYEE . OFFICE . FLOOR = 10 ,

constructed from the type EMPLOYEE and the attributes OFFICE and FLOOR, can be used to generate those employees whose office is on the 10th floor.

Another generic facility is the ability to create and destroy data units and relationships between them. Again, these operations are specified using the terms defined in the data model. For example, if E is any expression denoting an employee, the statement

update E . PRIMARY-PHONE to 875

could be used to change the value of the PRIMARY-PHONE attribute of that employee to 875.

A third generic facility is the ability to alter the behavior of the environment through *rules*. Rule-based activity will be predicated on the occurrence of some parameterized condition, i.e., a description. Some rules will be used to maintain the *consistency* of the data. Such rules are best thought of as extensions to the data model that allow a specification of interdependencies among data elements. In general, a consistency rule specifies some *transition* to the database that might violate consistency, and some *repair* procedure that can reestablish consistency if it was in fact violated. *Constraint equations* [11] provide a higher level means of specifying consistency rules for a limited but commonly occurring class of situations. Abstractly, a constraint equation permits the expression of a consistency condition by equating two expressions over a common set of variables. Analysis of these expressions determines the possible *transitions* that could violate the condition. *Annotations* to the expressions specify nonprocedurally the desired means of repairing the consistency condition when it is violated. For example,

EMPLOYEE . BACKUP-PHONE = EMPLOYEE . SECRETARY . PRIMARY-PHONE

is the textual representation of the consistency condition "an employee's backup phone must be the same as the primary phone of the employee's secretary." This condition could be violated by transitions involving backup phone, secretary, or primary phone attributes. Annotation of this rule to read

EMPLOYEE ! BACKUP-PHONE = EMPLOYEE . SECRETARY ! PRIMARY-PHONE

constitutes a specification of the repair method to invoke upon violation of that condition. The "!" on the left side of the equation indicates that an employee's backup phone attribute may be changed to restore consistency if the violation was caused by a change to one of the relationships on the right side of the equation. Analogously, the "!" on the right side specifies that a violation caused by a change to an employee's backup phone may be repaired by changing that employee's secretary's primary phone (but not by changing the employee's secretary.)

Other rules will be used to *automate* activity that would otherwise have to be mediated by some human user of the computing environment.

The final generic facility is a uniform interactive interface to the other generic facilities [12]. It will be possible to interactively extend and modify the data model, build descriptions as a means of retrieving information, create and destroy data units and relationships, and install rules to alter the behavior of the system.

We note that, to provide these facilities, it is crucial that the components of the data model, the descriptions, and the rules themselves be modeled concepts. Thus our system must be what is commonly termed "self-describing."

4.3.2 Interactive Use of Computing Services

The fundamental activity of an IM user is to take some given data model and an instantiation of that model and either explore ramifications of that instantiation or compute a new instantiation. The interactive user is presented with a computing environment consisting of a database of interrelated objects, and operations that can be performed for any of the following purposes:

- changing this environment by creating/destroying objects or creating/destroying relationships between objects, thus producing a new instantiation of the data model.
- interacting with the environment outside this database, including I/O devices connected to the user's workstation. In particular, the moment-to-moment content of the user's display device will be defined by user-selected or user-composed descriptions of the information to be presented. The display will be dynamic in that the displayed information will change to reflect changes in the database from which it was derived.
- making requests of other users' environments.

Service boundaries do not exist in an IM system. All communication between modules (on a single workstation) is done by passing direct or descriptive references to objects in a global shared data space. It is still possible for modules to provide stream- or file-based communication interfaces, but the interservice boundaries across which such protocols are required have been removed. Some interactive communications will still be text-based, although *pointing* at representations of information objects will replace textual reference in a pervasive way. Also, because there are no service boundaries, it will always be the case that any visible representation of an information object can be pointed to when interactive reference to that object is required, regardless of whether the reference was anticipated by the module that produced the visible representation of the object. This capability is in contrast to some workstation "window" environments that tie windows to processes in such a way that a user cannot provide input to a process connected to one window by pointing at an object in the window of some other process.

4.3.3 Software Construction and Evolution

Another use of the IM computing environment is to *extend* the data model that the environment instantiates. In IM, this activity is not formally differentiated from the activity of using any other capability of the environment. The user of any set of capabilities views those capabilities as consisting of operations, rules, and data abstractions that he has learned to use in certain ways to achieve certain results. The developer/extender of those capabilities views them as information objects to be manipulated by other operations, rules, and data abstractions—his programming environment. In the case of IM, what the interactive user described in the previous section treated as a *model*, the software developer is treating as the *instantiation* of a different model.

In some advanced programming environments program objects can be treated as data and manipulated by the operators of the programming language (e.g., Interlisp [14], Smalltalk [6], and Cedar [5]). The main benefit these environments provide to the programmer is management of several significant kinds of consistency:

- between source code definitions and active (installed) definitions;
- between source versions and compiled versions of modules;
- between definitions and usages.

While the programming environments mentioned above do not enforce these forms of consistency—indeed, strict enforcement may not be desirable—they record where inconsistencies exist (relieving the programmer of the bookkeeping task) and thus make it relatively easy to reestablish consistency when desired. Because consistency monitoring is a generic IM facility, the above-mentioned examples of consistency, and others deemed important for software development/maintenance, need not be programmed on an individual basis. What *is* necessary is the identification of consistency conditions relevant to the programmer; once identified (specified), the necessary monitoring activity is provided without additional programming.

The second way in which programming environments aid the software developer is provision of a database and query facility regarding the software itself (exemplified by the MASTERSCOPE and File Package facilities of Interlisp [14]). But this is an instance of precisely the activity that IM provides generically for all computing activity—it does not need to be specially provided for software development. Since IM's generic facilities include modification as well as query of its database, modification of software can be accomplished by applying the generic capabilities to software objects.

Several key ingredients are needed to achieve the desired support for software development:

- modeling of software components in the IM database;
- identification of important consistency rules for software components;
- identification of composite views of software that are relevant to software developers; and
- identification of common composite operations on software components.

In support of our contention that software development is really very much like other IM capabilities, we point out that these four ingredients are the same ones we expect to deal with when adding a cluster of capabilities for any new domain of activity.

The argument that IM software will be easier to evolve than conventional software is founded on the specification basis of IM software. Software behavior is defined by a declarative model, by operations defined in terms of generic capabilities, and by rules. The implementation of the behavior so defined is carried out by the system (in some cases with guidance from a "programmer" in the form of annotations to the model). Systems so specified will, we believe, be more *loosely coupled* than conventional software. That is, the change to a specification needed to effect a given change in the behavior will be localized in a specification relative to an implementation. However, the coupling of assumptions that occurs in implementation will be recorded and managed by the system rather than by the implementor. Not only should there be less to change in order to alter the behavior of a loosely coupled system, but it should be easier to find the locus (loci) of the necessary changes. The simplification will follow in part because specifications embody concepts that more directly correspond to those we want to use in describing system behavior, and in part because IM can use histories to attribute observed behavior to existing specifications.

4.3.4 IM Prototype

The second major accomplishment during this year was the implementation of a prototype version of an information management system. This prototype was first developed in Interlisp–10, running on both DEC KL–20 and VAX 11/780 hardware with HP2640 terminals. It was then ported to run on Symbolics Lisp Machines within the Interlisp compatibility package supported by Symbolics. Within the prototype a small demonstration service was built. This service houses data concerning 150 ISI employees, their offices, their business phones, and their ARPANET mail addresses.

This prototype implements a number of the generic capabilities we desire in our system:

- *Objects* can be created and classified within a lattice of *types*.
- *Attributes* may be defined to relate these objects. Each attribute may be constrained to permit it to relate only objects that are classified within specified types. Once created and classified, objects may be related to one another with these attributes.
- *Descriptions* are used for several purposes. Objects may be *found* by associative retrieval from the database, where a description specifies the conditions for retrieval. New objects may be *created* so as to satisfy a given description. Existing objects may be *altered* so as to satisfy a given description. A description is itself an object whose behavior with respect to object retrieval, creation, and modification is determined by its relationship to other objects and descriptions.
- *Rules* may be defined for automatically maintaining consistency among the relationships between objects and for automatically invoking arbitrary processes (programmed in LISP) based on conditions specified in terms of the types and attributes.
- *Views* may be specified to control the display of objects. Any number of views may be associated with a given type. Within the prototype, a view specifies which attributes of an object are to be displayed, and in what order. It may also specify, independently for each attribute, the view to be used in displaying the value of that attribute. The system autonomously maintains two default views for every type: one to display all possible attributes of an object of that type, and one to display the attributes specific to that type but not those inherited from supertypes. The views displayed on the user's terminal are dynamic, in that the display of any information is automatically updated if that information changes in the database.

We implemented an interactive interface for the prototype, permitting a user to refer to an object either via text entered from the keyboard or by pointing at a representation of the object on the display. Pointing is accomplished with "mouse" hardware on the Lisp Machine, and with cursor positioning on the HP terminals. Generic operations (including interactive browsing) and service–specific actions are selected from menus in the Lisp Machine interface, and by keystrokes interpreted through a command table (in the style of the text–editor EMACS) in the case of HP terminals.

4.4 IMPACT

A successful conclusion of our research effort will result in more tightly integrated and evolvable systems, produced with fewer man–years of effort. The integration will arise by composing capabilities from a collection of generic, universally accessible concepts. The ability to evolve capabilities over time will result from the specificational basis of all capabilities in the environment. Both properties will contribute to productivity enhancement, both in what is traditionally thought of as system *construction* and what is traditionally thought of as system *use*.

As a side effect of this primary impact, it should become possible to produce systems with functionality beyond what is achievable from current programming environments. Evolvability will make it economical to provide community–wide or user–specific upgrades to functionality not

anticipated by a service designer. Ease of sharing and specification-based software will reduce the cost of providing a given piece of functionality, encouraging service designers to provide additional functionality in their original designs.

4.5 FUTURE WORK

Bridging the gap between the existing prototype and the desired testbed requires advances in four areas: extending the base of specification concepts for building and maintaining software, enhancing the ability of system designers to control the implementation of their specifications, populating the testbed with useful computing capabilities, and improving the user interface for both the service building and execution activities.

4.5.1 Additional Specification Concepts

The existing specification basis for our IM system consists of a restricted subset of the concepts we developed for the Gist specification language. Certain other concepts from Gist are candidates for inclusion in IM. In particular, we intend to introduce some notion of *events*, which can be used to record a *history* of computing activity. The ability to record a history of system behavior in terms of events defined by the data model will support several capabilities of interest to users:

- It will enable the attribution of observed behavior to the pieces of the data model and to rules that are responsible for that behavior.
- It will enable retraction ("undoing") of certain activity and its consequences when it is discovered after the fact that the activity produced undesirable results.
- It will provide the basis for more descriptive triggers of automation rules.

Another extension to the specification base of IM will be the modeling of procedural knowledge. Whereas our existing design deals in detail with static descriptions of the possible instantiations of a data model, it does not address the description of the possible *temporal sequences* of model instantiations. That is, IM currently provides "specification" level language to define how the world "is," but "program" level language to describe how it changes. Gist includes many specificational constructs dealing with temporal issues. These will be one source of enrichment of IM's terminological base.

4.5.2 Implementation Control

IM will provide the capability to compile and execute any service specified within its service definition language. This capability will result from functionally adequate "default" implementations of each construct in that language. IM will also provide the ability to annotate service specifications so as to select alternative implementations. This is similar in spirit to capabilities that have been tried in Prolog [4] and SETL [13]. IM's mechanism for delivering these capabilities will be technology that has been developed over the past few years in ISI's Transformational Implementation [1] and Mapping [8] research.

At the lowest level of the implementation, we expect a revised database representation to increase object locality, thereby reducing both the execution time for database searching operations and the paging time for the most common kinds of data access. This will improve the efficiency of our most prevalent default implementations.

An example of the form of annotation to be provided is the ability for a specifier to indicate, for a given piece of derived data, whether that data is to be derived on request, explicitly maintained at all times, or derived and cached. Relevant technology for implementing the latter two alternatives can be adapted from truth maintenance systems [9].

4.5.3 Computing Capabilities

Enhancement of capabilities will take place both in the area of additional generic facilities built from the primitive specification concepts, and in specific services.

The most significant near–term extension to the functionality of the IM kernel will be support for *session continuity*. We intend to provide a convenient way to specify a subset of a database to be saved in a permanent form so that it can be restored into a different database at a later time.

In order to ensure a useful computing environment at the termination of this work, and to derive feedback as the work progresses, we will be integrating several office–work–oriented capabilities into the testbed as it evolves. These will include electronic mail, text editing, and document preparation, all of which will make significant use of existing software support for these services.

4.5.4 User Interface

Improvements to the user interface are expected from two sources. ISI's CUE project (see Chapter 2) is building a forms–based interface [7] that we plan to integrate into our testbed within the next year. This provides substantially higher level management of interactive I/O than we currently have available. IM research in the area of user interfaces will focus on providing relevant tools to enable a system designer to effectively use the screen management capability of the CUE interface.

REFERENCES

1. Balzer, R., C. Green, and T. Cheatham, "Software technology in the 1990's: Using a new paradigm," *IEEE Computer*, 1983.

2. Balzer, R., N. Goldman, and D. Wile, "Operational specification as the basis for rapid prototyping," in *Proceedings of the Second Software Engineering Symposium: Workshop on Rapid Prototyping*, ACM SIGSOFT, April 1982.

3. Brachman, R., *A Structural Paradigm for Representing Knowledge*, Bolt Beranek and Newman Inc., Technical Report, 1978.

4. Clocksin, W. F., and C. S. Mellish, *Programming in Prolog*, Springer–Verlag, Berlin, 1981.

5. Deutsch, P., and E. Taft, *Requirements For an Experimental Programming Environment*, Xerox Palo Alto Research Center, Technical Report CSL–80–10, June 1980.

6. Ingalls, D., "The Smalltalk–76 programming system: Design and implementation," in *Fifth ACM Symposium on Principles of Programming Languages*, ACM, 1978.

7. Kaczmarek, T., W. Mark, and D. Wilczynski, "The CUE project," in *Proceedings of SoftFair*, July 1983.

8. London, P. E., and M. S. Feather, "Implementing specification freedoms," *Science of Computer Programming*, (2), 1982, 91–131.

9. McAllester, D., *Reasoning Utility Package User's Manual*, M.I.T. Artificial Intelligence Laboratory, AI Memo 667, April 1982.

10. McLeod, D., *A Semantic Data Base Model and its Associated Structured User Interface*, Ph.D. thesis, M.I.T. Laboratory for Computer Science, 1978.

11. Morgenstern, M., "Active databases as a paradigm for enhanced computing environments," in *Proceedings of the Ninth International Conference on Very Large Data Bases*, 1983.

12. Neches, R., R. Balzer, D. Dyer, N. Goldman, and M. Morgenstern, "Information Management: A specification-oriented, rule-based approach to friendly computing environments," in *Proceedings of the IEEE Conference on Systems, Man, and Cybernetics*, 1984.

13. Schonberg, E., J. T. Schwartz, and M. Sharir, "An automatic technique for selection of data representations in SETL programs," *ACM Transactions on Programming Languages and Systems* 3, (2), April 1981, 126–143.

14. Teitelman, W., *Interlisp Reference Manual*, Xerox Palo Alto Research Center, 1978.

5. INTERNET CONCEPTS RESEARCH

Research Staff:
Jon Postel
Danny Cohen
Carl Sunshine
Paul Kirton[1]
Paul Mockapetris
Greg Finn
Annette DeSchon
Jim Vernon
Joyce Reynolds

Research Assistants:
Alan Katz
Dave Smallberg

Support Staff:
Ann Westine

5.1 PROBLEM BEING SOLVED

The Internet Concepts Research project extends and enhances computer communications. This work is based on the ARPA Internet, a system of interconnected computer communication packet networks. The rules of communication (or protocols) are the key element in successful computer communication. The ARPA Internet has working protocols for communication between heterogeneous computers via an interconnected collection of heterogeneous packet networks.

This research involves work at several levels, host-to–host and gateway–to–gateway protocols and applications protocols. The basic protocols are largely complete now, but a number of extensions are being explored to integrate and extend the packet network technology. In the gateway and host level protocols, these extensions include mechanisms and procedures for monitoring and maintaining performance and survivability, for allowing growth of the Internet, and for dynamic control of the Internet. In the applications level protocols, the focus will be on new uses of the Internet such as multimedia mail, and new capabilities in the gateway protocols.

Another important aspect of the development of protocols is to investigate their correctness. This research studies the Transmission Control Protocol (TCP) using several protocol verification tools.

5.2 GOALS AND APPROACH

The long-term goals of this research are to provide appropriate and effective designs for the primary user service applications in the internetwork communication environment. The designs are based on a set of host and gateway level protocols that provide the full range of service characteristics appropriate to a wide variety of applications and that have been specified and analyzed to ensure their correct operation.

Our approach has been to pursue in parallel the analysis, application, and design of protocols. The interaction of these activities provides valuable insights into problems and potential solutions.

---

[1]Visiting Scholar supported by Telecom Australia

We have identified several program and protocol analysis tools and techniques that show promise of being useful in the study of protocols. We have explored the value of these tools and techniques by applying them to a series of example protocols. The example protocols incorporate features of the TCP.

We have selected computer mail as the application to use as a focus for demonstrating internetwork service. Within this application area we are developing two distinct mail systems, a Text Mail system and a Multimedia Mail system.

The Text Mail system has evolved from the long existing ARPANET mail system. Our contribution to this multi-organization activity focuses on the design and initial implementation of the TCP-based Simple Mail Transfer Protocol (SMTP).

The Multimedia Mail system is an experiment to learn about the future direction of computer mail systems. The goal is to provide mechanisms for computer-oriented data structures to communicate various types of data in messages, including text, graphics, and voice.

In both mail systems, our interest is primarily the communication mechanism rather than the user interface.

We have assisted in the design of several protocols in the Internet family. The areas of addressing, routing, and multiplexing are particularly subtle and require careful attention. We have concentrated on these areas and have explored many options in technical discussions and memos. Our approach is to develop an understanding of these technical issues and to advocate the inclusion of general-purpose supporting mechanisms in the protocol specification.

5.3 SCIENTIFIC PROGRESS

5.3.1 Protocol Analysis

This year we completed our investigation of protocol verification. We used two example protocols—the Alternating Bit protocol and the "three-way handshake" of TCP—to investigate the capabilities of four verification systems: Affirm, Formal Development Methodology (FDM, also known as Ina Jo), Gypsy, and the State Delta system. The work described here represents the conclusion of our protocol verification work. Short reports on this material may be found in [16,20]. A summary of all work done in this area and a complete list of reports produced may be found in [9].

The four automated verification systems we studied were chosen for a combination of factors, including initial estimates of quality, significance, representation of important approaches, and availability to us. Our major interest throughout this work has been on design verification rather than code or implementation verification. Hence we have attempted to develop "abstract" specifications for the services and entities of a given protocol layer and to prove that the combined operation of the entities plus the lower layer service has certain properties or meets some service specification. We have been less interested in the problem of verifying that a specific program or code correctly implements a protocol entity.

A uniform set of example protocols were employed with each system. These were the well-known Alternating Bit protocol (in a form including arbitrary message loss and retransmission) and the three-way handshake connection-establishment protocol from the DARPA TCP [12]. The former

served to test capabilities of the systems to handle "data transfer" type functions, while the latter served to test control functions.

Our goal was to evaluate the ability of existing automated verification systems to provide useful results in the domain of communication protocols. We did not expect to discover errors, since the protocols used as examples were quite mature, although our work with Affirm did reveal an obscure bug with the three–way handshake [10]. The bug has since been corrected.

5.3.1.1 Affirm

Affirm includes a specification language based on the theory of abstract data types, a verification condition (VC) generator, and an interactive theorem prover for proving properties of specifications or of programs [7]. There is also a library of already specified types (e.g., queues, sets) and their properties that may be used in building new types.

The basic modeling approach used with Affirm is to specify each protocol entity as an abstract machine data type. The state variables become "selector functions" of the type, and the inputs or commands become "constructor functions" of the type. A set of axioms must be given to define the effects of each constructor on the selectors (i.e., the effect of an input on the state variables). Thus, there is a relatively straightforward translation of an abstract machine into Affirm axiomatic data type specifications [8].

Properties to be proved were then obtained from the service specification or developed informally. They were entered as theorems along with the specifications, and were proved interactively.

Specifications for an abstract machine were relatively easy to develop in Affirm once the basic approach was developed. However, they were rather tedious to enter at first since an axiom had to be explicitly provided for each state variable/event pair. Many of these were "no–change" axioms, and eventually a language construct was developed to fill in all no–change axioms so that only "interesting" axioms had to be explicitly specified.

Affirm was very good at supporting hierarchical use of data types (i.e., making use of instances of other independently defined types such as queues or records). Composition of separate specifications within a protocol layer was not supported, however, and a global model of the layer had to be prepared.

The theorem prover of Affirm is undoubtedly the most polished of the automated tools sampled, offering a number of useful features including good command syntax, maintenance of good proof histories, flexible movement around the proof tree, ability to "undo" undesired steps, and a well–debugged and well–documented system with online help facilities. Nevertheless, proofs were often a tedious business, because the system forces the user to provide a complete proof, and many cases are similar or mechanical. Greater initiative in completing proofs on its own would be welcome, but this is quite a challenging problem.

In the Alternating Bit protocol, theorems to be proved were derived by "mapping down" a formal service specification. For the three–way handshake, a correctness theorem was derived informally. Both cases required an iterative process of changing theorems and specifications until some of the subtler points of what the system was expected to do and how it accomplished it matched. In particular, inability to prove the original connection–establishment theorem revealed both a bug in the protocol and an unrealizable service expectation [10].

The proof process typically involved development of numerous lemmas to structure the proof. That is, relatively high-powered lemmas had to be formulated in order to prove the top-level theorems, and these in turn had to be proved from lower level lemmas and ultimately from the protocol specification. This may be viewed as development of the "theory" of the protocol "type" in question. The flexibility of Affirm to introduce lemmas in the course of a proof, and to keep track of proof dependencies while letting the user tackle problems in any order, was extremely helpful.

Affirm runs on DEC TOPS-20 and is implemented in Interlisp.

5.3.1.2 Formal Development Methodology (FDM)

Our main interest in FDM was its explicit support for abstract machine models. Thus we expected that it would be easy for us to specify our example protocols, already formulated in terms of an abstract machine model. We also hoped to use the explicit mapping constructs in FDM to do hierarchical proofs showing that a protocol implemented its service. Finally, we wished to experiment with the automated tools associated with FDM, particularly the Interactive Theorem Prover (ITP), to assess their capabilities.

As in Affirm and Gypsy, FDM includes a specification language (Ina Jo), a language processor, a verification condition (VC) generator, and an ITP. The system is specifically intended to model hierarchies of abstract machines, with mappings from higher to lower layers defined. The language is an extension of predicate calculus with some built-in data types (integers, booleans, enumerated sets, lists, records), and it allows the definition of new subtypes and combinations of these types [4].

The basic unit of specification is an event (called a "transform") for which the effect on all state variables must be defined. "No change" is assumed for all variables not mentioned. Properties to be proved about the highest level specification may be conventional invariants (called "criteria") and may also include "constraints" relating the values of state variables before and after an event. (We did not find a need for constraints in any of our protocol examples.)

The Ina Jo language converts specifications (including properties to be proved) into theorems to be proved, one for the initial conditions and one for each transform stating that the properties are maintained. All proof is by contradiction, so these theorems are in a form such that a contradiction must be shown between the hypotheses and all disjuncts of the conclusions. Thus the ITP is more specialized than in Affirm, with induction (over the transforms) and proof by contradiction built in. The prover also automatically generates a number of "corollaries" to each expression resulting from a proof step, based on a large set of built-in facts about the basic types and operators of the language. Either the direct result or any of these corollaries may be used in further proof steps.

It was indeed convenient to write state transition type specifications in Ina Jo, so long as only relatively simple data types were needed. The translation from Affirm specifications to Ina Jo or vice versa is quite straightforward for the type of specifications used. Terse operator syntax reduced understandability by relatively new users but was felt to be advantageous by some experienced users.

Efforts to construct formal mappings from service level to protocol level for the Alternating Bit protocol were unsuccessful, because a nondeterministic mapping was required to represent the faulty medium. Ina Jo supports only the fixed mapping of a higher level transform into lower level transforms. However, this is a common weakness of most systems supporting formal mapping. In other kinds of systems the mapping constructs (where they are applicable) have proved quite useful.

The proof process in Ina Jo is more specialized; induction and proof by contradiction are built into the specification processor and ITP, which produce theorems to be proved false. Our experience with Affirm shows that in some cases invariants can be usefully simplified before induction is employed, eliminating identical steps in each induction case. In the ITP, the proof-by-contradiction method proved a convenient way to break down proofs into components. The subcase selection commands were also well developed. Automatic generation of corollaries was a mixed blessing, since many were not used, but those that were used came for "free."

A serious shortcoming in Ina Jo is the requirement that all lemmas to be used in a proof must be either introduced before the proof is started or proved at their point of introduction. This makes the kind of proof development process that is inevitably necessary in a complex system highly frustrating and tedious. This is all the more true since there is no effective way to "replay" the commands of a previous proof effort other than by retyping them.

FDM was developed by the System Development Corporation. It runs on an IBM 4331 VM/370 system and is based on a LISP-like proprietary compiler writing system called CWIC. "Ina Jo" is a registered trademark of the System Development Corporation.

5.3.1.3 Gypsy

Our main interest in Gypsy was its orientation toward buffer history type specifications with no explicit internal state variables. We also hoped to exploit the modular proof capabilities of Gypsy so that only those portions of a proof affected by changes would have to be redone.

Gypsy is a Pascal-based language with extensions supporting concurrent processing and program verification [3,13]. The language encourages program modularity by forbidding global variables. All interprocedural communication must take place through parameters. A procedure may start up the parallel execution of other procedures. These processes may communicate only through shared message queues, called buffers. A process may send a message to or receive a message from a buffer and will block if the buffer is full or empty, respectively.

To allow verification that a program performs the task it is supposed to perform, assertions may be attached to each procedure. An entry assertion must hold whenever the procedure is invoked; an exit assertion must hold whenever the procedure terminates; and a blockage assertion must hold whenever execution of the procedure is blocked waiting to send to or receive from a buffer. Gypsy enforces modular specifications by requiring that these external assertions not refer to internal variables of the procedure; instead, they must be expressed in terms of the procedure's parameters.

A procedure is proved to meet its external specification by the standard inductive assertion method. Every loop must contain an assertion which holds every time it is encountered during program execution. A VC generator follows every path through the program from one assertion to another, generating VCs which, if true, ensure that the procedure meets its specification. If the VC generator encounters a procedure call along some path, that procedure's entry condition must be checked and its exit assertion may then be assumed. When an operation is encountered which could cause the procedure to block, a VC is generated saying that if the operation blocks, then the procedure's blockage assertion holds.

The VC generator can prove only trivial VCs by itself. Others are left for the human user to prove with the assistance of the Gypsy theorem prover. The prover performs expression simplification

automatically, but most other tasks (such as substitution of equalities, case splitting, and chaining on implications) are best done with human guidance. (There is a command which instructs the prover to try these techniques on its own, but the command doesn't usually work effectively until the last few steps of the proof.) The user may introduce lemmas at any point in the proof.

Specifications for Gypsy programs involving concurrent processes usually contain assertions about the sequences of messages that the processes send to or receive from their message buffers. Gypsy supports the user in making and proving statements about these buffer histories. There is no support, however, for histories of program states, so in general it is impossible to make assertions about liveness properties in Gypsy. For this reason, we considered only safety properties in our specifications of the Alternating Bit and three–way handshake protocols.

Unlike the other systems we worked with, which require nonprocedural formulations of specifications, Gypsy encourages the user to write specifications as programs, at least for concurrent systems. This is because the way to specify a multiprocess system in Gypsy is to define an overall procedure with the individual processes in a Cobegin statement and their interconnecting buffers specified as parameters. Then the VC generator automatically manages the details of building theorems about a parallel program from the assertions describing the behavior of each component process.

By its restrictions on interprocedural communication, Gypsy encourages its users to develop modular specifications. At times, however, we felt that Gypsy imposed a little too much modularity. The prohibition against global variables sometimes required the introduction of extra variables in order to state properties to be verified. For protocols with many states, we found that the requirement that our exit assertions be written in terms of buffer histories forced us to restate in functional form the state transitions which had already been expressed as program text.

The implementation of Gypsy is continually being improved. We noticed that some of its limitations vanished during the course of our research. The major limitation of Gypsy for protocol specification and verification is that liveness properties cannot even be stated, much less verified.

Gypsy was developed at the University of Texas and runs on a DEC TOPS–20 system under ELISP.

5.3.1.4 State Delta

Our major interest in the State Delta system was its ability to perform symbolic execution to accomplish proofs without user guidance. This was very attractive after our often tedious experience with interactive provers. We were also interested in testing whether the explicit time bounds supported in state deltas would facilitate proofs and allow the handling of progress properties as well as safety.

Because the State Delta system was in an early prototype state of development and hence was rather cumbersome and difficult to use, our experiments were quite limited. First we give some background; then we discuss the results of these limited efforts.

The State Delta system includes a specification language and a symbolic execution system and simplifier for carrying out proofs [2]. The initial system covered only a single sequential process, but recent extensions to "concurrent state deltas" (CSDs) have been made by Overman [11,15]. The basic unit of specification is a CSD which gives a precondition, a postcondition, a read list, a mod list,

and time bounds. The meaning of this CSD is that if the precondition ever becomes true, then at some future time within the specified time bounds the postcondition will be true, and in the interim only the variables on the read list will be referenced and only those on the mod list will be modified. There is also a Wait construct, which specifies a delay until either a given condition is satisfied or some maximum time transpires, with postconditions for either case.

Higher level specifications are themselves CSDs. Proof proceeds by determining which lower level CSDs are enabled (have true preconditions) from the preconditions of the high-level CSD and then symbolically executing all the low-level CSDs, keeping track of time and generating all possible interleavings of CSDs that are simultaneously completed in different processors. Any conflict in the use of shared variables is noted, and the proof succeeds if the symbolic execution necessarily leads to a state satisfying the high-level CSD's postconditions (and read list, mod list, and time bounds).

Unlike the other proof systems described, which are interactive, the symbolic execution is completely automatic and requires no user aid. In practice, however, system resources are exhausted for specifications of any complexity, and the user must provide some appropriate intermediate CSDs to force pruning of the proof tree (identical states reached on different branches are not recognized unless explicitly entered as intermediate CSDs). Induction is not directly supported and must also be introduced explicitly if needed, for example, via a loop CSD.

The CSD system is the only one to include time bounds and hence to be able to deal with progress concerns simultaneously with safety. If a proof succeeds, it shows that the goal is reached in the specified time (if any), not merely that the goal may be reached. The time bounds may also be used effectively to eliminate paths from the execution tree that would otherwise have to be considered (e.g., specifying that retransmission interval is greater than transmission delay). However, the inclusion of time bounds also complicates the symbolic execution and so is not always practical.

The basic flavor of CSD specifications is quite different from that of an abstract machine with atomic events separated by long periods when nothing happens. With state deltas, the events have a definite duration and the atomic points in time are their completion/commencement. State variables may change values during a CSD, since only their values at its completion are specified. Since the symbolic execution traces give the sequence of CSD completions, it is difficult to compare them with the state reachability graphs of conventional abstract machine models.

Another limitation of CSDs is the requirement that exactly one CSD be firable at any moment within a single processor. This causes difficulties in modeling nondeterministic behavior such as the condition when both a user input and a message from the network are queued for processing by an entity.

The CSD system is in an early stage of development and hence is still rather clumsy to use. There is little documentation, and only the system's implementors are really capable of using it. The specification language is simply LISP expressions with a particular form required, so input of specifications is rather painful. If a small portion of the specification is changed, there is no capability to determine which portions of previous proofs might be unaffected and thus avoid repeating them.

For simple cases, where the CSD system can complete a proof automatically, it is clearly superior to interactive provers. This is particularly relevant for the state reachability type of properties important in the three-way handshake. The difficulty of proving the simple (no loss or retransmission) case with CSDs was much less than with the systems based on invariant properties. However, when behavior

becomes more complex, a great deal of human ingenuity is still required to formulate successful intermediate-level CSDs, many having the form of invariant properties that must be proved by induction. In this case, the kind of insight needed and the difficulty with all the proof systems becomes similar.

The CSD system was developed at USC/Information Sciences Institute and runs on a DEC TOPS-20 system under Interlisp.

5.3.1.5 Conclusions

Two results of our experience with automated verification systems are clear: None of the systems have all the features desired, and none of them are ready for routine or mechanical application to real-world protocols. All of the interactive systems lack ability to complete seemingly simple or similar portions of a proof themselves, while symbolic execution in the CSD system shows promise in this dimension. All of the systems except Gypsy are missing the capability to omit redoing portions of a proof unaffected by small changes in the specifications or theorems. Affirm omits proof by contradiction, while FDM insists on it. Only Affirm supports induction directly. FDM lacks the ability to introduce lemmas on the fly, as they are needed in the course of a proof.

With the exception of the CSD system, none of the systems are able to handle progress or liveness properties very well. And despite much effort to provide support, hierarchical verification (e.g., of a formal service specification rather than just a set of plausible properties) remains quite difficult for protocols because of their nondeterministic error-recovery mechanisms. Certainly none of the systems have integrated the kind of performance or even probabilistic concerns necessary to go beyond functional correctness of protocols.

If we tried to rank the systems by their ease of use and maturity, Affirm would be a clear first, with Gypsy and FDM in a second category and CSD a distant third.

It should be noted that our experiences and hence our conclusions were colored by our emphasis on verification of protocol designs (i.e., specifications) rather than code. Affirm is particularly strong in this area, while Gypsy and FDM are oriented more toward development and proof of operational code and include features for these purposes that were not fully exercised by our experiments.

Our experience confirms the fact known to verification experts, but not widely appreciated by others, that the major contribution of automated verification systems is NOT to reduce the amount of human ingenuity required to accomplish a proof but rather to increase the certainty of correctness. If the user has the ingenuity to formulate the problem in a tractable fashion and the stamina to follow through all the tedium, the formally verified conclusion does seem to be far more reliably correct than that of hand proofs. Thus some useful results, albeit at high cost, can be obtained from current automated verification systems in analyzing features of real-world protocols.

A number of useful improvements identified in the course of this research have already been incorporated into several of the systems.

We feel that the field as a whole shows sufficient promise that more widespread and routine formal verification of protocol designs may be feasible within a few years if research into automated verification continues to be supported. Whether the best features of different systems can be combined into one more successful system remains a tantalizing question. Again, for more detail on this work please see [8,22].

5.3.2 Protocol Applications

5.3.2.1 The Multimedia Mail system

This message system model takes the view that a message service can be divided into two activities: message reading and composition, and message delivery. Message reading and composition is an interactive activity in conjunction with a User Interface Process (UIP). The message delivery activity may be carried out by background processes called Message Processing Modules (MPMs). Our work concentrates on the message delivery aspects and leaves the development of sophisticated user interfaces to other projects (e.g., Consul and CUE).

The internetwork multimedia message system is concerned with the delivery of messages between MPMs throughout an interconnected system of networks. It is assumed that many types of UIPs will be used to compose and display messages delivered by MPM processes. The MPMs exchange messages by establishing a two-way communication path and sending the messages in a tightly specified format. The MPMs may also communicate control information by means of commands.

A message is formed by a user interacting with a UIP. The user may utilize several commands to create various fields of the message and may invoke an editor program to correct or format some or all of the message. Once the user is satisfied with the message, he "sends" it by placing it in a data structure shared with the MPM. The MPM takes the data, adds control information to it, and transmits it. The destination may be a mailbox on the same host, a mailbox on another host in the same network, or a mailbox in another network.

The MPM calls on a reliable communication procedure to communicate with other MPMs. In most cases, this is a Transport Level protocol such as the TCP. The interface to such a procedure typically provides calls to open and close connections, and to send and receive data on a connection.

The MPM receives input and produces output through data structures that are produced and consumed by UIPs or other programs. The MPM transmits the message, including the control information, in a highly structured format using typed data elements in a machine-oriented yet machine-independent data language [5,6].

This year, the second version of the MPM for TOPS-20 was completed and tested on systems at ISI, Bolt Beranek and Newman Inc., SRI International, and the Massachusetts Institute of Technology. The MPM is now capable of providing regular service.

Programs are available to manipulate bit-map images on the PERQ personal computer and on TOPS-20. Translations of image data between bit-map format and either RAPICOM-450 or CCITT-T.4 facsimile format are possible. Images may be entered into the system via the RAPICOM-450 facsimile machine or may be printed on that machine from files stored on the system.

A simple UIP has been completed on the PERQ. This system has been used frequently for tests and demonstrations of the multimedia mail capability within ISI.

Even though this simple UIP is operational, many difficult problems remain. A number of separate programs should be integrated to provide a better user environment. The appropriate interaction interface for editing multimedia messages has not been studied. The integration of voice into multimedia messages is quite primitive.

5.3.2.2 The Text Mail system

During this year the ARPANET hosts made the transition from the old host-to-host protocol (NCP) to the Internet-based Transmission Control Protocol (TCP) [14]. At this same time, the computer mail system for text mail was changed from the old FTP-based scheme to the new TCP-based Simple Mail Transfer Protocol (SMTP) procedures.

These SMTP procedures are based on our work last year with a prototype Mail Transport protocol and the subsequent revision and improvements. This year we finalized the SMTP specification [19] and provided an initial implementation of the procedures for TOPS-20.

Our initial SMTP system was used also at BBN and at CalTech. We provided debugging aid to many of the implementors of SMTP for other systems. We now view our work on the text mail system as complete, although we will be building interconnections between the multimedia mail system and the text mail system.

5.3.3 Protocol Design and Concepts

This year we began the design of the Domain Naming system, which provides a hierarchical subdivision of the name space to allow an administrative subdivision of the task of maintaining the large and rapidly changing data base.

The Domain system uses structured names to reference things (hosts, mailboxes) in the Internet. The resolution of the names to addresses is accomplished by accessing a Domain Server. Some initial papers on this concept were distributed this year [18]. This work will be further developed in the next year.

Also this year an implementation of a very simple Name Server [1] was completed and put in service on TOPS-20. The program is written in PASCAL. This year we reissued the specifications for Telnet (including the Binary, Echo, Suppress-Go-Ahead, Status, and Timing-Mark options) [48,49,50,51,52,53,54,55] and the "Little Services" (Echo, Discard, Character-Generator, Quote, Active-Users, Daytime, and Time) [56,57,58,59,60,61,62]. We provided TCP-based implementations of the Little Services for our Berkeley-VAX-UNIX systems.

We conducted surveys of the state of implementation of TCP, Telnet, FTP, and SMTP on Internet hosts [27, 28, 29, 30, 34, 35, 36, 37, 39, 40, 41, 42, 43, 44], and we surveyed the Internet hosts for implementation of the Little Services [45].

We studied local network communications from the host's point of view and determined that few protocol systems and interfaces allow the hosts to utilize the network at full speed. Typically, delays in processing prevent a host from sending or (especially) receiving back-to-back packets on the network. We explored design concepts for a high-speed protocol interface. Our resulting design is called the "filter" interface [33]. We will further explore this design in the next year.

We have continued to support the Internet Working Group through presentations, meetings, and preparation of meeting notes, agendas, and other routine documents [17,23,24,25,26,38,47]. We also coordinate the monthly report for the DARPA Internet Program.

5.4 IMPACT

5.4.1 Protocol Analysis

The use of more precise specification methods will facilitate cheaper, faster, and more reliable implementation of the ever-increasing number of communication protocols in DoD computer networks. The research described here has already had some impact on major protocol development projects sponsored by the Defense Communications Agency (DCA) and by national and international standards groups. The previously widespread informal narrative specification methods are being augmented by more formal specifications, particularly of the state transition or abstract machine variety.

The development of protocol verification techniques also promises to improve reliability and reduce debugging time in implementing network systems. Because formal verification allows analysis of protocol designs prior to actual implementation, problems are detected earlier. Esoteric bugs that would probably escape detection by ordinary testing and debugging may also be revealed by formal verification. Several results of this nature have already been discovered and applied to the evolution of TCP.

This research program has also influenced other research projects, particularly in the area of program verification, toward developing analysis techniques that are applicable to computer networks. The results of our work have been widely reported in conferences, journals, and books, and are often cited by others in the field [31,32].

One conclusion of our work in this area is that the verification tools need more power. Unfortunately, we do not see clear evidence that efforts are being made to improve these tools.

5.4.2 Protocol Applications

Computer mail is the most significant use of the new communication capability provided by packet-switching networks. Our work to extend the range and capabilities of computer mail will have important consequences for DoD.

The potential for multimedia communication in a computer-assisted environment is great. The ability to communicate diagrams or maps and to then talk about them will increase the effectiveness of remote communication tremendously. The combination of text, speech, graphics, and facsimile into a common framework and data structure may have substantial impact on other applications as well.

The power of a communication system is directly related to the number of potential communicants. For computer mail, this means that the power of a system is related to the number of people who have access to that system. To have access to a computer mail system requires the use of compatible components: terminals, programs, and protocols. Our work on protocols and programs will increase the power of computer mail by enlarging the set of compatible components.

5.4.3 Protocol Design and Concepts

The selection of the IP and TCP protocols by DoD as the basis for a DoD internetwork protocol standard shows the impact of the work of the DARPA community on DoD communication systems. The development of the Defense Data Network (DDN) by the DCA will be a major use of these protocols in an operational military system. This influence is demonstrated further by the mounting requests for information about IP/TCP from companies interested in producing commercial products or bidding on government contracts.

Through our participation in discussions at the Internet Working Group meetings and in technical meetings with other contractors, we have successfully influenced the development of many protocols and protocol features. Our publication in conferences and journals extends the impact of this work to others who design similar systems [21,46].

5.5 FUTURE WORK

We will work in seven task areas: Intelligent Communication, Hierarchical Naming System, Multimedia Mail, High-speed Interfaces, Design Issues Studies, Surveys and Measurements, and Protocol Specifications.

1. *Intelligent Communication.* We will study a representative sample of existing local interprocess communication mechanisms, develop a draft Internet IPC Protocol, work with other DARPA contractors to refine this draft, and specify a final Internet IPC Protocol. We will create a demonstration implementation of the IPC protocol. We will specify a standard for work order and negotiation formats. We will identify (in cooperation with DARPA) a distributed application and implement it.
2. *Hierarchical Naming System.* We will further develop the Internet Domain Naming Concept and supporting protocols. We will create demonstration implementations of the server and the resolver functions. The resolver and the server will communicate via the Internet.
3. *Multimedia Mail.* We will continue to operate the prototype MPM and develop a simple user mail interface program. We will explore the integration of our simple user interface with the advanced user interfaces produced by the CUE and Consul projects. We will experiment with various styles of preparation of multimedia messages, including various editing techniques and granularity of media elements in the composite messages.
4. *High-speed Interfaces.* We will develop the design of the high-speed filter communication interface. We will conduct software experiments to guide the design decisions. We will build a prototype interface. We will explore the replication of the interface based on a reimplementation in VLSI.
5. *Design Issues Studies.* We will study and prepare reports on Internet protocol design issues, as described in the previous section.
6. *Surveys and Measurements.* We will conduct surveys of Internet protocol implementation features and conduct some performance measurements. We will report the results to the DARPA Internet community.
7. *Protocol Specifications.* We will produce up-to-date documents of the protocols used in the DARPA community on an as-needed basis, and we will manage the assignment of protocol parameters to network experimenters as needed.

REFERENCES

1. Postel, J., "Internet Name Server," USC/Information Sciences Institute, IEN 116, August 1979.

2. Crocker, S., *State Deltas: A Formalism for Representing Segments of Computation,* Ph.D. thesis, University of California, Los Angeles, 1977. Also appears as USC/Information Sciences Institute, RR-77-61, September 1977.

3. Good, D., et al., *Report on the Language Gypsy,* University of Texas at Austin, Technical Report ISCMA-CMP-10, September 1978.

4. Locasso, R., et al., *The Ina Jo Specification Language Reference Manual*, System Development Corporation, Technical Manual TM-(L)-6021/001/00, June 1980.

5. Postel, J., "Internet Message Protocol," USC/Information Sciences Institute, RFC 759, August 1980.

6. Postel, J., "A Structured Format for Transmission of Multimedia Documents," USC/Information Sciences Institute, RFC 767, August 1980.

7. Gerhart, S., "An Overview of Affirm: A Specification and Verification System," *Information Processing 80: Proceedings of the IFIP Congress*, Melbourne, Australia, October 1980.

8. Sunshine, C., et al., "Specification and Verification of Communication Protocol in Affirm using State Transition Models," *IEEE Transactions on Software Engineering*, SE-8, (9), September 1982. Also appears as USC/Information Sciences Institute, RR-81-88, March 1981.

9. Sunshine, C., *Formal Modeling of Communication Protocols*, USC/Information Sciences Institute, RR-81-89, March 1981.

10. Schwabe, D., "Formal Specification and Verification of a Connection-Establishment Protocol," *Proceedings of the Seventh Data Communications Symposium*, Mexico City, October 1981. Also appears as USC/Information Sciences Institute, RR-81-91, April 1981.

11. Overman, W., *Verification of Concurrent Systems: Function and Timing*, Ph.D. thesis, University of California, Los Angeles, 1981.

12. Postel, J., "Transmission Control Protocol," USC/Information Sciences Institute, RFC 793, September 1981.

13. Good, D. and B. DiVito, "Using the Gypsy Methodology," University of Texas at Austin, Draft Report, October 1981.

14. Postel, J., "NCP/TCP Transition Plan," USC/Information Sciences Institute, RFC 801, November 1981.

15. Overman, W., and S. Crocker, "Verification of Concurrent Systems: Function and Timing," in *Proceedings of the Second International Workshop on Protocol Specification, Testing, and Verification*, Idyllwild, California, North-Holland, May 1982.

16. Sunshine, C., "Experience with Four Automated Verification Systems," in *Proceedings of the Second International Workshop on Protocol Specification, Testing, and Verification*, Idyllwild, California, North-Holland, May 1982.

17. Sunshine, C., "Routing for Stub Gateways," USC/Information Sciences Institute, June 1982.

18. Su, Z., and J. Postel, "The Domain Naming Convention for Internet User Applications," Network Information Center, SRI International, RFC 819, August 1982.

19. Postel, J., "Simple Mail Transport Protocol," USC/Information Sciences Institute, RFC 821, August 1982.

20. Sunshine, C., "Protocol Specification and Verification Work at USC/ISI: Summary Report," USC/Information Sciences Institute, IEN 211, August 1982.

21. Postel, J., C. Sunshine, and D. Cohen, "Recent Developments in the DARPA Internet Program," *Pathways to the Information Society, Proceedings of the Sixth International Conference on Computer Communication*, ICCC, London, September 1982.

22. Sunshine, C., and D. Smallberg, Automated Protocol Verification, USC/Information Sciences Institute, RR-83-110, October 1982.

23. Postel, J., "Request for Comments on Requests for Comments," USC/Information Sciences Institute, RFC 825, November 1982.

24. Postel, J., "The Remote Telnet User Telnet Service," USC/Information Sciences Institute, RFC 818, November 1982.

25. Postel, J., and J. Vernon, "Requests for Comments Summary - Notes 600-699," USC/Information Sciences Institute, RFC 699, November 1982.

26. Postel, J., and J. Vernon, "Requests for Comments Summary - Notes 700-799," USC/Information Sciences Institute, RFC 800, November 1982.

27. Smallberg, D., "Who Talks TCP? - Survey of 7-Dec-82," USC/Information Sciences Institute, RFC 832, December 1982.

28. Smallberg, D., "Who Talks TCP? - Survey of 14-Dec-82," USC/Information Sciences Institute, RFC 833, December 1982.

29. Smallberg, D., "Who Talks TCP? - Survey of 22-Dec-82," USC/Information Sciences Institute, RFC 834, December 1982.

30. Smallberg, D., "Who Talks TCP? - Survey of 28-Dec-82," USC/Information Sciences Institute, RFC 835, December 1982.

31. Sunshine, C., ed., *Proceedings of the Second International Workshop on Protocol Specification Testing and Verification*, North-Holland, 1982.

32. Sunshine, C., "Guest Editorial: Protocol Specification, Testing, and Verification," *Computer Networks*, 6, (6), December 1982.

33. Mockapetris, P., *Communication Environments for Local Networks*, USC/Information Sciences Institute, RR-82-103, December 1982.

34. Smallberg, D., "Who Talks TCP? - Survey of 4-Jan-83," USC/Information Sciences Institute, RFC 836, January 1983.

35. Smallberg, D., "Who Talks TCP? - Survey of 11-Jan-83," USC/Information Sciences Institute, RFC 837, January 1983.

36. Smallberg, D., "Who Talks TCP? - Survey of 18-Jan-83," USC/Information Sciences Institute, RFC 838, January 1983.

37. Smallberg, D., "Who Talks TCP? - Survey of 25-Jan-83," USC/Information Sciences Institute, RFC 839, January 1983.

38. Postel, J. "Assigned Numbers," USC/Information Sciences Institute, RFC 820, January 1983.

39. Smallberg, D., "Who Talks TCP? - Survey of 1-Feb-83," USC/Information Sciences Institute, RFC 842, February 1983.

40. Smallberg, D., "Who Talks TCP? - Survey of 8-Feb-83," USC/Information Sciences Institute, RFC 843, February 1983.

41. Clements, R., "Who Talks ICMP, Too? - Survey of 18 February 1983," USC/Information Sciences Institute, RFC 844, February 1983.

42. Smallberg, D., "Who Talks TCP? - Survey of 15-Feb-83," USC/Information Sciences Institute, RFC 845, February 1983.

43. Smallberg, D., "Who Talks TCP? - Survey of 22-Feb-83," USC/Information Sciences Institute, RFC 846, February 1983.

44. Westine, A., "Summary of Smallberg Surveys - February 1983," USC/Information Sciences Institute, RFC 847.

45. Smallberg, D., "Who Provides the Little TCP Services?" USC/Information Sciences Institute, RFC 848, March 1983.

46. Cohen, D., and J. Postel, "Gateways, Bridges, and Tunnels in Computer Mail," *Local Net 83*, Online Conferences, London, March 1983.

47. Postel, J., "Official Protocols," USC/Information Sciences Institute, RFC 840, April 1983.

48. Postel, J., and J. Reynolds, "Telnet Protocol Specification," USC/Information Sciences Institute, RFC 854, May 1983.

49. Postel, J., and J. Reynolds, "Telnet Option Specifications," USC/Information Sciences Institute, RFC 855, May 1983.

50. Postel, J., and J. Reynolds, "Telnet Binary Transmission," USC/Information Sciences Institute, RFC 856, May 1983.

51. Postel, J., and J. Reynolds, "Telnet Echo Option," USC/Information Sciences Institute, RFC 857, May 1983.

52. Postel, J., and J. Reynolds, "Telnet Suppress Go Ahead Option," USC/Information Sciences Institute, RFC 858, May 1983.

53. Postel, J., and J. Reynolds, "Telnet Status Option," USC/Information Sciences Institute, RFC 859, May 1983.

54. Postel, J., and J. Reynolds, "Telnet Timing Mark Option," USC/Information Sciences Institute, RFC 860, May 1983.

55. Postel, J., and J. Reynolds, "Telnet Extended Options – List Option," USC/Information Sciences Institute, RFC 861, May 1983.

56. Postel, J., "Echo Protocol," USC/Information Sciences Institute, RFC 862, May 1983.

57. Postel, J., "Discard Protocol," USC/Information Sciences Institute, RFC 863, May 1983.

58. Postel, J., "Character Generator Protocol," USC/Information Sciences Institute, RFC 864, May 1983.

59. Postel, J., "Quote of the Day Protocol," USC/Information Sciences Institute, RFC 865, May 1983.

60. Postel, J., "Active Users," USC/Information Sciences Institute, RFC 866, May 1983.

61. Postel, J., "Daytime Protocol," USC/Information Sciences Institute, RFC 867, May 1983.

62. Postel, J., and K. Harrenstien, "Time Protocol," USC/Information Sciences Institute, RFC 868, May 1983.

6. COMMAND GRAPHICS

Research Staff:
Richard Bisbey II
Benjamin Britt
Pamela Finkel
Dennis Hollingworth

Consultant:
Danny Cohen

Support Staff:
Lisa Holt

6.1 PROBLEM BEING SOLVED

A major issue for the military is improved utilization of available data to enhance the Command and Control decision-making process. The military, like its private sector counterpart, currently finds itself in the midst of an information explosion. More computers and computer-controlled systems are being acquired, generating information in ever-increasing quantity and detail. For this information to be useful in decision making, it is necessary that computers take more active roles in storing, retrieving, analyzing, integrating, and presenting data.

The man-machine interface is a critical link when computers are used to aid the decision maker. Information must be presented to the decision maker in ways that enhance and facilitate the decision process. Two-dimensional graphics can play an important role in improving this interface. Where spatial relationships exist, plotting the information on a graph, bar, or pie chart or positioning the information on a map can aid in the rapid assimilation of the information by the decision maker. Such two-dimensional displays can even disclose perspectives (e.g., a trend on a graph or a clustering of forces on a map) that would not be readily apparent from a table or list of numbers. Finally, two-dimensional displays provide a natural medium for integrating and fusing information.

6.2 GOALS AND APPROACH

To address the above needs, ISI designed a graphics system based on the premise of distributing the processing load across hosts in a computer network [2]. The architecture defines a graphics system as a series of isolatable functions that are pairwise connected by any available intraprocess/interprocessor communications mechanism. Information is communicated between functions using a uniform protocol. The architecture supports a wide variety of configurations ranging from clustering all functions on a single host to distributing each to a different host.

ISI has also developed a set of generic graphics primitives (Graphics Language [3]) by which pictures can be described and interacted with at the application level. The graphics primitives are transformed by the Graphics System at program execution into specific operations and display modes appropriate to the display device to which the system is connected.

6.3 SCIENTIFIC PROGRESS

A major effort during this period has been the reimplementation of the Graphics System for the UNIX operating system. The Graphics System was originally designed for and used in DARPA's Advanced Command and Control Architectural Testbed (ACCAT). The computing architecture used in ACCAT consisted of a network of PDP-10s connected to PDP-11s. The PDP-10s were used for large

computations and distributed database storage, while the PDP-11s were used as Remote Site Modules (RSMs) for secure terminal/network access to the PDP-10s and for attaching special display hardware. The Graphics System was written for and run on the PDP-10s, initially under the TENEX operating system and subsequently under TOPS-20.

The Graphics System is now being adapted for use in DARPA's Strategic C3 Program. Here the computing architecture consists of a network of VAXes running the UNIX operating system. This adaptation required reimplementing the Graphics System in the "C" programming language and interfacing it to the UNIX operating system.

Functionally, the UNIX implementation of the Graphics System is identical to the TOPS-20 implementation. The same overall design found in the TOPS-20 implementation has been retained in the UNIX implementation. The system is distributable, i.e., portions of the system can be run on separate hosts, and is based on a logical "data channel" over which messages are passed to cooperating processes. Distributability makes it possible to run half of the Graphics System on a PDP-10 running TOPS-20, and the other half of the system on a VAX running UNIX.

The UNIX implementation of the Graphics System Backend (i.e., that portion of the Graphics System responsible for external Graphics Files, internal segment/pseudo-display files, and device order generation) was completed during this period. Two different device types are supported in this initial implementation, the Tektronix 4010/12/14 series monochrome displays and the Advanced Electronic Design 512 color bitmap display. The Backend code underwent extensive testing using the previously mentioned distributed processing capability of the system. Test applications and the Graphics System Frontend were run on a TOPS-20 host while the Backend was run on a VAX. Major portions of the Graphics System Frontend were also implemented in "C" during this period. The UNIX conversion tasks remaining are the completion of the Frontend implementation, and testing and integration of the Frontend with the Backend.

During the reporting period, fundamental changes took place in the ARPANET in both the protocols and the devices used. These changes severely impacted the Graphics System. In January 1983, the underlying protocol used on the ARPANET was changed from the Network Control Protocol (NCP) to the Internet Protocol/Transmission Control Protocol (IP/TCP). Since the Graphics System and all the graphics applications had been written using NCP, all the network communications routines had to be reimplemented. This process was further complicated by the fact that major sections of the IP/TCP system code used by the Graphics System for communicating both graphic protocol and display device orders had never been tested. Thus, the conversion process was not limited to the conversion and testing of Graphics System code, but also included the testing and debugging of IP/TCP system code.

At the same time the NCP-to-IP/TCP conversion was taking place, hardware on the ARPANET was also being replaced. In particular, TIPs were being replaced with TACs. Unfortunately, the TACs were missing several important fundamental capabilities, including the ability to send and receive 8-bit binary data and the ability to allocate individual terminal lines. A considerable amount of time was expended diagnosing these problems and getting responsible people to make corrections, first for NCP, and then for IP/TCP. Problems remain with the TAC that prevent its use with 8-bit binary data, and ISI has continued to interact with TAC programmers to fix the remaining problems.

Two papers were written [1, 4], with the latter presented at the NATO-sponsored conference on Interoperability of ADP Systems held in The Hague, Netherlands. During the conference, several graphics demonstrations were given using the Graphics System in a multinetwork environment. Figure 6-1 shows the network topology used for the demonstrations.

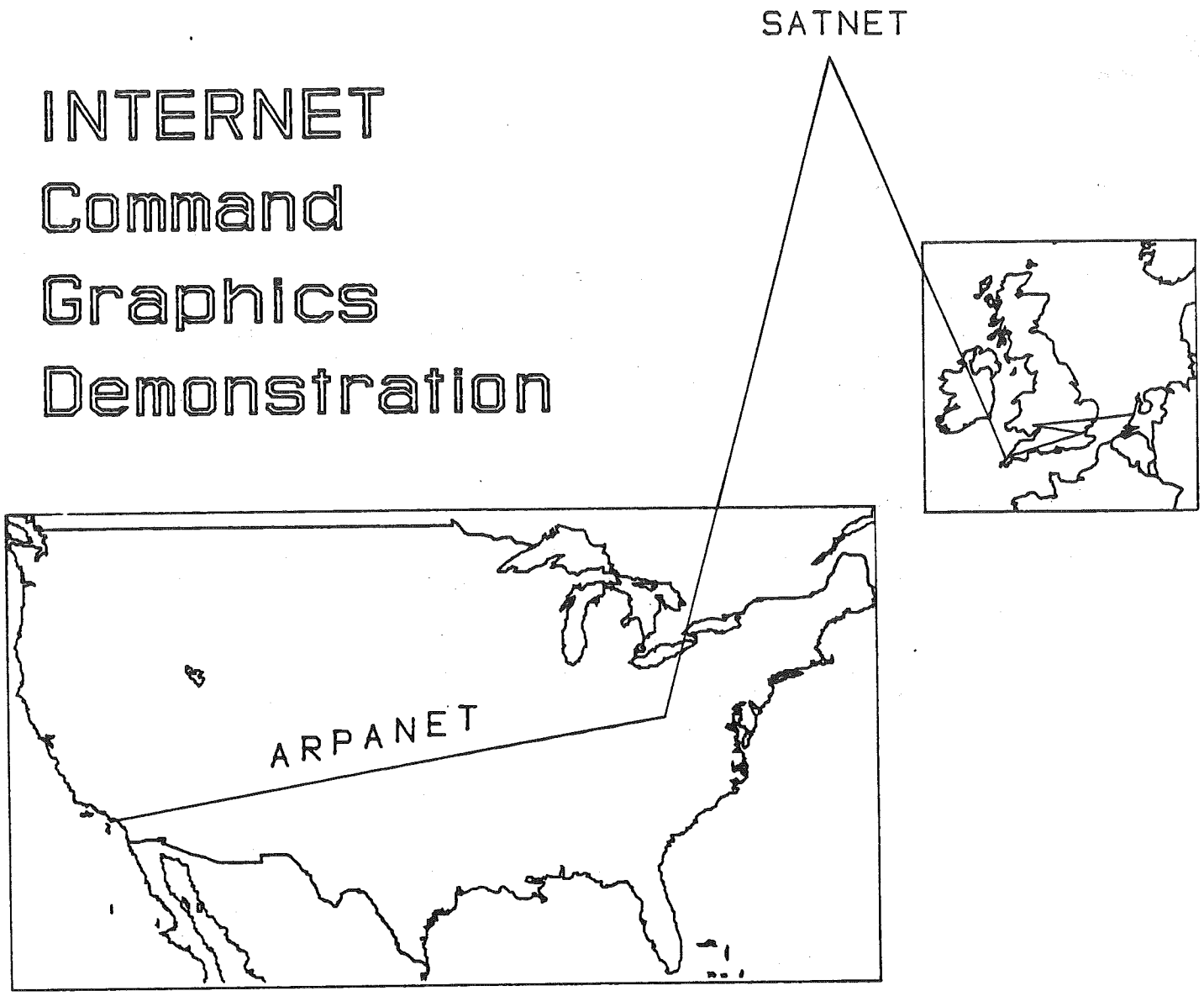


Figure 6-1: Network topology used for Graphics System demonstrations

6.4 IMPACT

The principal impact of this work will be felt in military environments where command mobility is paramount and a portable, network-based graphics capability is necessary for information to facilitate command decisions. This work will make the Graphics System available in the HERT (Headquarters Emergency Relocation Team) and other mobile environments.

The UNIX-based Graphics System will provide the fundamental network graphics capability for the Strategic C3 Experiment. Graphics applications can be run on a single VAX host or distributed between multiple VAX hosts. Graphic applications can also utilize PDP-10 computing resources, either distributed or nondistributed. Finally, the UNIX/"C" implementation of the Graphics System will form the basis for a Motorola MC68000 microprocessor version, permitting a highly portable graphics capability.

6.5 FUTURE WORK

Three basic components are needed if graphics is to play a role in military decision making. First, there must be a graphics system, a program that converts basic graphics primitives (e.g., lines, text, filled solids) into the order codes for a particular display device. Second, there must be high-level tools in the form of intelligent, domain-independent display agents that, given a collection of data, produce a graphics representation of that data. Third, there must be high-level applications and decision aids that perform the semantic binding between the user's data and the high-level graphics representations provided by the display agents.

Having already produced a network-distributable graphics system, ISI will focus its efforts in the area of domain-independent display agents. In particular, ISI will design and build a Geographic Display Agent. This "intelligent" agent will automate cartographer functions of drawing and annotating geographic-based displays. It will assume total responsibility for the production of the display, including generation of background maps and generation, placement, and symbol collision resolution of user annotations. The initial use of this agent will be for the production of geographic displays for the C3 Experiment Bomber Recovery Application.

REFERENCES

1. Bisbey, R., II, D. Hollingworth, and B. Britt, "A network graphics system for command and control," in *Proceedings of the Symposium on Interoperability of Automated Data Systems*, North American Treaty Organization, The Hague, Netherlands, 1982.

2. Bisbey, R., II, and D. Hollingworth, *A Distributable, Display-Device Independent Vector Graphics System for Command and Control*, USC/Information Sciences Institute, RR-80-87, 1980.

3. Bisbey, R., II, D. Hollingworth, and B. Britt, *Graphics Language*, USC/Information Sciences Institute, TM-80-18, 1980.

4. Hollingworth, D., *C2 Graphics Editor User's Manual*, USC/Information Sciences Institute, TM-83-24, 1983.

7. WIDEBAND COMMUNICATIONS

Research Staff:
Stephen Casner
E. Randolph Cole
Ian Merritt

Support Staff:
Victor Brown
Lisa Holt
Jeff LaCoss
Jerry Wills

7.1 INTRODUCTION

The Wideband Communications (WBC) project at ISI is one of several groups involved in the joint DARPA/DCA Wideband Packet Satellite Program. The objective of the Wideband Program is to explore the technical and economic feasibility of packet voice on a large scale, and to begin investigations of other media, such as packet video.

ISI's role is to conduct experiments using the Wideband Network as it becomes available for regular, active use. ISI has been closely involved in efforts to make the network as reliable and useful as possible, and has established a schedule of regular tests and experiments in an effort to encourage real use of the network, thus gaining as much practical experience as possible.

7.2 PROBLEM BEING SOLVED

As the military is asked to do more and more with less manpower, communications becomes increasingly important. Reaction times must be fast, and many different kinds of data must be transmitted, correlated, and analyzed. This is very difficult, even impossible, to achieve when different networks are used for voice, data, facsimile, video, and other media. It is difficult enough to provide voice communications—let alone data and other media—in a tactical environment.

The DARPA Network Secure Communication (NSC) program took one of the first steps toward integrated military communications when it provided voice communication along with data on a packet-switched network. The NSC program showed very convincingly that packet voice was feasible. Moreover, a study by the Network Analysis Corporation [4] showed that a single packet network for both voice and data could support the DoD's communications needs much more efficiently than conventional alternatives.

However, the data rate of the network used (the ARPANET) was not high enough to support a meaningful volume of voice traffic. In addition, the cost of a packet voice terminal (speech coder and decoder and network interface) was prohibitively high.

Advances in VLSI have now made it possible to build a packet voice terminal, about the size of a normal telephone, for a few thousand dollars. However, a packet network with a high enough data rate to support reasonable volumes of voice traffic was still needed.

The DARPA/DCA Wideband Packet Satellite Network was built in 1980 to provide a high-rate packet network for experiments with multiple packet voice streams and other high-rate data. The Wideband Network provides a raw data rate of about 3 Mb/s, which is sufficient for a thousand or more voice channels when a narrowband speech compression technique such as LPC is used.

The Wideband Network is currently being actively used for packet voice experiments. Since the packet voice terminals used have not yet been produced in large numbers, much of the load is generated by traffic simulators at various points in the network. The ISI WBC project has expended considerable effort in developing an interface which provides voice access to the Wideband Network from ordinary telephones, in order to promote everyday use of the network and gain realistic user experience as early as possible.

Since the Wideband Network can transmit data at a rate more than fifty times that of the ARPANET, it will allow experiments with other media, such as packetized compressed video, which require inherently high data rates. Since video signals share many of the characteristics of voice signals, i.e., a "bursty" nature and a high degree of redundancy, the ISI WBC project is conducting experiments to demonstrate the utility of packet video in the same way that the NSC program demonstrated the utility of packet voice. Thus the Wideband Network will serve as a research testbed for packet video, just as the ARPANET served as the original research testbed for packet voice.

The Wideband Network will help further extend the concepts of integrated networking to much higher data rates, while maintaining all the advantages of security and survivability which are inherent in packet switching. In addition, the Wideband Network is powerful enough to connect together high-bandwidth local area networks (LANs) for the first time.

Compared to existing terrestrial packet networks, satellite-based networks such as the Wideband Network have some disadvantages: longer delay and a higher bit error rate. However, satellite-based networks have a much higher typical data rate and an inherent broadcast nature. Communications systems using the Wideband Network will be designed to take advantage of its strong points and to compensate as much as possible for its weaknesses. Much of the research task will lie in designing the communications systems to fit the characteristics of the network.

7.3 GOALS AND APPROACH

The initial goal of the DARPA/DCA Wideband Program is to test and validate the feasibility of packet voice on a large scale. Later, attention will be focused in other directions, including experiments with high-rate transmission of more conventional non-real-time data and the application of packet-switching techniques to media such as packet video for the first time.

The overall approach of the ISI WBC project is twofold: to expedite the transition of mature technology, such as packet voice, from the laboratory into the everyday working environment; and to apply the unique capabilities of the Wideband Network to new areas, such as packet video and high-rate data transmission. This approach requires a broad spectrum of efforts, including development of hardware, software, and protocols. Development of research technology into practical systems requires a combination of highly reliable hardware and robust, easy-to-use software. Development in new areas such as packet video requires powerful new hardware which is flexible and programmable, along with development and testing of new software and protocols.

The ISI WBC project has been working in four specific areas:

1. distribution, support, and improvement of the Switched Telephone Network (STN) interface, a device designed to greatly expand the availability of packet voice and encourage its regular use;
2. design and implementation of hardware and software for packet video experiments, including offline software simulations of the algorithms to be implemented;

3. installation of hardware at a second site (the MIT Lincoln Laboratory), which will participate with ISI in packet video experiments;
4. enhancement of Wideband Network reliability by working to solve system and interference problems and by improving the physical installation of the ISI earth station hardware.

Useful experience with any new technology is the result of its exposure to a large number of potential users. Even though the cost and size of packet voice terminals have decreased greatly, as demonstrated by the Lincoln Laboratory PVT, existing terminals are still too large and expensive to supply to large communities of users who may use them only occasionally. To expand the availability of packet voice and gain the experience of a wider user community, the ISI WBC project designed and built the STN interface [1] and distributed it to other Wideband sites.

The ISI WBC project is working to answer the question: "Is video as well suited to the packet-switched network environment as voice?" Video signals, like voice, are "bursty," i.e., the picture can change rapidly from one scene to the next and then stay fairly constant for some time. Current digital video transmission systems are designed to work with fixed-capacity networks, and they use the full channel capacity whether or not it is required. Packet-switched systems, on the other hand, are designed to carry bursty traffic, taking advantage of the fact that, given a large number of sources of traffic, the overall network load stays fairly constant.

The goal of the packet video experiments is to build a video transmission system which is optimized for a packet environment. This goal permits—even dictates—the design of a system which can detect change, or the lack of change, from frame to frame in the video input. Other characteristics of packet networks, such as selectable priority on a per-packet basis, can be used to allow the network to discard less important video information while making sure the most important video information always gets through.

It should be possible to build a system capable of transmitting color television-type video with moderate motion at a data rate of 1.5 megabits per second. Along the way, monochrome video and perhaps slower frame rates will be used.

Our approach has been to survey the literature and technology of video bandwidth compression and select the best overall compression technique, and then to develop a hardware/software system that is as programmable and flexible as possible in order to support a variety of experiments.

Programmability will make the ISI system unique. Although commercial video bandwidth compression systems are available, they are not programmable in any way, and they cannot be customized to take advantage of the Wideband packet satellite channel.

In order to provide a realistic, useful test of the ISI packet video system, an identical set of hardware for packet video experiments is being assembled at Lincoln Laboratory, and video transmissions across the Wideband Network will begin as soon as software work is completed. Initial transmissions will consist of uncompressed frames of video, transmitted as often as the network allows. Experiments with compressed video will begin as soon as the necessary hardware and software are available.

7.4 SCIENTIFIC PROGRESS

7.4.1 STNI—Switched Telephone Network Interface

In order to increase the availability of the Wideband Network for potential voice users, ISI has designed the Switched Telephone Network (STN) interface [1]. The STN interface sits between a telephone line and a Wideband Network voice terminal (the Lincoln Laboratory PVT), and receives commands from either DTMF (Touch Tone) signals from the user's telephone or digitally, from the PVT. The STN interface is based on a Z80 microprocessor, and consists of a single 7–inch by 7–inch circuit board which plugs directly into the Lincoln Laboratory PVT (or it can be used stand–alone).

To handle an incoming call, the STN interface first answers the ring and sends the caller a second dial tone. The user then commands the interface with a series of key presses (called a dialing sequence), and the interface uses that information to make a Wideband Network voice connection, either to a normal PVT or to a PVT which contains another STN interface.

To generate an outgoing call, the user first makes a Wideband Network voice connection to a remote PVT with an STN interface. Connection information is passed over the Wideband Network to the remote PVT, which instructs its STN interface to pick up the line and place a call.

STN interfaces are now permanently installed at ISI, Lincoln Laboratory, and SRI International. Permanent installation of the STN interfaces at Lincoln Laboratory was delayed until February 1983, due to the installation of a new PBX there. Since that time, however, ISI WBC project members have used the STN interfaces and the Wideband voice network for calls to the Lincoln Laboratory area whenever possible.

The people at SRI International have modified their STN interfaces to serve as the standard audio input/output device with their Packet Radio Network terminals, in addition to the normal STN interface functions. A number of improvements have been made to the STN interface software, including

- serial vocoder protocol support for use in LPC or other low–rate voice transmissions;
- a timeout to disconnect a hung–up connection that did not return to dial tone;
- special handling to accommodate the timing requirements of the ISI PBX;
- enhanced interaction between the STN interface's silence detection and its echo–suppression logic to eliminate occasional superfluous packets.

In addition, a number of enhancements have been made to the PVT software which deals with the STN interface, including

- better integration of the ISI dialing module into the PVT software;
- a timer to return to dial tone after a long period in which no digits were received from the user;
- new translations to allow dialing to area codes adjacent to 213, including 619 (San Diego area), 714 (Orange County), 805 (Ventura County), and 818 (San Fernando Valley area, as of January 1984);
- improved status feedback for users during conferences.

A pesky problem with echoes results when any device is connected to the switched telephone network. Echoes are not apparent when there is little or no delay in the audio path or when the

communication is half duplex (one direction at a time). However, the Wideband packet voice network is full duplex, and the satellite link introduces a relatively long delay (230 milliseconds).

The current version of the STN interface uses a technique called echo supression, which basically forces the voice conversation to be half duplex, i.e., one party cannot speak while the other party is speaking. A device called an echo canceller can eliminate the echo problem, but these devices have been relatively large and expensive. Newly introduced microprocesors capable of real–time signal processing can be programmed to do echo cancellation, and future plans include integrating a single–chip echo canceller into the STN interface.

7.4.2 Channel Improvements

As is the case with any research and development tool, the amount of progress on Wideband Network applications depends largely on the availability of the Wideband Network itself. To ensure maximum availability, two kinds of effort are needed: a sustained effort to keep all the components of the system aligned, calibrated, and working properly, and an occasional concerted effort to implement a new feature or function.

In the past year, the ISI WBC project has

- participated in a special task force of Wideband Program contractors established to address and solve reliability problems;
- worked with Probe Systems, Inc., to isolate and cure a significant RF interference problem;
- worked with Western Union to relocate the ISI earth station hardware from a non–ideal outdoor enclosure to an inside rack (which greatly improved reliability), and to refocus the ISI earth station antenna to increase its gain and thereby decrease the channel's bit error rate.

7.4.2.1 The Wideband Program Task Force

The Wideband Task Force was established in March 1983, with a participant from each of the primary Wideband Program contractors. The job of the Task Force was to focus a large amount of effort and attention on a number of then–existing problems.

The method chosen by the Task Force was to identify the problem areas, assemble the key people required to investigate and fix the problem at a single site, and let them work on the problem for a week, or as long as it took to identify and solve the problem. Generally, more than one problem was targeted, and all the problems were isolated and either repaired on the spot or the equipment returned to its source for repair.

The Task Force met at Lincoln Laboratory in April, and at ISI in May and June 1983. Weekly reports were circulated by Lincoln Laboratory to all Task Force members. The Task Force has accomplished the following:

- Numerous system problems have been resolved through the intensive on–site testing and debugging.
- The following channel improvements have been made:
  - antennas at ISI and Lincoln Laboratory have been refocused, with 1 dB improvement at both sites;

- an improved low noise amplifier (LNA) has been tested at Lincoln Laboratory and found to result in a 1 dB improvement;
- the Earth Station Interface (ESI) now has improved burst aggregation and hardware error checking;
- the ISI Packet Satellite IMP (PSAT) has had numerous software bugs removed and is capable of reliable two-site operation at 1.544 Mb/s.

7.4.2.2 Channel interference location

Since the Wideband Network became operational at ISI, an intermittent but severe interference problem had existed. Initially, the interference was sufficient to crash the network software in the PSAT. In time, software safeguards were installed. The interference no longer crashed the system, but it was still quite bothersome. The interference occured several times per hour, primarily during the daytime, and lasted up to five seconds. ISI WBC project personnel observed and recorded numerous instances in which images transmitted over the Wideband Network were partially or completely obscured by noise patterns.

A number of experiments were conducted to assure as much as possible that the interference was not internally generated in the ISI network hardware or software. When nothing was found in the ISI system, a contract was issued to Probe Systems, Inc., to try to locate, identify, and solve the interference problem. The Probe Systems investigators rapidly confirmed that the source of interference was indeed external, and that each instance of the interference appeared as a series of strong, periodic pulses in the intermediate frequency (IF) stage of the ISI hardware. The source of the interference was not, however, in the 3.7 to 4.2 GHz frequency band used by the ISI receiving equipment.

The source of the interference was found to be radio altimeters on aircraft flying over the Los Angeles International Airport at about 20,000 feet. The ISI earth station antenna, a 5 meter parabolic dish, is aimed to the southeast, directly over the airport. Like all satellite antennas, it is very sensitive to signals within a very narrow beam angle (1.1 degrees). Even though the radio altimeters transmit in another frequency band (an adjacent band at 4.2 to 4.4 GHz), the radio altimeter signals are so strong because of their proximity that harmonic mixing creates signals within the IF frequency of the ISI receiver. The aircraft took approximately 5 seconds to traverse the antenna's beam, hence the 5-second duration of the interference.

The radio altimeters were operating legally, and the interference was due solely to severe overloading within the ISI earth station hardware. Similar interference has occasionally been noted at other Wideband sites, some of which are also near airports.

As a temporary solution, the ISI WBC project installed a tunable bandpass filter (borrowed from COMSAT) before the ISI downconverter, allowing signals in the satellite band to pass through undisturbed while rejecting signals in the radio altimeter band. ISI built a monitoring device and confirmed that the interference was eliminated. Several bandpass filters are currently being evaluated and the best one will be selected and installed at all Wideband sites.

7.4.2.3 Earth station relocation

The Wideband Network earth station at ISI, which consists of the satellite transmitting and receiving equipment, was originally installed in an outdoor enclosure at the base of the ISI antenna. The earth station needs to be located as close as possible to the antenna itself in order to avoid signal loss in

the cables and waveguides that connect the equipment and the antenna. The enclosure was a fiberglass–covered, insulated wooden structure about the size of a small refrigerator, with a built–in air conditioner.

After the initial installation, other equipment was added to the earth station and the transmitting amplifier was replaced twice with a larger, more powerful model. Keeping the equipment at the proper temperature became more and more difficult, and thermal sensors often shut the equipment down, taking ISI off the air. In addition, the moist, salt–laden ocean air was starting to corrode the equipment.

It was apparent that the situation was resulting in excessive downtime for ISI and an excessive need for maintenance and attention from the local Western Union staff. Two alternatives were examined, either installing the equipment in a larger, better controlled shelter at the base of the antenna, or moving it into the ISI computing facility, which had since been expanded into the area on the floor immediately under the antenna. The second alternative was chosen, and ISI made the necessary construction arrangements and provided space in the ISI computing facility. In March 1983, Western Union moved the equipment inside. The flexible waveguide and standard coaxial cables connecting the earth station to the antenna were replaced with solid waveguide and "Heliax" cable to avoid additional signal losses caused by the longer signal paths. The equipment has been much more reliable in its new location, and the result has been increased Wideband Network availability at ISI.

7.4.2.4 Antenna refocusing

It had been suspected for some time that the 5 meter parabolic dish antenna on the roof at ISI, as well as the antennas at the three other original Wideband Network sites, was not performing up to specifications because of insufficient manufacturing quality control. Tests run with the antenna at Lincoln Laboratory indicated that the antenna was indeed not up to specifications. Although the manufacturer was willing to replace the antennas at no cost, the replacements would have required expensive modifications to the antenna support structures at the sites.

A simple compromise was chosen: to fine–tune each antenna by changing the position of the secondary reflector to compensate for the slightly non–parabolic shape of the antenna. The antenna adjustment (about an inch), made at ISI in June, resulted in an increase of 1 dB in both transmit and receive gain. Transmitter output power was decreased by 1 dB, to maintain constant amplitude at the satellite. A significant improvement in the bit error rate was noted.

7.4.3 Packet Video

The objective of the packet video work at ISI is to determine if there is a good match between the needs of a video transmission system and the capabilities of packet–switched networks, as was the case with voice. The Wideband Network is the first packet–switched network which can support such an experiment.

Digital transmission of video signals is nothing new. Neither are bandwidth compression systems which can reduce the data rate of a color video signal with moderate motion from the normal 100 Mb/s to 1.5 Mb/s or even less. However, such systems are designed to work with fixed–rate channels, are hard–wired to perform only one set of functions, and are very expensive.

The packet video experiments are based on the following facts and observations:

- The required data rate for video communication is the product of the spatial (x and y), shade and color (z), and temporal (t) resolutions used.
- The scene being transmitted will remain fairly constant, except for occasional sudden changes as the camera is zoomed, panned, or switched.
- The system must cope with sudden changes in channel performance (data rate, BER, delay, etc.).
- Both the video encoding and decoding equipment will have significant storage and processing ability.

Therefore, the packet video system must

- be as flexible and programmable as possible to handle new situations and to exploit as yet unknown opportunities;
- detect motion, or the lack of motion, and transmit as much or as little data as necessary;
- degrade gradually and gracefully, rather than suddenly and drastically, in response to decreasing channel capacity or increasing channel error rate.

These and related issues in satellite video communications are discussed in much greater detail in [5].

The ISI Wideband Communications project is engaged in designing and building a programmable system for packet video communications. Figure 7-1 is a block diagram of the ISI system. The hardware is based on a commercial video I/O/frame buffer system built by IKONAS (a subsidiary of Adage, Inc.), chosen because of its modular structure, which is based on a wide, fast, backplane bus. The ISI-built hardware consists of the Video Engine, a two-card set which performs the block transform processing in real time, and a high-speed serial interface for input and output of coded image data.

The following steps have been accomplished to date:

- A literature study was conducted and a candidate family of algorithms (block transform coding using the Discrete Cosine Transform, or DCT) was chosen.
- IKONAS raster graphics systems were acquired and installed at ISI and Lincoln Laboratory to provide the foundation on which to build the programmable video bandwidth compression hardware.
- The Packet Video Protocol (PVP) was designed and published [3].
- Software simulation of the hardware and the initial algorithms is almost complete.
- The hardware design is nearly complete.

7.4.3.1 The Video Engine

The key feature of equipment for packet video experimentation must be *programmability*. Flexible, programmable hardware minimizes the number of unalterable decisions which have to be made and which may later prevent the system from taking advantage of new opportunities. In addition, a programmable system allows a relatively simple system to be built fairly quickly, with advanced features added later. Unfortunately, none of the commercially available hardware for real-time video processing is programmable.

Therefore, the ISI WBC project chose to use a commercial system (the IKONAS system) to handle video input and output and frame buffering, and to design and build additional hardware to do the real-time video bandwidth compression and high-speed serial I/O tasks. The additional ISI-built hardware is called the Video Engine.

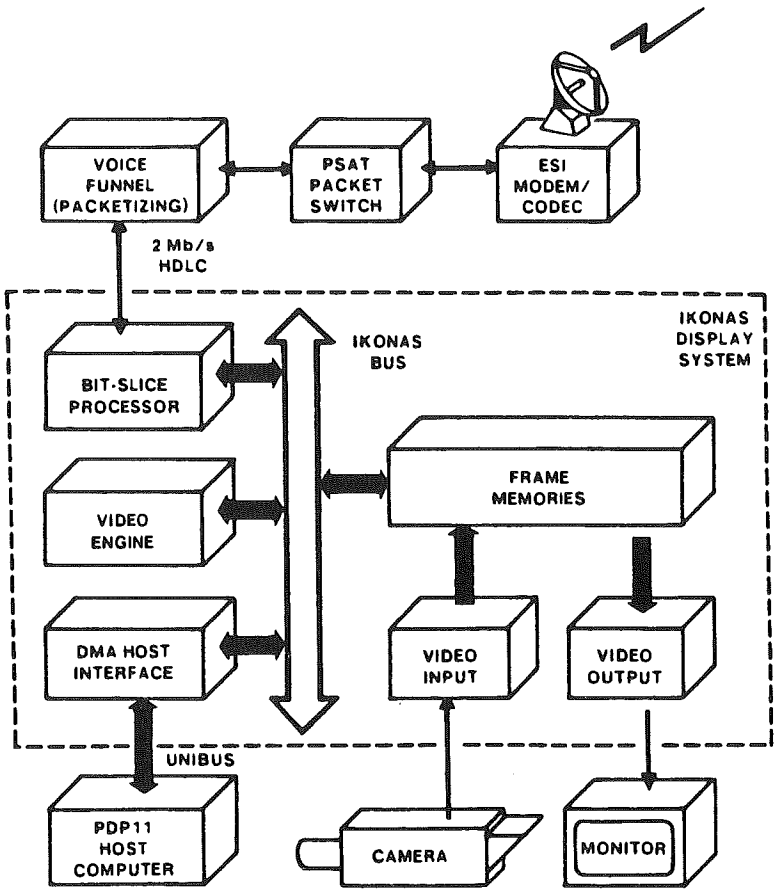


Figure 7-1: Block diagram of real-time video system

The Video Engine is a single-instruction, multiple-data machine based on 8 Texas Instruments TMS 320 microprocessors. The TMS 320 is a 16-bit signal-processing-oriented microprocessor which executes one instruction every 200 nanoseconds, with a full 16x16 multiplier, a 32-bit accumulator, and a small (144-word) internal RAM. Thus the Video Engine operates at a rate of 40 MIPS.

The Video Engine is designed to be installed in the IKONAS/ADAGE RDS-3000 raster graphics system. The IKONAS system provides video input, frame buffering, and video output, and contains a high-speed 32-bit bus, which the Video Engine will use for its source and destination of video data. More than one Video Engine can be put on a single IKONAS system, but their computational throughput will ultimately be limited by the data rate.

The primary design goal of the Video Engine was to achieve maximum programmability and flexibility at a reasonable cost. The conventional architectures for abundant processing power for video applications are based either on bit-slice microprocessors or on dedicated, special-purpose logic.

Since the Video Engine has to be programmable, a dedicated, special-purpose (i.e., inflexible) processor was not considered. Bit-slice microprocessors can be used to achieve considerable computing power at a fairly reasonable cost, but they are somewhat difficult to program and can achieve at best about 5 to 10 MIPS per processor if flexibility is to be retained.

Microprocessors designed for signal processing, such as the TMS 320 and the NEC 7720, are fairly new. The cost per unit is fairly high, currently about $150, but it can be expected to decrease rapidly. The 320 and the 7720 are easier to program than a bit-slice microprocessor, though harder than an ordinary micro. Their architecture is fairly primitive compared to that of a conventional micro, but they are more than an order of magnitude faster.

Since algorithms for video and other real-time signal processing are generally parallel in nature, a parallel architecture with many identical processors is a good fit to the task. In addition, such an architecture is much more suitable for eventual VLSI implementation. For these reasons we chose the multi-microprocessor architecture.

We chose the TMS 320 over the NEC 7720, because it can operate from a fairly large (4K) external program RAM, compared to the 512-word internal ROM or EPROM used by the 7720. It is slightly faster than the 7720 for transform-oriented programs.

Figure 7-2 is a block diagram of the Video Engine, which has two main components:

1. a RAM-based sequencer to control data flow and supervise the operation of the 320s;
2. the eight 320s themselves, along with an external buffer memory and an external table memory for each, and a single external program memory.

The sequencer uses a 4K by 40-bit RAM for control, and operates with a 100ns cycle time. In addition to controlling the 320s, the sequencer is responsible for reading and writing pixel data via the IKONAS bus. The 40-bit sequencer microcode word is horizontal, i.e., the bits in the word directly control the hardware and the fields are not decoded. A single-level subroutine capability is provided.

The Video Engine is designed to operate on blocks of data, typically 8 by 8, 8 by 16 or 16 by 8, 16 by 16, etc. A typical application would allocate a different block to each of the eight 320s. Blocks can be overlapped, but the IKONAS is structured to read groups of 8 pixels in a row most efficiently, and therefore blocks always start on 8-pixel boundaries horizontally. No such limitation exists vertically.

The sequencer addresses a pixel by taking a frame start address, adding a block offset to it, and adding a pixel offset to the result. The block and pixel offsets are fairly large, so a block could consist, for example, of every nth group of 8 pixels horizontally and/or every mth pixel vertically. The block offset is larger than a frame, allowing for even more flexibility.

Each 320 has a buffer RAM of 4K 16-bit words associated with it. The buffer RAM smooths out the bursty nature of the processing, allowing the 320s to run at full speed, and augments the small RAM in the 320s. Buffer RAM contents are transferred to and from the bulk frame memory under control of the sequencer, and transferred to and from the 320s via their I/O ports. Each 320 also has a table RAM (4K 8-bit words) which can be used for storage of coding and decoding tables, etc. This allows each 320 to code and decode different data, even though all the 320s are executing the exact program code simultaneously.

Only one of the 320s has a program memory associated with it; the other seven 320s are synchronized to that one. A simple memory mapping mechanism allows the use of 16K words of program memory, compared to the normal 4K word maximum. This mechanism allows code for several algorithms to reside in memory at once, perhaps allowing the use of different algorithms for different channel error rates, etc.

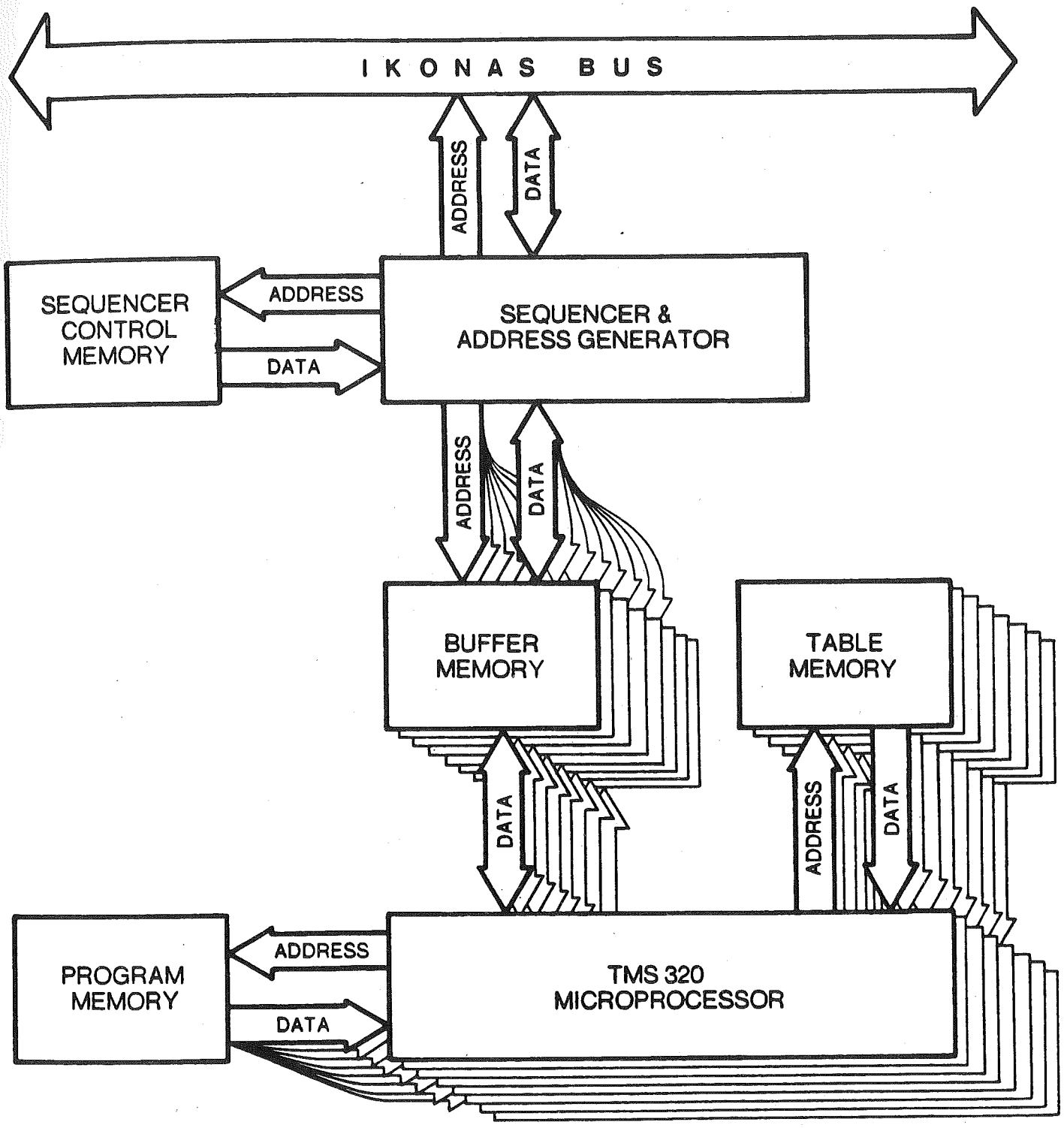


Figure 7-2: Block diagram of Video Engine

The Video Engine consists of a set of two cards, with the sequencer on one card and the 320s and their associated memories on the other. Power consumption is in the 100-watt range, but the IKONAS is able to supply adequate power and cooling. Logic is Schottky and low-power Schottky. The memories are 4K by 4 high-speed (55ns) static RAMS.

To help in hardware and software testing, all of the memories in the Video Engine can be loaded and read from the IKONAS bus. The IKONAS system is installed on a PDP 11/45, and the 11/45 or other host is responsibile for loading and starting the IKONAS/Video Engine.

7.4.3.2 Hardware and algorithm simulation

To support the packet video effort, it is necessary to simulate (1) the video bandwidth compression algorithms and (2) the video processing hardware. In the past year, the two simulation programs have been integrated, thus producing an accurate simulation of the complete video processing system. The resulting simulation package, called DCTSIM, pays particular attention to the simulation of interfaces between the components of the system.

Simulation of the video bandwidth compression currently implements the initial algorithm to be used in the real-time system, and transforms and encodes blocks of 8 by 8 pixels. The resulting compressed image has between one and two bits per pixel. When images coded to two bits per pixel are decoded and inverse transformed, the resulting image is virtually identical to the original. Coding to one bit per pixel causes noticable degradations in the reconstituted image, with quality increasing with the number of bits per pixel. The algorithm is currently being "tuned" to achieve a compression to somewhere between one and two bits per pixel without noticable degradation.

The ISI WBC project is currently extending the simulation to blocks of different sizes, such as 8 by 16, 16 by 8, and 16 by 16 pixels. Larger blocks should allow greater compression with the same output quality as the 8 by 8 pixel blocks. In the near future, more sophisticated coding schemes will be implemented and evaluated on the simulator before they are put into the real-time system.

The other function of DCTSIM is to test and develop the sequencer portion of the video processing hardware. The simulator is functionally equivalent to the hardware, and the code that drives the simulator is exactly the same as that which will drive the real-time hardware. The simulator has already proved very useful in assessing the functionality of the real-time hardware and pointing out errors and omissions in the design.

7.4.3.3 Hardware additions

Earlier this year, a second IKONAS/ADAGE video system was purchased and installed at Lincoln Laboratory in preparation for building a duplicate real-time system there. As soon as the real-time hardware is debugged and running at ISI, a second set of boards will be built and installed in the machine at Lincoln Laboratory. Even before the real-time system is running, frame-at-a-time video will be exchanged between ISI and Lincoln Laboratory to test and debug the video protocol and the supporting software.

Both IKONAS systems have been equipped with a bit-slice processor (BPS), which is a high-speed microprogrammed machine designed to handle internal processing chores for an IKONAS system. Although the BPSs are too slow to perform the block transform function, they will provide additional processing power for control of the entire real-time system, and they will supervise the input and output of coded image data.

7.5 IMPACT

DARPA-sponsored work continues to have a major impact in the packet voice area. Most high-speed local area networks (LANs) have incorporated packet voice, although most use it to store voice messages rather than for interactive communication.

The Wideband Network is a jointly sponsored effort of DARPA and DCA. In addition to the original DCA Wideband Network site at DCEC, three additional sites (Fort Monmouth, New Jersey, Fort Huachuca, Arizona, and the Rome Air Development Center in Rome, New York) have been established and are undergoing tests. The new sites will be an important mechanism for transfer of packet voice technology to the services. Personnel from one of the new sites have consulted with ISI WBC project members about providing video capability at some of the new sites. The ISI Packet Video Protocol (PVP) [3] could be used to support such an effort.

An architecture like that of the Video Engine should be well suited to any processing task that can take advantage of parallelism but does not require a large amount of code. Signal processing applications in general fall into this category.

The TMS 320, in particular, has had a big impact on the market, and improved versions of the 320 by Texas Instruments and similar chips by other manufacturers are certain to follow. The Video Engine is perhaps the first processor that attempts to use this type of chip in a highly parallel form. Such an architecture is likely to have a significant impact in the future.

7.6 FUTURE WORK

7.6.1 Packet Voice Experiments

Experiments with packet voice on the Wideband Network will continue into the foreseeable future, with new features and capabilities added, tested, and put into everyday use. ISI will continue to participate in the further development of Wideband packet voice, and will also continue to participate in efforts such the Task Force to upgrade the overall reliability and utility of the network itself.

To increase the quality and utility of the STN interfaces, echo cancellers will be built and retrofitted to the STN interfaces.

7.6.2 Packet Video

The immediate goal of the packet video effort is to get the initial real-time implementation up and running. Since the approach of the packet video effort is to start with a basic capability and steadily refine it, future packet video work will consist of increasing the performance of the video system by finding and exploiting ways to use the unique characteristics of the Wideband Network. In parallel with these experiments, the packet video system will be extended to handle color.

In addition to the ISI-developed real-time video system described in this report, the ISI WBC project will perform experiments with another video system which has been developed for DARPA. This system is a novel teleconferencing system which operates at 19.2 Kb/s and produces binary images resembling sketches of the conference participants [2]. A conference participant sits facing a bank of several camera/monitor units, one for each of the other participants in the conference. Each of the

monitors displays a sketchlike head-and-shoulders view of one of the other conference participants. The corresponding camera "sees" a view of the local participant from that monitor, preserving the spatial sense of the conference.

Integration of the 19.2 Kb/s system into the Wideband Network should be fairly simple, since the software in general, and the Packet Video Protocol in particular, is designed to be flexible. Current plans call for installation of this system at the MIT Laboratory for Computer Science and the DARPA office in Arlington, Virginia.

REFERENCES

1. WBC Project, "Wideband Communications," in *1982 Annual Technical Report*, USC/Information Sciences Institute, SR-83-23, 1982.

2. Brody, H., "Reach out and see someone," *High Technology*, August 1983, 53-59.

3. Cole, E. R., PVP – A Packet Video Protocol, USC/Information Sciences Institute, W-Note 28, August 1981.

4. Gitman, I., and H. Frank, "Economic analysis of integrated voice and data networks: A case study," *Proceedings of the IEEE* 66, (11), November 1978, 1549-1570.

5. Casner, S. L., D. Cohen, and E. R. Cole, "Issues in satellite packet video communication," in *Conference Record of the IEEE International Conference on Communications*, IEEE, Boston, 1983. Also available as USC/Information Sciences Institute, RS-83-5.

8. VLSI

Research Staff:
George Lewicki
Danny Cohen
Vance Tyree
Joel Goldberg
Ron Ayres
Barden Smith
Yehuda Afek
David Booth

Support Staff:
Victor Brown
Victoria Svoboda
Jasmin Witthoft
Lee Magnone

8.1 PROBLEM BEING SOLVED

The VLSI design communities of DARPA and NSF require fabrication capabilities in order to investigate the design methodologies and architectures appropriate to VLSI where gate–count will exceed one million gates per device. Recognizing this need, DARPA established the MOSIS (MOS Implementation Service) system at ISI in January 1981. MOSIS has met its design objectives:

- reduced the cost of VLSI prototyping;
- shortened turnaround time for VLSI prototyping;
- freed designers from fabrication idiosyncrasies; and
- made design less dependent on specific fabrication lines.

A cost reduction of one to two orders of magnitude has been achieved by spreading the fabrication cost over many projects. By centralizing (and computerizing) the idiosyncratic knowledge about all vendors, MOSIS eliminates the need for designers to familiarize themselves with many details. Serving as the only interface between its design community and the vendor base, MOSIS is able to provide turnaround times of four to six weeks for standard technology runs, except when unusual fabrication problems occur. Nonstandard technologies and experimental runs generally require longer fabrication schedules.

8.2 GOALS AND APPROACH

MOSIS involves various aspects of multiproject wafer assembly, quality control, and interaction with industry, as shown in Figure 8-1.

The major components of the MOSIS system are

- interaction with the designers;
- handling of their design (CIF) files;
- communication over either the ARPANET or TeleMail;
- placement of projects on dies, and dies on wafers;
- matching of MOSIS design rules to specific vendors' design rules, addition of alignment marks, critical dimensions, and test devices;
- fabrication of E–beam mask sets (via subcontract);
- fabrication of wafer lots (via subcontract);

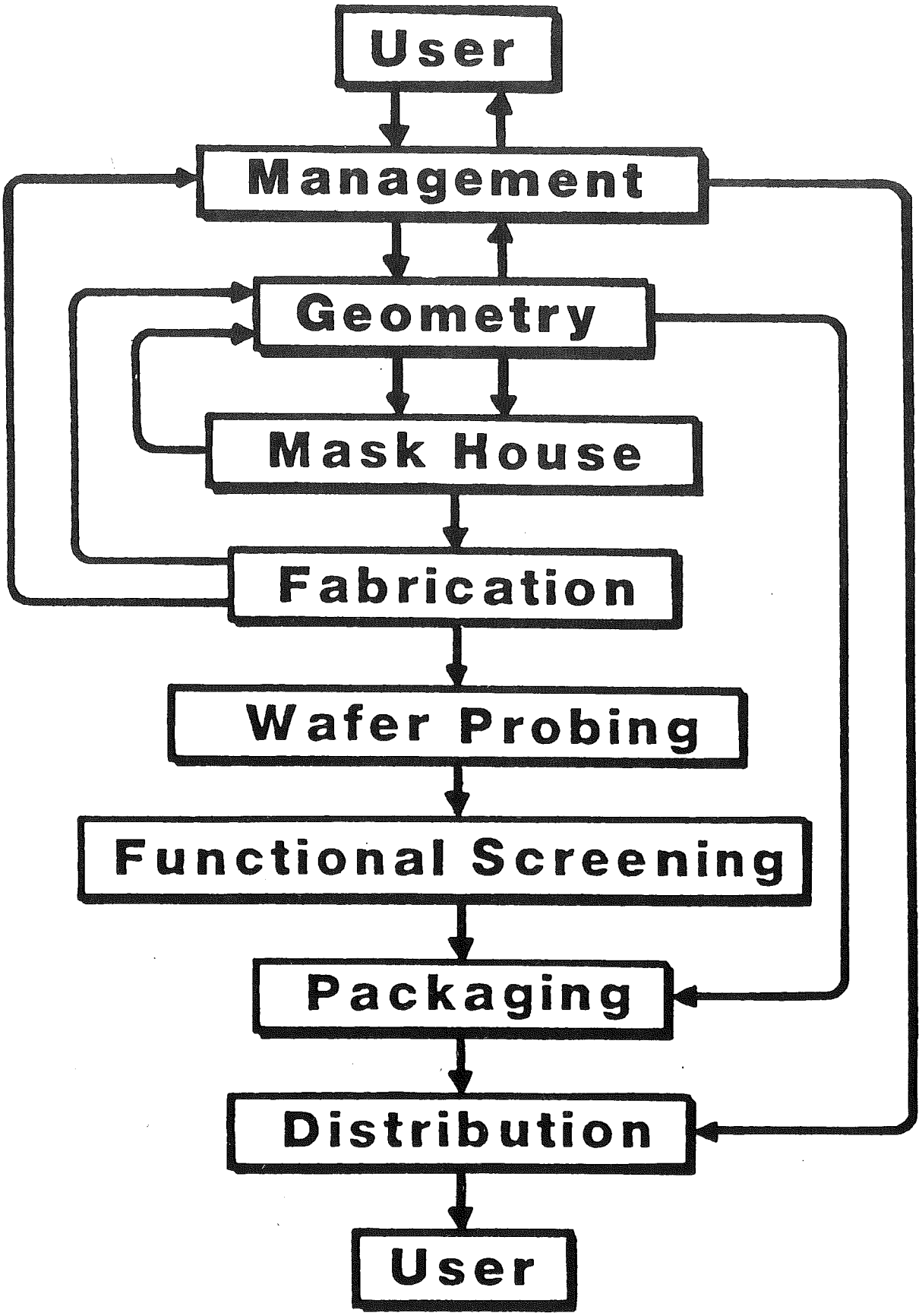


Figure 8-1: General flow of the MOSIS system

- wafer probing and data analysis;
- generation of bonding maps;
- wafer sawing, die packaging, and bonding (via subcontract);
- device distribution.

Designers use any available design tools to create artwork files that are sent to MOSIS via the ARPANET or other computer networks. MOSIS compiles a multiproject wafer and contracts with the semiconductor industry for mask making, wafer fabrication, and packaging. MOSIS then delivers packaged IC devices to the user. The user perceives MOSIS as a black box that accepts artwork files electronically and responds with packaged IC devices, as shown in Figure 8-2.

Though MOSIS may be instrumental in providing cells and design tools to the user, it is the sole responsibility of the user to see that the submitted patterns yield working designs. One may compare MOSIS to a publisher of conference proceedings compiled from papers submitted in "camera-ready" form, where the publisher's responsibility is to produce the exact image on the right kind of paper using the appropriate ink and binding—but not to address the spelling, grammar, syntax, ideas, or concepts of the various papers.

MOSIS provides a clean separation of responsibility for the "printing" of chips. The semiconductor manufacturer is responsible for the processing of the parts and must satisfy MOSIS's rigorous quality control. MOSIS is responsible to the user for the quality and timeliness of the fabrication. The user is responsible for the proper design of the parts and may use any design methods he finds appropriate for his needs.

It is quite common that very advanced and sophisticated chips fabricated by MOSIS work on "first-silicon." An example of this is Caltech's MOSAIC—this is an amazing accomplishment of the existing design tools. Unfortunately, this is done at a considerable cost; for example, it is estimated that Caltech's MOSAIC chip consumed over 1,000 CPU hours on various VAXes before it was submitted to MOSIS for fabrication.

8.3 SCIENTIFIC PROGRESS

8.3.1 Technology Base for Fabrication Runs

nMOS

MOSIS routinely supports nMOS at 3.0 and 4.0 micron feature sizes, with *buried*, rather than *butting*, contacts, in accordance with the Mead-Conway design rules. At least 20 vendors can fabricate devices according to these nMOS design rules.

CMOS/Bulk

MOSIS supports CMOS/Bulk (typically P-well) with 5.0, 4.0, and 3.0 micron feature sizes, usually with a capacitor layer and occasionally with a second metal layer. Most vendors support the MOSIS design rules for CMOS/Bulk, which were developed primarily by the Jet Propulsion Laboratory. However, not all of the vendors can support them at the 3.0 micron level.

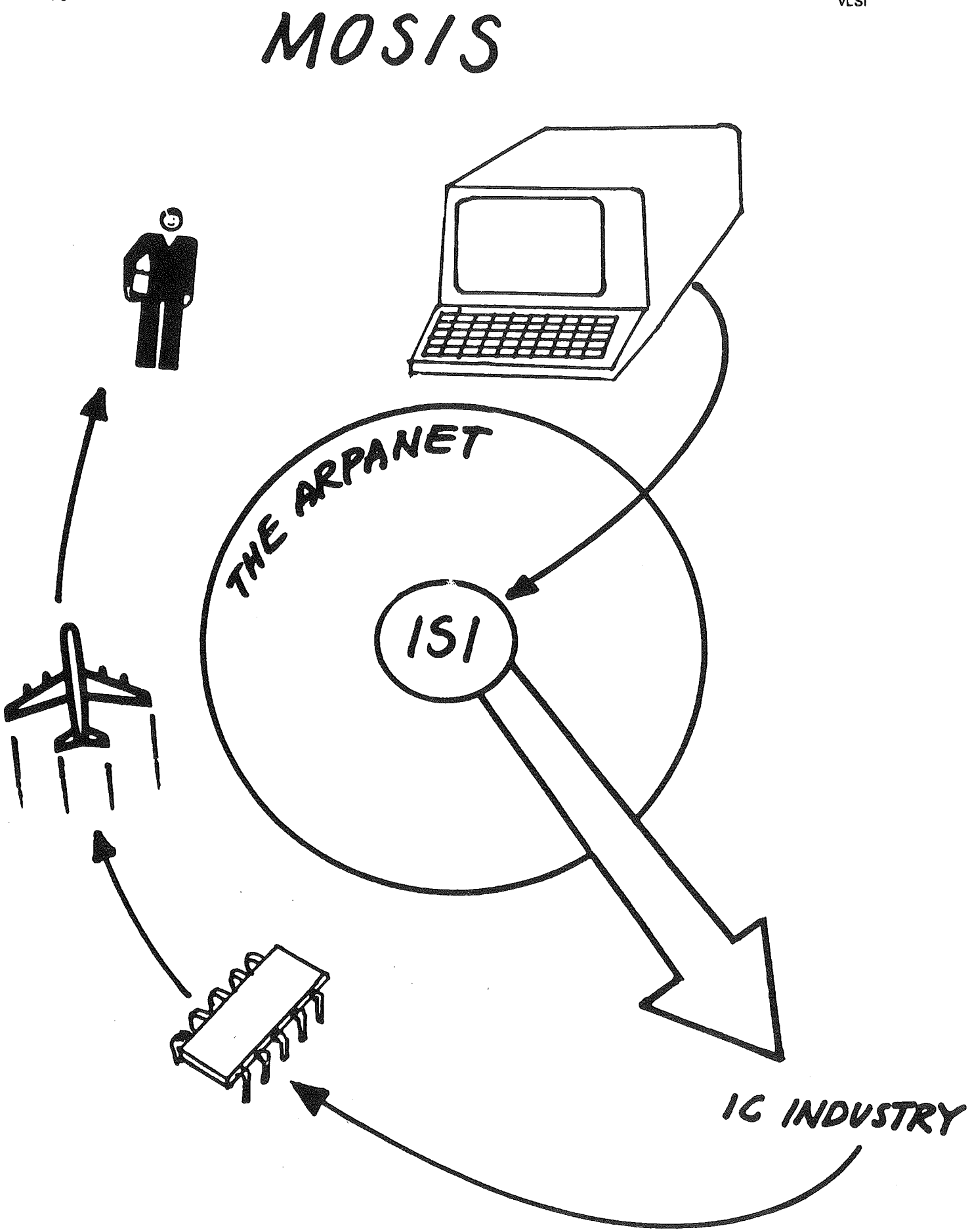


Figure 8-2: Interaction—user to MOSIS to user

CMOS/SOS

MOSIS supports CMOS/SOS fabrication with 4.0 micron feature size in accordance with the Caltech design rules. All of the SOS vendors support these design rules.

Completed Fabrication Runs

The following is a categoric breakdown by technology of the fabrication runs completed during this reporting period:

20 runs nMOS, 4 microns
3 runs nMOS, 3 microns
7 runs CMOS/Bulk, 5 microns
4 runs CMOS/Bulk, 3 microns
2 runs CMOS/SOS

8.3.2 Computing Interface

Since its inception, MOSIS has used KL-2060s (running TOPS-20) not only to communicate with users on the network, but also for its considerable computing requirements. It has been determined that VAX 11/750s would provide a more efficient and economical environment for computing. Substantial progress has been made in porting the MOSIS geometry software to the VAX 11/750.

MOSIS is on the verge of generating MEBES files on the VAX 11/750. In recent pilot tests, the MEBES files generated on a VAX 11/750 were identical to those generated on a KL-2060. Computing on VAX 11/750s has several benefits over KL-2060s:

- maintainability;
- extensibility;
- economy;
- single usage eliminates the burdens of shared resources;
- software implementation is more exposed.

The transition to VAX 11/750s will be transparent to MOSIS users; they will continue to access MOSIS on KL-2060s. The VAX 11/750 will be used primarily for geometry crunching, preparing MEBES files, and writing MEBES tapes.

The MOSIS software provides for placement and routing of parts of VLSI designs into fully operational, single, unified designs. Tentative measurements indicate that the software runs only a factor of two slower on a VAX 11/750 than on a KL-2060, significantly less than the projected factor of four.

8.3.3 Fabrication Interface

The MOSIS vendor base has expanded substantially during the reporting period. Increased user feedback and more extensive test results have allowed the MOSIS project to determine and communicate fabrication requirements to new vendors. This has resulted in higher quality wafers and the development of consistently reliable vendor sources for mask making and nMOS fabrication.

MOSIS has instituted procedures to manage the vast amount of information inherent in dealing with a multi-vendor base. Many administrative tasks have been automated, including the maintenance of templates to determine fabrication requirements specific to vendor and technology (a single vendor often provides several fabrication technologies).

Three micron CMOS/Bulk presents many complications that do not exist in nMOS: fabrication requirements vary considerably from vendor to vendor. We have been largely successful in standardizing the design rules for the users by deriving nonstandard designs from user-supplied designs, as in the case of extra layers. The availability of such fabrication options varies widely. For instance, a vendor may offer combinations of desirable options such as capacitors via an electrode layer and a second metal layer (capacitors, second metal, either but not both, or both). We anticipate that the standard 3 micron CMOS/Bulk technology will offer a choice of options, but not both. The availability of these options, coupled with increased volume and a diverse vendor base, necessitates continued automation of the fabrication interface.

The quality of parts fabricated in the CMOS/Bulk technology equals that of parts fabricated in nMOS. Further, CMOS/Bulk is a more desirable technology because of its capability for lower power consumption and higher circuit density. Therefore, MOSIS is preparing for the eventual shift away from nMOS by improving turnaround and vendor reliability for CMOS/Bulk fabrication.

8.3.4 Quality Assurance/Design Interface

Most MOSIS devices are prototypes without established functional testing procedures. Generally, the designers who receive these devices are still debugging the designs, rather than checking for fabrication defects introduced by less-than-perfect yield.

MOSIS's extensive quality assurance program is aimed primarily at the parametric level. This guarantees that the electrical properties of the wafers are within specifications established by the best a priori simulations used in the design process. Work has continued to increase the accuracy of the SPICE parameters which are made available to MOSIS users. SPICE provides simulated mathematical modes for behavior of transistors, allowing designers to assess a small digital circuit idea, to avoid faulty design, and to improve their chances of success in fabrication. The electrical criteria are a superset of the SPICE parameters at level II. They include a ring oscillator, which gives a rough idea of the speed of the resulting circuitry. The electrical properties of the wafers are extracted first by the fabricator, who uses either his own process control monitoring devices or the MOSIS test structures. Only wafers passing these tests are delivered to MOSIS.

It is a common practice in the IC industry to save functional probing time by probing wafers in only a very few sites. This practice makes sense, because all parts are subject to functional testing and because this parametric probing serves only to eliminate disastrously bad wafers.

Since designers hand-test most MOSIS devices, MOSIS requirements for parametric testing are higher than industry standards. MOSIS inserts its own test strip on every device, if space permits. When the test strips cannot be inserted, MOSIS probes a large number of test sites. This probing provides important statistics on the electrical properties and their distribution across the wafer. Most wafers have uniform distribution; some, however, have other statistical patterns, such as significant gradients and bimodal distributions.

These in-depth statistics are available only to the fabricators. Designers receive the general statistics (mean and variance) for each run. Interested users can request the specific values of the parameters extracted near any of their chips.

Users comparing the performance of actual chips to their simulations find it useful to rerun the simulation with the actual a posteriori parameter values extracted near that chip. The marking on each chip (which can be checked by opening the package lid and using a low power microscope) identifies the run in which the chip was produced and its position on the wafer. Along with the wafer ID marked on the package, these markings provide the identification needed to determine the most relevant probing site.

A perfect set of electrical parameters does not guarantee perfect yield, so there is always a need for functional testing. MOSIS does not have the facilities for high-speed functional testing, but can perform partial functional testing. This screening typically catches the majority of fabrication defects, such as shorts. The screening is performed by applying and reading user-provided vectors to each device before the wafer is cut; those failing the test will not be packaged. By screening the larger chips—which typically have lower yield and higher packaging cost, and are required in larger quantities—MOSIS significantly reduces the packaging cost.

8.3.5 Standard Pad Frame/Packaging

MOSIS's current packaging strategy is to package enough parts to ensure a 90 percent probability of delivering a defectless part to the designer. This strategy was acceptable when most of MOSIS's community was designing small circuits and the fraction of packaged defective parts was small. However, a significant portion of the community has successfully completed the development of large designs and now wants from 300 to 3,000 working parts to begin developing prototype systems based on parts obtained through MOSIS. The yield for these large designs is expected to be 25 percent at best. If MOSIS were to follow its current strategy of packaging parts without any testing to indicate functionality, it would be packaging four times the required number of parts to achieve a requested quantity. This becomes a serious problem as quantities increase. Packaging costs dictate a more economical approach.

To avoid such waste, MOSIS has worked with Stanford University to define a functional test language (SIEVE) and is developing hardware to effect the testing specified by that language. Users will soon have the option of submitting text describing limited test procedures to be used at wafer probe to screen out bad parts. The purpose of this screening is to detect the types of "trivial" defects that cause the majority of bad parts and, therefore, to reduce packaging costs. Full functional testing is expected to be done by the user.

For designs with custom pad layouts, it will be the responsibility of the designer to provide MOSIS with the custom probe card to probe his circuits. To eliminate the inconveniences associated with generating custom probe cards for every design, MOSIS is currently developing a set of standard pad frames, each specifying exactly where the pads are positioned. MOSIS will then stock probe cards for each of the frames.

These standard frames are also expected to facilitate packaging. Bonding diagrams for projects currently submitted are generated manually, because several attempts to automate this process have met will only limited success. Bonding diagrams instruct the packager to connect a specific pad on the chip to a specific pin on the package. Standard pad frames will have standard bonding diagrams, eliminating the the need to generate a new diagram for each project.

Standard frames also allow the bonding process itself to be automated. Automated, programmable bonding machines are currently available. Standard pad frames make possible a scenario in which an operator would identify a first pad and package pin; programmed information would then control the bonding on a chip.

Automating the packaging phase could significantly improve MOSIS's turnaround time. The best times experienced by MOSIS for various parts of a run in nMOS are (1) four days to convert submitted geometry into a format acceptable by a mask maker and to generate the first masks required by a fabricator to start a run; (2) five days for wafer fabrication; (3) one day for wafer acceptance; (4) one week for packaging; and (5) one day for overnight delivery of packages. Standard pad frames with automated bonding could reduce packaging time to two or three days and reduce the time from submission of designs to mailing of packaged parts (in very conventional technologies) to approximately two weeks.

8.4 IMPACT

MOSIS's main function is to act as a single interface ("silicon broker") between a geographically distributed design community and a diverse semiconductor industry. As such an interface, MOSIS has significantly reduced the cost and time associated with prototyping custom chips.

The greatest impact of MOSIS, however, is in the community it has created. The MOSIS user community shares not only the fabrication services, but also experience, cells, tools, and software. The rapid growth of the community proves that the services provided by MOSIS are useful and important to both the academic and the R&D communities.

8.5 FUTURE WORK

8.5.1 Printed Circuit Boards

MOSIS will soon offer the fabrication of Printed Circuit Boards (PCBs) to its users. MOSIS will treat PCBs as just another technology, surprisingly similar to nMOS and the various dialects of CMOS. Although PCBs and ICs are made of different materials, they share a common method of specification of the images required on each of their layers. Both are fabricated by a "photographic" process which transfers images to the media surface from a master tooling.

This approach of using common tooling preparation methodology for both ICs and PCBs has many advantages: it allows designers to use common tools for the design process (with details tailored specifically to each technology) and allows MOSIS to apply the same management procedures and the same geometrical processing and tooling preparation methods to PCB technology.

Probably the most important feature of the expansion of MOSIS to include PCBs is that the expansion will be done in a way which is expected to carry over to other packaging technologies, such as various hybrids, ceramic carriers, and plastic tapes.

8.5.2 CMOS/Bulk

CMOS/Bulk is the emerging VLSI technology. MOSIS has adopted the philosophy that 3 micron CMOS/Bulk is going to be the work horse of the design community for the next ten years. Only in special cases will designers opt for a finer feature size, expected to be 1.2 microns. MOSIS plans to rapidly expand its vendor base for 3 micron CMOS/Bulk based upon a set of design rules already in hand.

MOSIS is also taking steps to position its design community to exploit the 1.2 micron CMOS/Bulk technology that is being developed at several commercial and industrial laboratories. Towards this goal, MOSIS will be supporting a number of investigators in their development of rules and design techniques for 1.2 micron CMOS/Bulk, with wafer fabrication carried out by four vendors. The idea is to develop these rules and techniques in parallel with the development of the technology, allowing the entire MOSIS community to use this technology upon the completion of its development. This strategy will narrow the gap between the availability of a new technology and the ability to use it.

9. OFFICE ENVIRONMENTS

Research Staff:
Don Chadwick
Steve Brown
Tom Capalety
John Diniakos
Brian Donnelly
Bernice Glenn

9.1 PROBLEM BEING SOLVED

As the man–machine relationship becomes an integral part of our lives, and the computer an extension of our minds, our environment must facilitate the use of hardware while offering a human dimension. The Office Environments project developed a plan for a workspace which offers physical ease and comfort, and allows for a high degree of performance.

Current research in related areas was analyzed to help identify major issues and to avoid duplication. Combining research, design development, construction, and testing, we produced a scaled prototype of an electronic office based on an existing ISI office area.

9.2 GOALS AND APPROACH

Several premises for the design of an electronic office were outlined. Issues concerning information management, job satisfaction, and productivity were studied to determine their relation to the physical environment:

- Electronic technology should enhance, not threaten, the worker's ability to make significant, creative, timely decisions and judgments.
- The introduction of new technologies which provide the workers with opportunities to be more autonomous and creative does not imply the need for designing isolated office environments, but instead environments which encourage human interaction.
- The measure of productivity in the office can no longer be attained through the quantitative industrial method of comparing input to output. A radically different measurement of productivity will imply substantial changes in the job design and function.
- The concept of a worker's participation in his environment is extremely significant; any successful design plan must reflect the inputs of the worker.

The research and development of the physical environment focused on the following areas:

- Furniture (seating, etc.);
- Work surfaces (desk, CRT);
- Storage needs;
- Lighting control;
- Temperature control;
- Privacy and noise control;
- Safety (wire management);

- Vision–related problems (glare);
- Range of computer capabilities;
- Capablity and functionality of all office elements; and
- Space needs for office interaction;

9.3 SCIENTIFIC PROGRESS

9.3.1 The Workspaces

Figure 9-1 is a plan view of the current ISI 11th floor office layout. The shaded area in the southeast corner is the 1600–square–foot area represented in the scale model. Presently, this area accommodates seven people, each in a conventional private office.

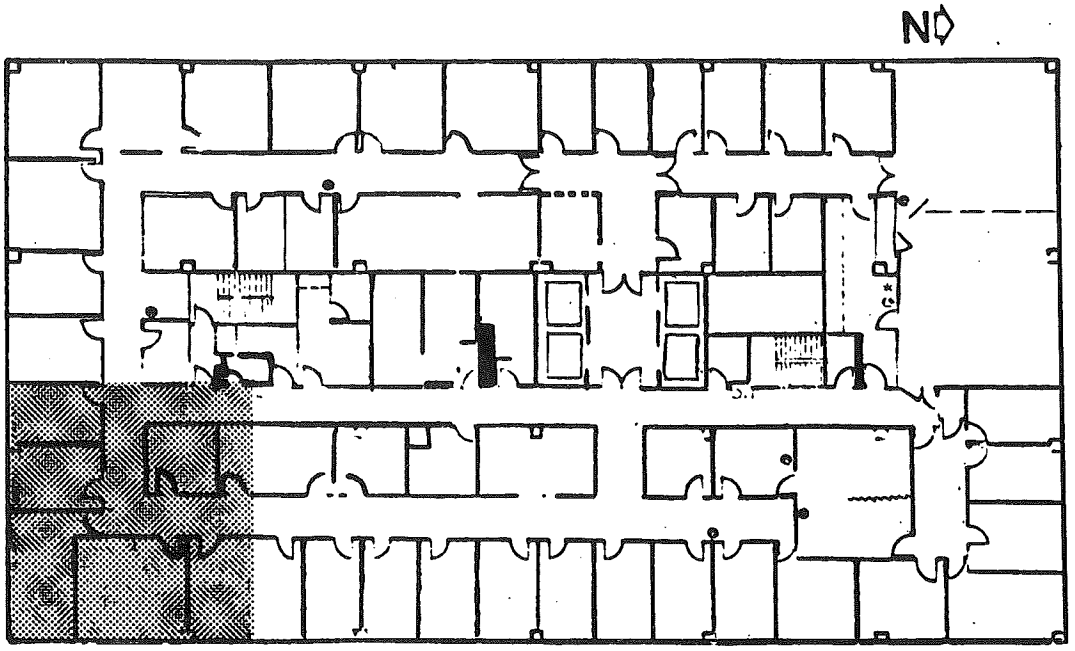


Figure 9-1: Current ISI floorplan with area represented by scale model

We examined the individual design of each workspace within the context of the total office environment. We began by eliminating all existing interior walls, planning the layout of the wall system to house the workspaces. We created eight individual workspaces and a social/conference area. We worked to create an environment which maximizes comfort, efficiency, and productivity.

All persons interviewed agreed on the necessity of control over their work areas on many levels, particularly audio and visual privacy. Thus, in contrast to the "open plan" and component wall system (typically low dividers and thin partitions), we proposed four–inch–thick walls, solid doors, and frosted and clear glass windows, all used in recognizable architectural scale.

9.3.2 Wire Management

In Figures 9-2 and 9-3, a three-channel communications and power cable raceway is visible along the wall and under the workstation top. This "power track" houses lines for data, communications, and electrical power along the ceiling, walls, or floor, distributing them to an individual work area. There is access to the track at any point. In addition, the power track acts as a horizontal support element of the work surface. The communications and power cables enter the workstation either at a full floor-to-ceiling column (Figure 9-3) or from the floor into a worktop-height column (Figure 9-2).

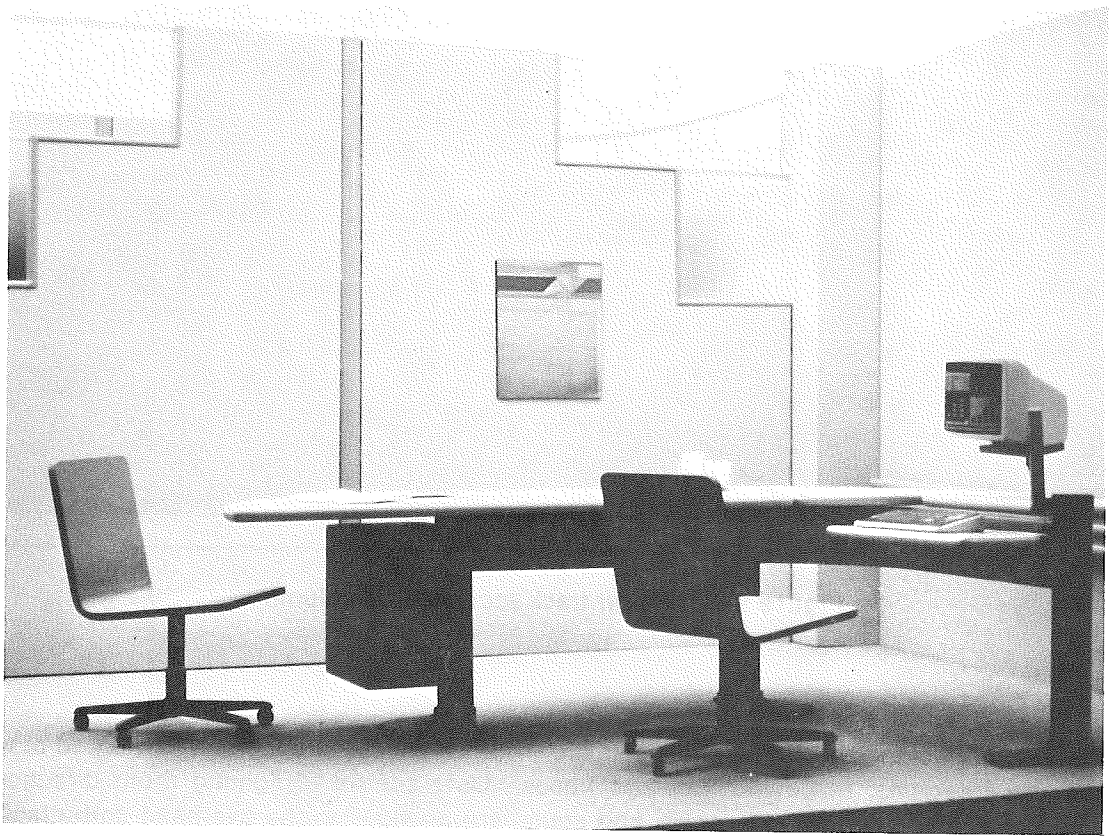


Figure 9-2: Power track access to worktop-height column

The column arrangement selected depends on the type of communications and electrical system employed in the office building and the needs and desires of the specific user. These columns also support various storage units and articulated arms, which in turn support keyboards and visual display units.

9.3.3 Physical Storage

Based on our research, the majority of paper information tends to be pending work. Most existing offices negate this notion by providing minimal display and a large amount of hidden storage (drawers, file cabinets). Yet office workers are acquiring more paper with no adequate way of accommodating it. Our plan offered various display spaces on worktops.

Figure 9-3: Power track access to full column

9.3.4 User Control

The development of the electronic office sets the stage for the emergence of a new breed of worker who has new freedoms, responsibilites, and needs. User control of lighting (both direction and intensity), air ventilation and temperature, and sound are provided in the overhead unit attached to the column. The controls are accessible from a seated position (Figures 9-3 and 9-4).

Articulated arms that support keyboards and visual display units are flexible over a given three-dimensional area. Muscular-skeletal fatigue and visual fatigue, caused by improper positioning and screen glare, are eliminated by a combination of adjustable lighting and arms.

Figure 9-4 also depicts the use of frosted and clear glass in certain areas of the interior walls, creating a private office that is connected to the larger environment.

The work area in Figure 9-5 opens to the social/conference area, which is the hub of the office environment. Also evident is the worktop-height column that comes through to the surface, allowing easy access for telephones or other devices requiring cord connections.

Figure 9-4: Use of clear and frosted glass in creating interior walls

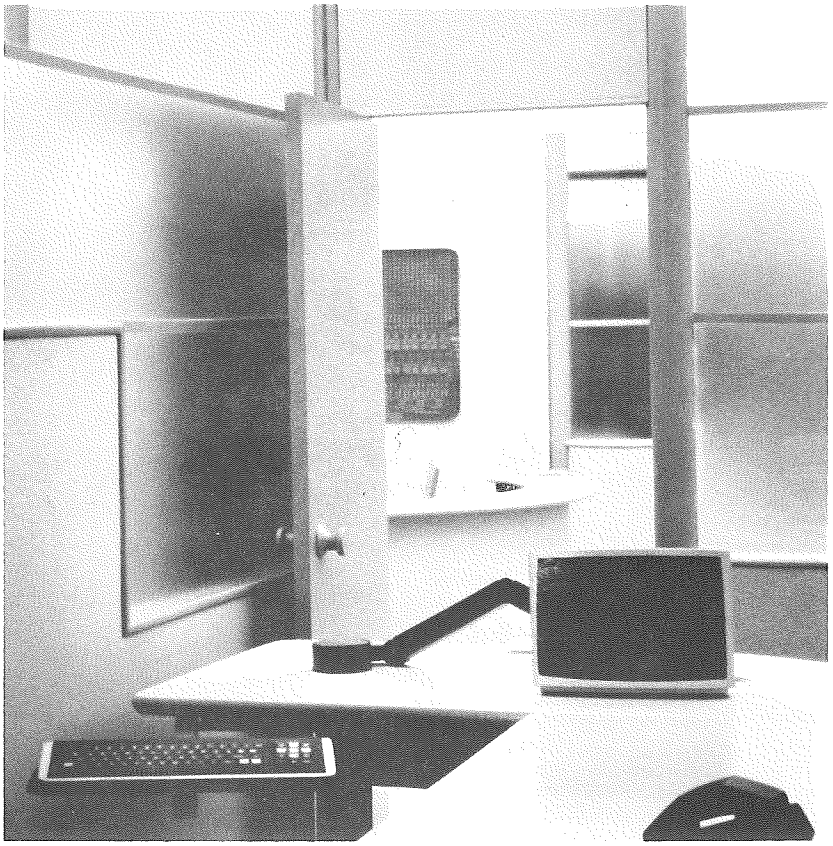


Figure 9-5: Work area with access to social/conference area

Figure 9-6: Social/conference area

Figure 9-7: Project leader's office

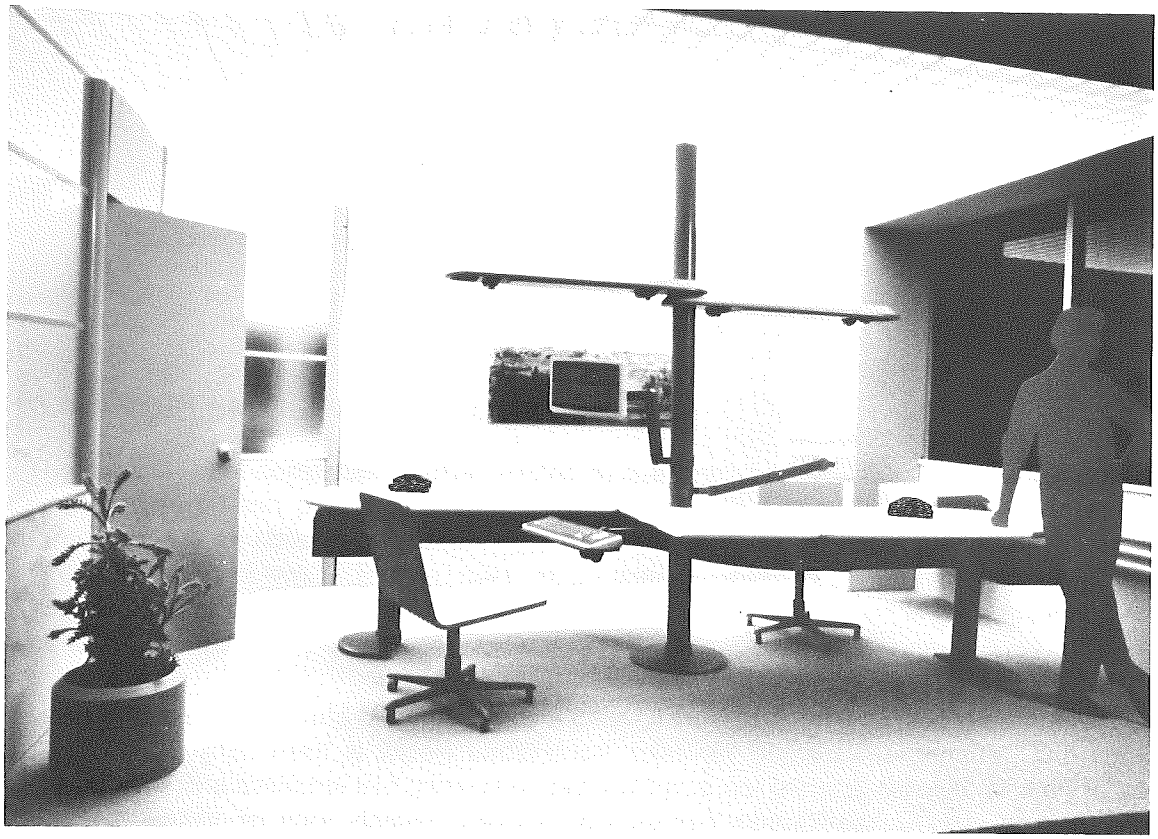


Figure 9-8: Shared office workstation configuration

9.3.5 Quality Human Interaction

Our research pointed convincingly to the notion that people prefer to work in an office setting rather than at home. There is a desire to be involved in an active and stimulating environment. The social/conference area (Figure 9-6) satisfies some very important needs of the electronic office worker. It is a casual area that anticipates informal meetings; the tables accommodate keyboards and visual display units and can display images and text on the larger wall screen.

Figure 9-7 is an office for a project leader, who has frequent meetings and work sessions with others. This work area provides true team interaction, incorporating teleconferencing equipment and a large display screen on the wall. We also designed a shared office workstation configuration (Figure 9-8) for those who find it advantageous to work in a shared space (project support, part-time researchers, graduate students).

9.4 IMPACT

Design of a complete office environment encompasses a complexity of interrelated issues. Research has resulted in recommended design dimensions for specifics such as keyboard height, visual display viewing distance, screen viewing angles, and work surface height. These specifications were researched during the first phase of the project and were incorporated in the second phase proposal.

The second phase proposal of this project, calling for full-scale construction of this plan, was not funded. Therefore, the impact of our design on an actual workspace is unknown.

10. KNOWLEDGE DELIVERY

Research Staff:
William Mann
Christian Matthiessen
Sandra Thompson
Michael Fehling
Michael Halliday

Support Staff:
Joyce Reynolds

10.1 PROBLEM BEING SOLVED

The usefulness of computers is often limited by their very poor ability to communicate with people. Computer users—and potential users—are often prevented from using information because that information is in an obscure, computer-internal notation. Training and experience can only partially overcome the problem. People who must interact with many computers, who need rapid understanding of changing situations, or who have only occasional needs for computer information, tend to be blocked by the computer notations.

Many of these limits could be removed by a general technique for expressing computer information in English. English itself is widely understood, relatively domain independent, and very flexible. Unfortunately, the available text generation technology is not general, reliable, or easy to apply. It is restricted mainly to very domain-specific, preplanned operations. The technology does not generalize well, does not combine sentences well, and does not transfer from one task to another at all.

A new technology of text generation is needed, one in which techniques can be refined and moved from one application system to another. Task-specific knowledge and techniques must be made separable from general techniques of English, with provisions for embedding the general portion in a variety of different application systems.

The problem is currently particularly critical in two areas:

1. Current systems can employ appropriately only a tiny fraction of the full expressive power of English sentences. Stronger capacity to control English grammar is needed.
2. There is a complexity limit on what can be expressed in computer-generated text. It is set by the difficulty of understanding intricate single sentences and by the inability of systems to combine sentences well. Computational methods for planning multiple coordinated sentences are needed.

10.2 GOALS AND APPROACH

A recent survey of text generation technology concluded that four critical technologies will pace development of text generation capabilities in this decade [1]:

1. Grammars of English
2. Models of Discourse
3. Knowledge Representation
4. Models of the Reader

Of these, the first two are most critical in the near term. They have a *producer–consumer* relationship: models of discourse are crucial to creating text plans, and grammars are crucial to executing text plans. Among past text generation systems, the ones producing the most fluent English have been those with large, linguistically justified grammars.

We have selected the Systemic Linguistics tradition as our basis for development of theory and programs.[1] Of the many linguistic frameworks available today, the systemic framework is used in the largest fraction of work in text generation—more than all others taken together. For text generation, it has the advantage of a strong orientation toward language function—what the language *does* for its user—and it contains functional accounts of a diverse spectrum of English syntactic constructions.

As represented in the linguistic and computational literatures at the beginning of this project, the systemic framework had several problems:

1. The notation, especially for realization (structure–building), was variable and somewhat informal.
2. Notational problems peculiar to actually creating large–scale grammars had not been addressed.
3. There was no semantic notation.
4. Although grammatical alternatives were well represented, *the issue of when to use which alternative* was not well addressed.

As part of our work on linguistically justified grammars we have addressed all of these problems, and we have complete solutions to all but the last one.

Regarding the problem of creating a computationally useful *model of discourse*, there were several sources of difficulty:

1. The literature from linguistics and related disciplines was (and is) vague and rudimentary.
2. The literature is oriented much more to narratives and stories than to the kinds of expository content for which text generation is needed most.
3. The available theories of text structure are oriented to describing existing text, not to saying how text is created.

Our approach to the discourse structure problem has been to discover and characterize the structure of expository texts, and then to augment the characterizations of text structure to include methods for using the structures in text creation. Of course, the approaches to discourse and grammar must be compatible: In addition to sketching the intended text structure, the discourse planner must produce the details the grammar needs.

The final phase of development is to apply the grammar, the text planner, and their associated domain–dependent processes to actual text generation problems. Such applications will test and refine all of the parts, including the domain–independent ones.

---

[1] Systemic Linguistics is an approach derived from work of Michael Halliday, begun in the late 1950's and still ongoing.

10.3 SCIENTIFIC PROGRESS

We have made significant progress, both in grammar development and discourse modeling, within a general design framework which encompasses both domain-dependent and domain-independent parts [3]. We have designed a text generation system, named Penman, to embody these developments.

We have accomplished several of our goals in grammar development:

- We have defined a precise, fully explicit notation for systemic grammars, including structure-building. The notation has been programmed (in Interlisp) as part of a large computational grammar named Nigel (this is a contribution to both computer science and linguistics) [6, 7].
- Available definitions of various fragments of the grammar of English have been gathered from the literature, reconciled, and expressed in the new notation [4].
- Gaps in the resulting coverage of English have been identified; many have been filled by the development of new regions of grammar [7].[2]
- The resulting syntax of English has been tested extensively for consistency and formal integrity. As part of the testing, integrity criteria for systemic grammars have been defined.
- A semantic notation for systemic grammars has been defined [2].
- This semantic notation has been applied to Nigel; new semantic definitions exist for over half of Nigel's choice points [5].

Nigel's semantic definition of English has revealed that ordinary English syntactic constructions can require information which is not commonly available in modern AI knowledge representations. Often, this type of information is not definitely ruled out in principle, but no demonstrated approach to representation exists for it. (For example, English provides modals such as *could* and *may*, which may be used to express either uncertainty or possibility; it provides a contrast between mass and count as kinds of reference; and it provides participant-oriented decompositions of actions (in clause structure), generalized possession (in its possessives); a broad variety of attributes (only partly in the numerous adjectives); reconceptualization of events as objects (in nominalization); and much more. For current knowledge notations, most of these are unsolved representational problems).

We see English (and all natural languages) as highly evolved knowledge representations that are well adapted for working with the kinds of problems and information that people create. Therefore, the fact that AI knowledge representations cannot cover the needs of English syntax means that the AI knowledge representations are unsuitable for a wide range of frequently encountered human problems. Nigel's semantics creates the necessary basis for two kinds of impact on AI:

1. identification of representational gaps in particular notations, and
2. setting priorities for new expansions of knowledge representation capabilities, filling those representational gaps.

---

[2] The resulting grammar appears to be the largest grammar of any natural language in any functional linguistic tradition.

We have accomplished the following in the creation of discourse models:[3]

- created a new descriptive theory of text structure;
- applied the theory to between 50 and 100 texts, mostly small edited texts from published sources; and
- developed a preliminary design for a text planner based on the theory.

In its constructive form, as a text planner, this theory specifies how to create a wide variety of different kinds of text. It insures that the text will be coherent. It also identifies a collection of inferences the reader can be expected to make, based on the text's structure above the sentence level. In effect, it includes a theory of how people will "read between the lines," which was missing from both prior computational work and prior linguistic work on written text.

As a result, the text planner will be able to decide what should be said explicitly and what can be left for readers to infer. Inferential expression is clearly a great source of efficiency in multisentential texts, but it has not previously been used because it is poorly understood in the literature. For text generation, identifying the inferential component has two consequences:

1. The generator can allocate part of what it is saying to inferential expression, which does not require its own sentences.
2. The generator can avoid saying things which would create undesired inferences and suggestions.

10.4 IMPACT

The impact of this work so far has been in computational linguistics, artificial intelligence, and linguistics. In computational linguistics, other research groups are seeking to use the Nigel grammar as a component in future research work on text generation systems. They have seen its definitional framework and its treatment of English as particularly attractive as a base for research in human–computer interfaces. In artificial intelligence, some of the representational gaps which Nigel has revealed are being studied with a view to filling them. In linguistics, Nigel's extension of the systemic framework is contributing to new studies of language function. All of these impacts have begun even prior to the release of definitive publications of Nigel's content. We expect that as Nigel is completed and published, its impact on these fields will be even greater.

Once Nigel is available as a research resource, researchers in text generation will be able to concentrate more attention on areas other than sentential fluency. Furthermore, Nigel is large enough that, when some significant gap in its grammar is found, it will be preferable to add the new construction to Nigel rather than starting over on a domain-specific grammar. Development of the grammar will become cumulative, with a resulting increase in research productivity simply because discoveries are retained and propagated. (Research in the field of text generation to date has been based principally on fresh starts rather than cumulative development.)

Part of the eventual nonresearch impact of the work will be to make computer interfaces easier to construct and use. Another part of the impact will be to make new kinds of computer applications practical—those which require easy expression of information to computer-naive people. This creation of new application possibilities may be even more significant than facilitation of existing ones.

---

[3]The discourse modeling work, which started after the work on grammar, is largely unavailable in documents. It has been the basis of part of a UCLA linguistics course, and we have published a preliminary report on one consequence of the theory [8]. Other documentation is in preparation.

10.5 FUTURE WORK

Future work on Knowledge Delivery is aimed at completing and integrating the existing developments in grammar and discourse, and on applying them in experimental text generators. Planned applications include description of bodies of numerical data, description of states of computer programs (especially AI systems), description of program execution, and explanation of program reasoning.

Although the usability of the new semantic notation has already been established, much work must be done to make the actual grammar of English more comprehensive and reliable. This work is best done by us at the source, as refinements by the supplier rather than repairs by consumers.

Work is also under way to assess Nigel's potential interaction with particular knowledge representations, including KL–ONE and the XLMS family, for use in experimental text generation applications.

REFERENCES

1. Mann, W. C., *et al.*, *Text Generation: The State of the Art and the Literature*, USC/Information Sciences Institute, RR–81–101, December 1981. Appeared as *Text Generation* in April–June 1982 AJCL.

2. Mann, W. C., *The Anatomy of a Systemic Choice*, USC/Information Sciences Institute, RR–82–104, October 1982. To appear in *Discourse Processes.*

3. Mann, W. C., "An overview of the Penman text generation system," in *Proceedings of the National Conference on Artificial Intelligence*, pp. 261–265 , AAAI, August 1983 . Also appears as USC/Information Sciences Institute, RR–83–114.

4. Mann, W. C., "A linguistic overview of the Nigel text generation grammar," in *Proceedings of the Xth International LACUS Forum*, Linguistic Association of Canada and the United States, Quebec, August 1983. Also appears as USC/Information Sciences Institute, RS–83–9, October 1983.

5. Mann, W. C., "Inquiry semantics: A functional semantics of natural language grammar," in *Proceedings of the First Annual Conference*, Association for Computational Linguistics, European Chapter, September 1983. Also appears as USC/Information Sciences Institute, RS–83–8, October 1983.

6. Mann, W. C., and C. M. I. M. Matthiessen, *Nigel: A Systemic Grammar for Text Generation*, USC/Information Sciences Institute, RR–83–105, February 1983. The papers in this report will also appear in a forthcoming volume of the *Advances in Discourse Processes Series*, R. Freedle (ed.): *Systemic Perspectives on Discourse: Selected Theoretical Papers from the 9th International Systemic Workshop*, to be published by Ablex.

7. Mann, W. C., and C. M. I. M. Matthiessen, *An Overview of the Nigel Text Generation Grammar*, USC/Information Sciences Institute, RR–83–113, April 1983.

8. Mann, W. C., and S. A. Thompson, *Relational Propositions in Discourse*, USC/Information Sciences Institute, RR–83–115, November 1983.

11. SPECIFICATION VALIDATION

Research Staff:
William Swartout
Robert Balzer
Donald Cohen

Support Staff:
Audree Beal

11.1 PROBLEM BEING SOLVED

At ISI, we are investigating ways of alleviating the problems of software development and maintenance through the use of a specification-based approach to software development. In this approach, one uses a high-level specification language to create a formal specification of the desired behavior of a program. This specification specifies *what* a program is to do, but does not necessarily indicate *how* it is to be achieved. High-level constructs in the specification language, not available in conventional programming languages, allow the user to specify the program's behavior without requiring him to state how that behavior is to be achieved. Because the specification is high-level, it should be easier to modify and test than a concrete program. Once the specification denotes the desired behavior, it is implemented by a process of *transformational implementation*, where equivalence-preserving transformations are selected and applied to the specification to replace the high-level constructs with more conventional low-level ones. (See Chapters 3 and 12 for a more complete discussion of this approach.)

Unfortunately, formal program specifications can be difficult to understand, regardless of the specification language used. Yet, because the specification plays such a central role in our approach, it is critical that it be comprehensible. Our experience with Gist, a high-level specification language being developed at ISI [1], has indicated that two major impediments to understandability are the unfamiliar syntactic constructs of specification languages and dynamic interactions between parts of the specification—parts that are often widely separated. These interactions may cause the specification to denote behaviors that were *not* intended by the original specifier, or not to denote behaviors that *were* intended.

11.2 GOALS AND APPROACH

The specification validation project, now completed, has attempted to overcome the impediments of unfamiliar syntax and non-local interactions by constructing computer tools to make specifications more understandable, both to specifiers and to those unfamiliar with formal specification languages.

One tool, the Gist paraphraser, addresses the syntax problem by directly translating a Gist specification into English. We have found the paraphraser to be useful in both clarifying specifications and revealing specification errors. We expected that the English translation would be useful to people unfamiliar with Gist, because it would make Gist specifications accessible, but we were surprised to discover that experienced Gist specifiers found it helpful for locating errors. The reason is that an English translation gives the specifier an alternate view of his specification, one which highlights some aspects of the specification that are easily overlooked in the formal Gist notation.

The paraphraser provides a static description of a specification. A pair of tools, a symbolic evaluator and a trace explainer, address the more difficult problem of making non-local specification interactions apparent by simulating the dynamic behavior implied by the specification. Our approach has been to discover non-local interactions by using the symbolic evaluator to analyze a specification. The symbolic evaluator gathers and integrates constraints from the different pieces of the specification. It discovers what sorts of behaviors the specification allows and what sorts of behaviors are prohibited by constraints. A symbolic evaluator does not require specific inputs. Instead, it develops a description of the range of possible responses to a given range of inputs. Because of this characteristic, it is possible to test a specification symbolically over a range of inputs that would require many test runs if specific inputs were employed.

A specifier interested in the behavior of his specification may direct the evaluator to execute one of the actions defined in the specification. As the evaluator executes the action, some apparently possible execution paths may be eliminated due to constraints, and a more detailed description of the interrelationships within the specification is developed.

The symbolic evaluator produces an execution trace, which details everything discovered about the specification during evaluation. The trace includes not only facts directly implied by the specification, but also any further implications that the evaluator may have derived from those facts using its theorem prover. In addition, the trace records the proof structures justifying the facts it contains. Unfortunately, the trace is much too detailed and low-level to be readily understood by most people. To overcome that difficulty, the trace explainer selects from the trace those aspects believed to be interesting or surprising to the user and uses that information to produce an English summary.

11.3 SCIENTIFIC PROGRESS

Several major observations have emerged from our research:

- Good quality English translations of Gist specifications can be achieved without imposing a burden on the specifier. In designing the paraphraser, we recognized that a specifier would have to adhere to certain style restrictions and might have to provide a few annotations to a specification to indicate how it should be translated, but we wanted the style conventions to be as natural as possible and the annotations to be as few as possible. This was done for two reasons: 1) we felt that the paraphraser would be used much more frequently if a specifier could employ it without making extensive modifications to his specification, and 2) if a specification could be translated making only minimal use of annotations, the translation would be more likely to accurately reflect the specification. We have found that even specifications written before the creation of the paraphraser can often be translated acceptably (though there is usually room for improvement), because the stylistic conventions imposed by the paraphraser are close to those that specifers follow normally.
- The paraphraser has also proved to be a useful tool for debugging specifications. Originally, we thought the paraphraser would be useful mainly for making specifications understandable to those unfamiliar with Gist. However, we discovered that the paraphraser was also very useful in making specification errors more apparent, even to experienced Gist users. Partly, this is because the English paraphrase is more understandable in most situations, but perhaps more important, it gives an alternate view of the specification that makes apparent some aspects of the specification that are not obvious in the formal notation.
- The Gist symbolic evaluator makes some fairly radical departures from the technology that has been developed for symbolic execution of more traditional implementation languages. The main motivation for this was the fact that Gist is based on a predicate calculus view of the world, rather than the implementation view of storage locations

containing values. The major departure is the use of a general inference engine to derive all results, rather than a few special functions that simplify expressions or recognize particular cases of impossible paths. A smaller departure is that conditional executions are described in terms of conditional results rather than a large (or infinite) set of separate paths. The main result of these departures is that the Gist symbolic evaluator can discover (and report) many more results which may be of value to the user for the purpose of understanding and debugging, even if they do not directly affect the execution.

We have found that some of this information is useful in an unexpected way. Often a fact appears which is surprising not because we would have expected it to be false, but because we would have expected a stronger result to be true. Of course, the symbolic evaluator does not say that the stronger result is false, but typically there is a good reason that it cannot be proven, and this points out some interesting (possibly unwanted) feature of the specification.

- Producing English descriptions of symbolic executions is much more difficult than paraphrasing the specification. A number of problems make the simple direct-translation techniques (which worked well for the Gist paraphraser) unsuitable for the trace explainer:

  - **Detail suppression.** The trace is much too detailed to be described in its entirety. The trace explainer uses the structure of the specification and heuristics about what the user is likely to find interesting or surprising in selecting what to describe.
  - **Proof summarization and reformulation.** The symbolic evaluator uses an augmented resolution-based theorem prover in deriving the consequences of the specification. While this approach is arguably attractive for its generality and simplicity, its arcane proof structures could impose a hardship on the user. The trace explainer attempts to reformulate resolution proof structures into more familiar and understandable ones.
  - **Referring expressions.** With the Gist paraphraser, it was usually acceptable to use the name given to an object in the specification as its referring expression in the English paraphrase. The trace explainer cannot rely on this technique alone, since there are objects in the trace that do not appear in the specification. Moreover, depending on context, different referring phrases may be necessary even though the same object is being referred to, and conversely, the same referring phrase may be most appropriate for more than one object.

11.3.1 An Example

This section presents an example specification and a machine-produced description of its symbolic evaluation. This section assumes some familiarity with Gist, although a detailed understanding is not required.

The example presented here is a simplified version of a specification for a postal package router (see [2, 3]). The package router is designed to sort packages into bins corresponding to their destinations. A package arrives at a location called the *source* and its destination is read there. A binary tree of switches and pipes connects the source with the output bins. It is the job of the package router to set the switches so that the package arrives in the proper destination bin (see Figure 11-1). The simplified specification contains just one switch and two bins. In addition, a location called the *input* has been defined as the place where all boxes are originally located. The formal Gist specification appears in Figure 11-2. It is not necessary to understand the formal notations, since an English translation of the specification (produced by the paraphraser) is available: Figure 11-3 is the English paraphrase of the specification's type structure and Figure 11-4 describes the possible actions in this specification.

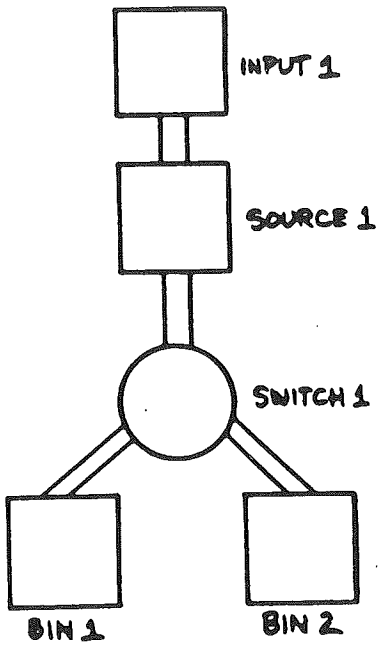


Figure 11-1: Package router

Having defined the type structure and actions, a specifier may wish to define some test sequences of actions to see how the constraints of the specification interact to limit the behavior of the specification in ways that are not obvious from the static specification alone. In Figure 11-5, the user has defined such a test sequence. The user has also given preconditions to define the initial state and the structure of the switching network, and a postcondition to describe the final goal of the system. Notice that in the action body, all operands are specified nondeterministically. For example, the first action invocation states that a box is to be inserted, but it does not say which box. The intent of such a statement is that any box may be inserted, as long as no constraints are violated. This nondeterministic reference is one of the freedoms Gist allows, giving the specifier greater expressive power and preventing him from having to over-specify behaviors. Because the user does not have to explicitly select parameters, he can see with just one test action whether it is ever possible to achieve the postconditions using the particular sequence of action invocations given.

After the symbolic evaluator runs, the specifier can use the trace explainer to see an overview of the results of symbolic execution (see Figure 11-6). The trace explainer highlights the behaviors that appear to be surprising and explains why they occur by translating the symbolic evaluator's mechanically produced proofs into English.

```
begin
 type box(Location | location, Destination |
                        bin);
 type location()unique supertype of
   <input() definition{input1};
    source(Source-outlet | switch)
          definition{Source1};
    internal-location()unique supertype of
      <switch
       (Selected-outlet
            |internal-location,
        Outlet|internal-location
              :multiple)
           definition{switch1};
       bin() definition{bin1, bin2}>>;
 agent PackageRouter() where
    action Insert[box]
     definition update :Location of box
            from input1 to Source1;
    action Set[switch]
     precondition ~$:Location = switch
     definition update :Selected-outlet
        of switch to switch :Outlet;
    action Move[box]
     precondition box:Location = Source1 or
            box :Location = a switch
     definition
       if box :Location = Source1
         then update :Location of box
           to Source1:Source-outlet
         else update :Location of box
           to box :Location
              :Selected-outlet;
    action Test[]
     precondition switch1 :Outlet = bin1
     precondition switch1 :Outlet = bin2
     precondition Source1 :Source-outlet =
                  switch1
     precondition for all box ||
        box :Location = input1
     postcondition for all box ||
        box :Location = box :Destination
     definition begin
              Insert[a box];
              Move[a box];
              Insert[a box];
              Move[a box];
              Set[a switch];
              Move[a box];
              Move[a box]
            end
  end
end
```

Figure 11-2: Formal Gist specification for package router

There are boxes, locations and package-routers.

Each box has one location. Each box has one destination which is a bin.

Internal-locations, sources and inputs are locations.
Bins and switches are internal-locations.
Bin1 and bin2 are the only bins.
Switch1 is the only switch. The switch has one selected-outlet which is an internal-location. The switch has multiple outlets which are internal-locations.
Source1 is the only source. The source has one source-outlet which is a switch.
Input1 is the only input.

Figure 11-3: Paraphrase of package router type structure

A package-router can insert a box, set a switch, or move a box.
To insert a box:
Action: The box's location is updated from input1 to source1.
To set a switch:
Action: The switch's selected-outlet is updated to an outlet of the switch.
Preconditions:
The switch must not be the location of any box.
To move a box:
Action:
If: The box's location is source1,
Then: The box's location is updated to the source-outlet of source1.
Else: The box's location is updated to the selected-outlet of the switch that is the box's location.
Preconditions:
Either:
1. The box's location must be source1, or
2. The box's location must be a switch.

Figure 11-4: English paraphrase of possible actions

To test:
Action:
1. Insert a box.
2. Move a box.
3. Insert a box.
4. Move a box.
5. Set a switch.
6. Move a box.
7. Move a box.

Preconditions:
For all boxes:
The box's location must be input1.
The source-outlet of source1 must be switch1.
An outlet of switch1 must be bin2.
An outlet of switch1 must be bin1.
Postconditions:
For all boxes:
The box's location must be the box's destination.

Figure 11-5: English paraphrase of a test action

1. A box, call it box1, is inserted.
Result: The new location of box1 is source1.
The explainer describes the action invocation as it was stated in the test case. It makes up the name "box1" for this box so that it can be conveniently referred to later. The explainer then describes the result of this action invocation.

2. A box is moved. The box must be box1 since
2.1 For all boxes except box1, the box's location is input1, and
2.2 The precondition of moving a box requires that either:
2.2.1 The box's location must be source1, or
2.2.2 The box's location must be a switch.
Result: The new location of box1 is switch1.
Something surprising has happened. In the test case, the action invocation was made with a non-deterministic parameter, but the constraints of the specification force the selection of one particular box, namely box1. The explainer recognizes this sort of behavior as surprising and describes not only the restriction on binding the parameter, but also the reasons behind it.

3. A box, call it box2, is inserted. The box must not be box1 since
3.1 The location of box1 is switch1, and
3.2 The location of the box to be inserted must be input1 since the update in inserting a box requires it.
Result: The new location of box2 is source1.

4. A box is moved. The box must be box1 since otherwise, at the start of step 5, the location of box2 would be switch1 but the precondition of setting a switch requires that the switch must not be the location of any box.
Result: The new location of box1 is the selected-outlet of switch1. Switch1 is not the location of any box.
At the start of step 4, box1 is at the switch, and box2 is at source1. It would appear that either one could be moved in step 4 since both satisfy the preconditions of the move. However, if box2 moved, it would be impossible to execute the next step. So, as the explainer describes, the non-local interaction with step 5 constrains the parameter binding.

5. A switch is set. The switch must be switch1 since there are no other switches.
Result: The new selected-outlet of switch1 is an outlet, call it outlet1, of the switch.

6. A box is moved. The box must be box2 since the precondition of moving a box requires that either:
6.1 The box's location must be source1, or
6.2 The box's location must be a switch.
Result: The new location of box2 is switch1.
The proof that the box to be moved must be box2 is actually quite involved. The system currently has no good way of summarizing proofs of this type, so it falls back on another heuristic. The explainer examines the proof structure to find the statement in the specification that was used specifically to constrain this choice and displays it. That is, rather than showing a proof, we just display the parts of the specification that became relevant in constraining this behavior. This heuristic seems to work well, and it provides the explainer with an "escape" so that it can convey some information even if it can't reformulate the proof.

7. A box is moved. The box must be box2.
Result: The new location of box2 is outlet1. For all boxes, the box's location is the box's destination.
Since the justification for this step is the same as for the preceding, the explainer omits it.

Figure 11-6: Machine-produced description of symbolic evaluation of test

11.4 IMPACT

It must be emphasized that the tools we have developed are laboratory prototypes and we have not attempted to apply them to large-scale specifications. Even so, we feel the research described here establishes the promise of this approach and, if further pursued, could benefit software production substantially. First, these tools will make specifications more accessible and understandable both to specifiers and to customers. As we have argued, formal specifications are often difficult to understand, yet understandability is crucial if the specification-based approach to program development is to succeed. Second, tools such as the Gist paraphraser can perform a valuable debugging function by presenting a specification from a different viewpoint. Third, tools such as the symbolic evaluator and trace explainer will enable a specifier to test a specification directly, so that errors can be caught at a high level, before substantial effort is invested in an implementation.

References

1. Balzer, R., Goldman, N. & Wile, D., "Operational specification as the basis for rapid prototyping," in *Proceedings of the Second Software Engineering Symposium: Workshop on Rapid Prototyping*, ACM SIGSOFT, April 1982.

2. Hommel, G. (ed.), *Vergleich verschiedener Spezifikationsverfahren am Beispiel einer Paketverteilanlage*, Kernforschungszentrum Karlsruhe GmbH, Technical Report, August 1980. PDV-Report, KfK-PDV 186, Part 1.

3. Swartout, W. and Balzer, R., "On the Inevitable Intertwining of Specification and Implementation," *Communications of the ACM* 25, (7), July 1982, 438-440.

12. SUPERVISORY CONTROL OF TRANSFORMATIONAL IMPLEMENTATION SYSTEMS

Research Staff:
Robert M. Balzer
David S. Wile
Martin S. Feather

Research Assistants:
Wellington Chiu
Steve Fickas

Support Staff:
Audree Beal

12.1 PROBLEM BEING SOLVED

Software specification, development, and maintenance continue to present an enormous problem to everyone involved with computers. We believe that the computer itself must play a far more significant role in the software development process than it does presently. The software designer's role should be streamlined to require only decision-making and guidance, while the computer's is expanded to include manipulation, analysis, and documentation. The key to such support is to capture in the machine all crucial information about the processes of specification, design, implementation, and maintenance.

For several years we have been developing an alternative software development paradigm based on this tenet [1].[1] We envision a future user developing a high-level specification of what he or she wants a program to do, and transforming the specification into an efficient program, using a catalog of (proven) correctness-preserving transformations (see Figure 12-1). Most debugging and all maintenance will be performed on the specification, which will have an operational interpretation suitable for testing. Reimplementation will be necessary to tune implementations exhibiting unacceptable performance characteristics.

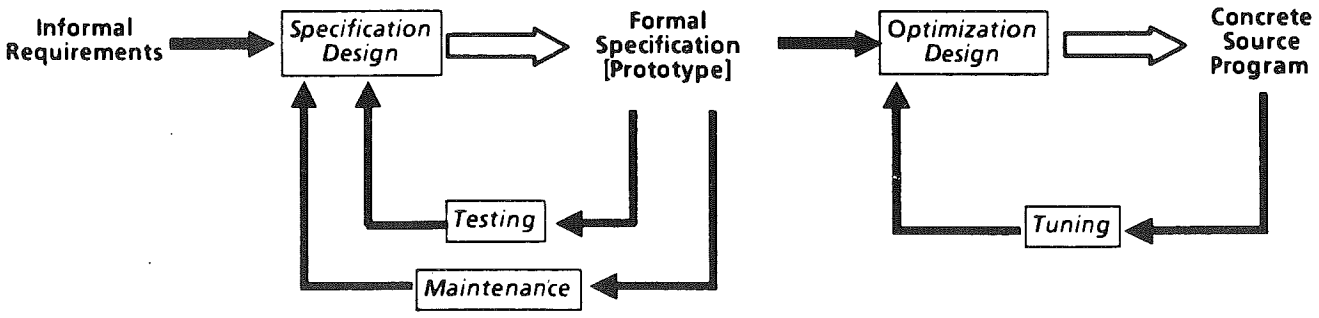


Figure 12-1: Alternative software development paradigm

[1] Several researchers led by Robert M. Balzer, working on projects funded by DARPA, NSF, and RADC, have provided the foundations for this paradigm.

This automation-based paradigm will rely on an integrated set of tools that directly support human developers in the processes of requirements specification, system design, implementation, and maintenance. It will be characterized by the fact that the entire evolution of a system—the history of all four of these processes, as directed by the developers—will occur and be recorded within the integrated environment. This history will provide the "corporate memory" of each system's evolution. The software development system will be an active participant in the development process, using the history to determine how these four processes interact: what assumptions they make about each other, what the rationale behind each evolution step was, how the developed system satisfies its requirements, and how to explain all of these to its developers.

We have developed a specification language called Gist, which formalizes some of the expressive constructs commonly used in natural language specifications, including nondeterminism ("a message from another site"), descriptive reference ("the longest message"), historical reference ("the last message the user looked at"), constraints ("never send multiple copies of a message to the same person"), and demons ("if a week passes without a reply to a request, inform the sender"). A Gist specification describes the behavior of both the program and its environment; this provides a natural way to specify embedded programs that interact in complicated ways with their environment. Gist's expressive power makes it possible to specify ***what*** behavior a program should exhibit without saying ***how*** it should be implemented [2].

Given that we already have a language for expressing specifications, three major problematical areas require research and development before our transformation-based paradigm can succeed:

1. Tools, methods, and support facilities for ***acquiring***, ***designing***, and ***developing*** the specification consistent with a user's intent;
2. Tools, methods, and support facilities for ***implementing*** and ***optimizing*** the specification;
3. A framework for ***understanding***, ***reusing***, and ***maintaining*** previous specifications and optimizations.

The primary goal of the ***Supervisory Control of Transformational Implementation Systems*** project was to design and develop a framework for understanding, reusing, and maintaining previous specifications and optimizations.

12.2 GOALS AND APPROACH

12.2.1 Overview

Over the years, the group has developed several tools to support the specification ***process***, i.e., to help bridge the gap between the informal intent inside the specifier's head and its formal expression in Gist. Specifications in any formal language tend to be difficult to understand, because they are usually concise and lack the redundancy found in natural language descriptions of the same behavior. Hence, we have designed a program that exposes static aspects of Gist specifications by paraphrasing them into English [8]. To expose dynamic implications of a specification, we have developed a symbolic evaluator that looks for interesting consequences of a Gist specification [4], and a program that explains the symbolic trace in English [9]. By clarifying what a specification really means, such tools help ***validate*** the specification, i.e., increase the specifiers' confidence that the specification matches their informal intent and the users' confidence that the resulting system will meet their needs. (For more information on the Gist paraphraser and the symbolic evaluator, see Chapter 11.)

However, we have discovered that when people specify programs to other people, they use an ***incremental*** technique for conveying the information necessary to develop the program. This technique is built on a very stylized set of methods, such as ***refinement*** of modeling detail, ***generalization*** of functionality, ***introduction of exception*** into idealized specification, and ***restriction*** of scope or functionality. In the future, our Mappings project (see Chapter 3) will address the problem of capturing these methods as ***formal*** methods, thus permitting the specification development to appear much more natural to future readers of the specification. We have called these methods ***high-level editing*** commands, to emphasize the fact that they do indeed change the specification, yet at the same time convey more information to a reader than conventional keystroke editing commands. The use of high-level editing commands to design specifications is analogous to the use of transformations to develop implementations (see Figure 12-2).

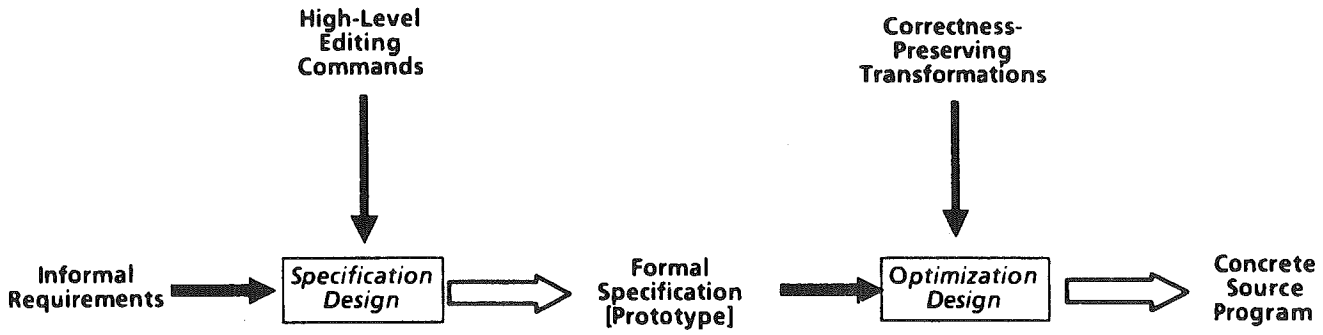


Figure 12-2: Design process mappings

Since its inception, the Mappings project has addressed the problem of representing programming knowledge in terms of our specification language, Gist, by discovering correctness-preserving transformations for translating Gist's high-level constructs into the lower level constructs used in more conventional programming languages. Hence, the Mappings project addresses the second problem mentioned above: encapsulating programming knowledge in terms of concise mappings from Gist into alternative ***implementations***. In the future, more emphasis will be given to mappings for ***optimization*** of these implementations.

The Supervisory Control of Transformational Implementation Systems project developed the system support needed to facilitate the automated implementation of specifications using transformations. It addressed the third problem mentioned above: the framework for describing the design activities of specification and implementation. This paradigm is based on capturing the ***processes*** of specification, design, implementation, and maintenance within the computer and supporting them via automated tools. We have developed an experimental system in which the user develops not only the implementation, but also a formal explanation of how it arose; the design decisions, assumptions, and optimization steps, and their structural relationships, are recorded in a formal design document suitable for automatic reuse by the transformation system (see Figure 12-3). Hence, using this ***formal development*** the system can replay optimizations on a changed specification to produce a new implementation automatically [10]. Naturally, all future mappings and transformations will be designed to fit into this framework.

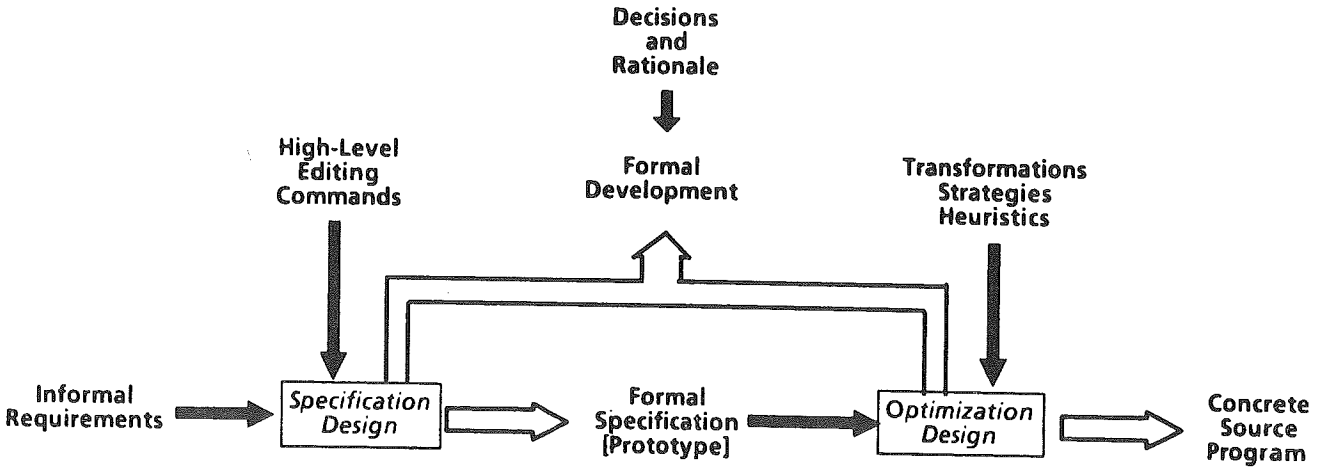


Figure 12-3: Formal developments

Our primary goal is to obtain a usable development system. We believe that the leverage provided by our alternative software development paradigm (especially in the beginning) will arise from its support of the maintenance process rather than from its support of the initial implementation. This is because the increased formalization of the development process requires more involvement by the human developer (at least until the level of automation is significantly increased) for the initial implementation. On the other hand, maintenance involves small changes. Increased formalization of these changes is insignificant if the already formalized development can be adapted and reused.

The SCTI project has made considerable progress toward this goal, including the creation of a language–independent development system called Popart, with facilities for grammar–based editing, analysis, and transformation; a formal representation of program development—explanations of how the implementation was produced; a compilable subset of our (wide–spectrum) formal specification language to act as a target language for transformation; a goal–directed transformation selection and application mechanism; and a system for recording and supporting the evolution of real systems (to improve our understanding of this evolutionary process). However, much remains to be done. Current limitations include inadequate automation of transformation selection and application, inadequate analysis tools (verifying the precondition of transformations), an incomplete transformation catalog, no performance prediction tools, no direct support of maintenance, and inadequate integration of current capabilities into a coherent system.

Our primary research goal for the future is to support ***reimplementations***, in a follow–on project called ***Transformation–Based Maintenance***. This support will take three forms: strengthening the formal representation of developments (especially the rationale behind each step), supporting the evolution of specifications through the same mechanisms that support maintenance (and in fact integrating the two processes), and developing a formal language of modification for both specifications and developments.

12.3 SCIENTIFIC PROGRESS

The SCTI project has focused on the formalization, mechanization, and automated support of the development process through transformations. Our primary tool is Popart,[2] a prototype program development system which applies transformations selected by the user. The detailed strategy used for such transformation applications, called a development, is formalized in a modifiable and executable language called Paddle. We have also created a set of analysis tools, which determine whether the enabling conditions of selected rules are satisfied. (If not, then they should become subgoals to be satisfied so that the development can proceed.) A goal-directed problem solver has been built to automate the achievement of these and other goals posed by the user (this problem solver has not yet been integrated into Popart). Another facility, structure comparison between the program and the transformation pattern, detects differences between the left-hand side of a selected transformation and the current state of the program. These differences are eliminated by the creation of subgoals. (This facility has not yet been integrated into Popart.) All of this transformation activity leads to the conversion of specifications written in Gist into programs written in a compilable subset called Will. Finally, we have built a separate system (not based on transformations), called the Develop System, to record the actual evolution of real systems in order to sharpen our perceptions about how that activity should be supported by our tools.

12.3.1 Formal Representation of Developments

We have created Paddle, our language for documenting the development structure of a program from a specification. This language represents a development as a hierarchical plan that incorporates subplans, strategies, refinements, and transformations. This development defines, in an executable form, the sequence of transformations and manipulations that must be performed to convert the specification into a program. In this mode, each refinement defines how to accomplish the refined goal by modifying the specification/program resulting from previous steps. The execution of such a development can be faulty, because some development step either has not yet been defined or is inappropriate for the current state of the specification/program being developed. When this occurs, the development can be edited using the Popart system (described below) and its execution can be either continued or restarted from some previous point.

The development is thus a "program of transformations" for implementing a specification which can be debugged via standard interactive debugging techniques. It is a structured, documented, and (when it successfully runs without manual intervention) fully automatic method for implementing the specification.

This formal manipulable representation of the development is our primary scientific achievement and is more fully documented in [10].

12.3.2 Prototype Program Development System

A grammar-based pattern matcher has been enhanced with several other grammar-based tools to become a prototype Program Development System, Popart. Popart includes grammar-based tools for editing and prettyprinting a specification/program, for transforming it via grammar-based pattern matching and replacement, and for automatically simplifying the transformed specification/program

---

[2] The underlined items identify our accomplishments and are described in detail below.

and "conditioning" it (changing its syntactic form) to facilitate subsequent transformations. The automatic simplification and "conditioning" mechanisms are currently quite primitive, while the other capabilities are more mature, complete, and usable. In addition, this Prototype Development System includes the development ideas described in Section 12.3.1 and is also described in more detail in [10].

12.3.3 Compilable Subset of Gist (Will)

We have identified and implemented a compilable subset of Gist. This subset, Will, shares much syntactic structure with Gist, reducing the need for "form" changing transformations, but it replaces many of Gist's semantic freedoms with similar, but lower level, constructs which can be more efficiently compiled. Generally, Will supports the data typing, structuring, and associative retrieval of Gist and its event-driven (demon-based) processing. It does not support the full semantics of historical references, constraints, or inferences. Instead, it supports more restricted versions, such as constraints that cause an exception to be raised when violated (in Gist, constraints affect the non-deterministic control structure, so that execution proceeds only along paths that do not, and will not, violate them), and monotonic inferences (once true, they remain true even if their support subsequently vanishes).

Will's significance lies less in its novel features than in its expected role in our efforts to develop a usable transformational implementation system. Now we have a (high-level) implementation language to use as a target for our developments that will be syntactically compatible with our specification language. This compatibility will permit us to focus our transformation-building efforts on the few (but important) semantic distinctions between the two languages.

Furthermore, using user-supplied annotations of how to compile specific program pieces (attached during the development steps taken in Popart) the compiler can produce the full variety of Interlisp and Zetalisp data structures. Annotations accomplish two important objectives. First, they act like delayed optimizations (the decision has been made but not yet carried out), allowing further transformations to be either sensitive or insensitive to the decided implementation (by examining or ignoring the annotation) without having to contend with the syntactic variability that carrying them out would entail. In a strong sense, the annotations form an abstract formal design language that acts as an explicit bridge between the decision space and the artifact being designed. We expect to continue investigation of the power of the annotation mechanism.

Second, the annotation mechanism permits optimizations that are competitive with hand construction. We expect to use Will as our implementation language for all future work and to use the unannotated form as a "rapid prototype." Thus, even without using our development system to map from specification to implementation, we will be able to use a much higher level implementation language to express prototypes that can later be optimized via annotations.

12.3.4 Goal-directed Development

In his doctoral dissertation [6], Steve Fickas has created an automated development system that selects and applies transformations to achieve developer-stated goals. He has created a language (Glitter) for stating goals, a catalog of methods for achieving those goals (these methods consist of primitive transformations and of higher level planning knowledge), a catalog of selection rules for choosing among competing methods, and a standard artificial intelligence back-chaining problem solver. All of this has been implemented in Hearsay-III [5], a framework for writing expert systems.

Experience with the system to date, though quite limited, indicates that the level of automation has been improved by about an order of magnitude (i.e., a developer has to make only 10 percent as many decisions as before). The contribution of his thesis lies in its demonstration of the feasibility of this level of automation, in its development of a goal language through which the developer directs the system, and in the codification of development knowledge and selection rules.

As a thesis project, the system was built separately from Popart. It has a quite limited knowledge base (methods, goals, and transformations) and is very slow because it uses Hearsay–III for both its problem state and program state representations. Nevertheless, we believe it is an important contribution and intend to integrate its capabilities into our Popart system. It is described in more detail in [6].

12.3.5 Analysis Tools

Flow analysis information—both data flow and program flow—is useful to the implementor when choosing optimizations and determining the truth value of enabling conditions for transformations. Tools to compute such information have been implemented for Gist. These tools dynamically recompute information on demand after specification changes have been made. Of particular significance is the language–independent, table–driven mechanism which was designed for expressing these computations and incorporated into Popart.

The particular analysis "tables" implemented are relation usage, relation retrieval, relation insertion, object creation, object deletion, action and demon invocation, and variable scoping. Some problems with the definitions of these tables have led to the belief that a more canonical form than the parse trees produced by Popart should be used. Hence, we are now attempting to base some of the transformations on a more "abstract syntax."

12.3.6 Develop System

We have created a system (Develop) that automatically captures the development and maintenance of actual programs written in Interlisp, in terms of the objects (data structures, functions, macros, etc.) created and modified (including the modifications themselves). This trace of development behavior is annotated by statements of intent by the user, indicating the type of development step (augmentation, generalization, revision, exception, specialization, establish–convention, reorganization, bug–fix, tuning, etc.) being initiated and a descriptive comment.

During the modifications, the user can recursively initiate substeps that modularize his efforts to achieve the intent of the higher level step (goal). Also, the user can post "pending steps" on an electronic blackboard for future implementation (this useful notepad allows us to record future plans as they arise). It is important to recognize that this represents a different paradigm for construction of the formal development than that supported by Popart and Paddle.

Limited use of this system has shown that the development history can become the primary source of internal documentation (for the maintainer) and that the existence of this development history encourages more orderly development and maintenance.

Our primary reason for creating this system was to gather histories of the actual evolution of real systems in order to provide insights into how users conceptually structure developments and into the

dynamics of system growth and modification. These insights will be fed back into the redesign of our formal representation of developments and the support mechanisms of Popart. In fact, they have already become the basis for the "high-level editing" language outlined in Section 12.2.1. However, we unexpectedly found that these development histories could also provide support for novel maintenance tools within the Develop System.

12.3.7 Structure Comparison

Another graduate student, Wellington Chiu, has completed his thesis on comparing two versions of a specification/program. This comparison can be used in two ways. First, it can be used to help people understand what has changed between steps in a development, either visually, via highlighted portions of the prettyprinted specification/program, or via a textual description of the changes at an interesting (semantic) level of granularity. Second, it can provide a description of the differences between the current state and a goal state (eventually intended as input to the goal-directed development system developed by Fickas).

The scientific contribution of this structure comparison system lies in its rules (some of which are heuristic) for forming explanations, and in its formalization and extension of the primitive modification operations upon which such explanations are built. Some of these primitive modification operations include semantic information about the specification/programming language. This work is more fully described in [3].

12.4 IMPACT

The Supervisory Control of Transformational Implementation Systems project is supportive of the transformational implementation software development paradigm [1]. Initially, this paradigm will dramatically improve programmers' ability to maintain software systems; ultimately, the entire lifecycle from design through specification, implementation, debugging, and maintenance will be streamlined. This will allow programmers to more rapidly produce and maintain software systems, and will make feasible the construction of larger, more complex systems with enhanced functionality and flexibility. In addition, this project will codify the strategic, language-independent knowledge used by programmers in specification and optimization. This knowledge will also be useful for programmer education, independent of its mechanized application.

12.5 FUTURE WORK

12.5.1 Attempted Realistic Use of the Popart System

Our grammar-based program development system, Popart, has been subjected to considerable experimentation. Transformations of several example specifications were attempted with varying degrees of success. Each of these examples is either a real system implemented by someone else or a part of our software system. Among the examples attempted were a package routing specification, two different portions of the parser used in the Popart system, and a VLSI implementation of a hidden line elimination and shading algorithm for a graphics display. Transforming these examples demonstrated that real programs could be developed successfully with Popart. Yet simultaneously, these examples were unsuccessful from the standpoint of practical use of Popart (in the sense that improvements in the overall effort, reliability, and understandability were not realized), for the following reasons:

- Popart itself was not adequately user-engineered for efficient program manipulation.
- A large body of transformations for the specification language, Gist, had not been built up.
- The language for expressing the optimization plan and transformations (Paddle) was found to be inadequate for the following reasons:
  - In Paddle, goals must be achieved in the same order that they appear in the goal structure. Frequently this restricts the user's ability to introduce goals as they naturally occur during the design process because their complete achievement will not be realized until much later.
  - The language for describing transformations is too primitive. This makes expression of frequently occurring idioms clumsy.
  - Generally, the commands in the editor are too low level to describe the particular optimizations needed. This is strikingly obvious with commands intended to focus the editor.
- The reliance of "original specifications" on their predicted implementations was discovered to be significant. Hence, how to feed information from the implementation back into the specification design is problematical.

The first two problems have been partially addressed. Some effort was spent improving the facilities for editor interaction. A rather thorough Popart manual was produced. Based on the Develop System, described in Section 12.3.6, some facilities for more efficient recording of a development were added. A small, useful set of transformations and editing commands specific to Gist was created.

Considerably more difficult are the problems with the language for expressing program developments, Paddle. Paddle is currently being redesigned to incorporate a more goal-directed model of system/user interaction, along the lines proposed by Fickas (discussed in Section 12.3.4 above). In addition, more powerful primitives for manipulating programs are being incorporated, along with a variety of options for maintaining the editor's focus of attention.

It is interesting that the final problem mentioned above—feedback from optimization to specification—has led to the discovery that the design of the original specification involves a process quite similar to the design of an optimization. In fact, Paddle is adequate for expressing the ***structure*** of both types of design, but not for expressing the ***content*** of such design structures.[3] For both optimization and specification design, a higher level editing language is necessary. These two tasks, integrating the design of the specification with the design of its implementation and creating higher level editing operations for such design, are part of our proposed new work.

12.5.2 Research Effort

We now understand that the increased formalization of the development process, which necessarily accompanies its capture and support within the computer, requires increased involvement and effort by the human developer. This additional "formalization" cost is presently very expensive for initial implementations of software systems. However, initial implementation is not the major expense in creating software systems. Instead, maintenance and extension of functionality consume the majority of all development effort. Fortunately, in our alternative software paradigm, supported by a program development system, the leverage lies in maintenance rather than in initial implementation.

---

[3] This similarity has been taken into account in Figure 12-3.

In the current software paradigm the information needed for maintenance is not available (because the development process is undocumented and the maintainer has only the source listings to work from), while in our alternative paradigm it is available in the form of a formalized development. The cost of formalizing the original development can provide leverage if it can be used as the basis for successive reimplementations during maintenance. Hence, developing the conceptual basis and technology to support automated reimplementation (via replaying the previous development) is the primary scientific focus of our follow-on project, Transformation-Based Maintenance.

In addition to this primary focus, we have two other major scientific goals and two engineering goals of interest. The two scientific goals are integrating the process of implementation development with that of specification evolution (and supporting them with common mechanisms), and developing a formal language of modification for both specifications and developments. Our two engineering goals are continued development of Popart and the Develop System in the framework of the Information Management system under development at ISI [7].

REFERENCES

1. Balzer, R., C. Green, and T. Cheatham, "Software technology in the 1990's: Using a new paradigm," *IEEE Computer*, November 1983.

2. Balzer, R., N. Goldman, and D. S. Wile, "Operational specification as the basis for rapid prototyping," in *Proceedings of the Second Software Engineering Symposium: Workshop on Rapid Prototyping*, ACM SIGSOFT, April 1982.

3. Chiu, W., *Structure Comparison and Semantic Interpretation of Differences*, Ph.D. thesis, University of Southern California, 1981.

4. Cohen, D., "Symbolic execution of the Gist specification language," in *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pp. 17-20, Karlsruhe, Germany, 1983.

5. Erman, L., P. London, and S. Fickas, "The design and an example use of Hearsay-III," in *Proceedings of the Seventh International Joint Conference on Artifical Intelligence*, pp. 409-415, Vancouver, B.C., Canada, 1981.

6. Fickas, S., *Automating the Transformational Development of Software*, Ph.D. thesis, University of California at Irvine, 1982. Issued as USC/Information Sciences Institute, RR-83-108 and RR-83-109, March 1983.

7. Neches, R., R. Balzer, D. Dyer, N. Goldman, and M. Morgenstern, "Information management: A specification-oriented, rule-based approach to friendly computing environments," in *Proceedings of the IEEE Conference on Systems, Man, and Cybernetics*, IEEE, January 1984.

8. Swartout, W., "GIST English generator," in *Proceedings of the National Conference on Artificial Intelligence*, pp. 404-409, American Association for Artificial Intelligence, Pittsburgh, 1982.

9. Swartout, W., "The GIST behavior explainer," in *Proceedings of the National Conference on Artificial Intelligence*, pp. 402-407, American Association for Artificial Intelligence, Washington, D.C., 1983.

10. Wile, D. S., "Program developments: Formal explanations of implementations," *Communications of the ACM* 26, (11), November 1983. Also available as USC/Information Sciences Institute, RR-82-99, August 1982.

13. INTERLISP

Research Staff:
Gary L. McGreal
Ray Bates
Mark Feber
Don Voreck
Dave Dyer

13.1 PROBLEM BEING SOLVED

This project is creating a self-sustaining support center for Interlisp. The goal of the project is to provide large-address-space portable versions of Interlisp for the VAX and for other hardware architectures.

13.2 BACKGROUND

A serious problem has arisen within the Artificial Intelligence and Computer Science research communities as the research projects using LISP have developed larger and more complex programs. A basic limitation of the PDP-10, on which current systems are implemented, has been reached. The address space available (256K words) is too small; current research projects routinely require several times as much space. A recent DARPA-sponsored workshop on symbolic computation estimated that minimum requirements are 8 million addressable words and that these requirements are expected to grow towards 2↑30 words; conventional wisdom estimates this growth at .5*bits per year.

Experience has consistently shown that programming costs rise exponentially as resource limitations are approached. Thus, many LISP-based research projects are expending considerable resources in system programming efforts to mitigate the effects of this address-space limitation.

This difficulty provided the incentive for the community to consider various alternatives. The first question was whether LISP was the best language for such large, complex research projects. The result of this discussion was the conclusion that there is no existing widely acceptable alternative (although this is an important research area). The second question was which dialect of LISP is best. There are two main dialects, Interlisp and Maclisp, which differ mainly in their control structures and the environment developed around the language. The user communities expressed strong preferences for their own dialects. The Maclisp community is pursuing a number of alternatives, including an attempt at a consolidation of dialects, the Common Lisp project, and two new dialects: CONS, implemented on new hardware (a specially designed LISP machine), and NIL, being implemented on an S1.

ISI's Interlisp project addresses the needs of the Interlisp community. For this community, the LISP machine option was rejected because of concern for the availability and maintainability of the hardware and because of dialect incompatibilities. Instead, the VAX computer was chosen as the initial machine for Interlisp implementation on the basis of its widespread use throughout the academic community and the opportunity for technology transfer such widespread use affords.

13.3 GOALS AND APPROACH

The ISI–Interlisp project was begun in mid–1979 to provide a newer, more powerful alternative to Interlisp–10 as a LISP environment suitable for research. The result is an efficient, portable, fully functional system compatible with other Interlisps and supporting a large virtual address space. ISI–Interlisp runs under two operating systems, VMS and UNIX.[1] The implementation of ISI–Interlisp on the VAX, one of the most popular machines in research facilities and on college campuses today, assures it of a long, productive future. The initial sponsor of ISI–Interlisp was DARPA, under contract number MDA 903–81–C–0335.

The implementation of ISI–Interlisp relied heavily on two sources: Interlisp–D, which runs on personal computers developed at the Xerox Palo Alto Research Center, and Multilisp, implemented at the University of British Columbia for the IBM–370. ISI–Interlisp is a more traditional implementation than some of the newer LISPs. It is written in Interlisp itself and in C, the implementation language for the UNIX operating system, and took approximately six person–years of effort to complete.

13.4 PROGRESS

The major accomplishment this year was the completion of a native VMS version of Interlisp based on VAX–11C. The first VMS version of Interlisp required the use of a UNIX emulator for VMS, called Eunice (developed at SRI International by David Kashtan). Using Eunice caused several problems. The major problem was that a VMS user would have to become familiar with UNIX file names to use Interlisp. Furthermore, a site would have to install Eunice in order to run Interlisp. Since we replaced Eunice with VAX–11C, ISI–Interlisp is easier to install and use, faster, and more reliable.

Other major tasks included the development of a new code generator having a peephole optimizer. More functions were coded inline and some bottlenecks were removed. The new code generator makes the system smaller and faster.

Many new Interlisp packages were produced or imported. One major package was for UNIX, enabling ISI–Interlisp to use UNIX pipes.

In addition to the development effort, there are several major ongoing Interlisp tasks. Interlisp is constantly growing and changing, and ISI–Interlisp must also change to keep up with improvements. The task of maintenance is another ongoing effort. User feedback and the maintenance effort allow problems and bugs to be discovered and eliminated.

13.5 IMPACT

The premise of the Interlisp project is that no one machine will (or should) dominate Interlisp usage in the future as the PDP–10 has until now. Instead, new, more cost–effective machines will be appearing with increasing frequency. Therefore the new, large–address–space Interlisp should be constructed to be portable, and the Interlisp support center should migrate this system to new hardware on a periodic basis. Interlisp has been a major dialect of LISP for the AI research community. The introduction of ISI–Interlisp will make it possible for a much wider set of users from the academic community, government, and private industry to become involved with AI tools.

---

[1] specifically Berkeley 4.1 VM/UNIX

13.6 FUTURE WORK

In addition to the ongoing maintenance tasks, several development efforts are planned for the next year:

- Micro-code some functions of ISI-Interlisp on a VAX 780 under VMS. This will improve the speed of ISI-Interlisp substantially by removing some bottlenecks.
- Produce a version of ISI-Interlisp that will run under Berkeley UNIX 4.2. UNIX 4.2 is a major release of UNIX, and Interlisp requires modifications in order to run on that system.
- Make ISI-Interlisp more usable under VMS. The first step is to allow the user to call any system service under VMS. The second step is to enable ISI-Interlisp to use VMS mailboxes.
- Make ISI-Interlisp respond more effectively when a critical resource of the operating system, such as sufficient memory or a floating point accelerator, is lacking.
- Maximize the sharing of memory among users on VMS and UNIX 4.3. The current system does not share pages among ISI-Interlisp users. By having ISI-Interlisp share pages, the operating system will be able to support more ISI-Interlisp users.
- Produce Interlisp for other machines. The ISI-Interlisp implementation of Interlisp is portable. The main requirement for a foreign machine is that it support the C language and have virtual memory support. The main tasks would be coding a new code generator and designing important data structures to match the foreign machine architecture. As the use of Interlisp and AI spreads, more and more manufacturers will want Interlisp. ISI will be ready to supply them with a version of Interlisp.
- Code a different garbage collector. Two different garbage collectors are being considered. ISI-Interlisp currently uses a stop and copy collector, which is acceptable for UNIX but is wasteful for VMS. A better collector would be a mark and sweep garbage collector. A reference count garbage collector would probably be the best solution for both VMS and UNIX, but this may be more difficult to implement.

14. COMPUTER RESEARCH SUPPORT

Director: Gary L. McGreal
Technical Assistant: Dale Russell
Manager NOSC Facility: Serge Polevitsky
Liaison Gunter Facility: Dave Hale

Technical Staff:

Systems Software:
Joel Goldberger
David Bilkis
Mike Butler
William Moore
Craig Rogers
Craig Ward
John Weber
Tom Wisniewski

Hardware:
Daniel Pederson
Glen Gauthier
Ramon Gonzales
Norman Jalbert
Raymond Mason
Jeff Rolinc
Daniel Trentham
Leo Yamanaka

Operations:
Walt Edmison
Steve Carroll
Sam Delatorre
Brent Gesch
Roylene Jones
Joseph Kemp
Linda Sato
Toby Stanley
Mike Zonfrillo

Network Services:
Vickie Gordon
Anna-Lena Neches
Wayne Tanner
Christine Tenney

Support Staff:
Manon Levenberg
Kay Freeman
Julie Kcomt
Robert Smith

14.1 PROBLEMS BEING SOLVED

The Computer Research Support project is responsible for providing reliable computing facilities on a 24-hour, 7-day schedule to the ARPANET research and development community. At the same time, the project makes available to ARPANET users the latest upgrades and revisions of hardware and software. The project provides continuous computer center supervision and operation, and a full-time customer-service staff that is responsive to user inquiries. This project supports three computer installations, the largest at ISI's main facility in Marina del Rey. The other supported facilities are at the Naval Ocean Systems Center (NOSC) in San Diego and at Gunter Air Force Base.

14.2 GOALS AND APPROACHES

The Computer Research Support project provides support in four interrelated, though distinct, areas: Hardware, System Software, Operations, and Network Services. The goals and approaches of each are summarized below.

14.2.1 Hardware

To achieve a reliability goal of 98.7 percent scheduled uptime, the preventive and remedial maintenance responsibilities have been assigned to an in-house computer maintenance group. This group provides 20-hour, 5-day on-site coverage and on-call standby coverage for after hours. To

maintain the reliability goals, preventive maintenance is very closely controlled, and on-line diagnostics and analysis are emphasized. A primary component in the reliability and availability of the hardware is the physical environment in which it exists. Accordingly, a significant amount of time and resources is expended in ensuring that the best, most cost-effective environmental controls are used at all facilities that ISI services.

14.2.2 System Software

The software group's primary goal is to install and maintain, at maximum reliability, ISI's VMS, UNIX and TOPS-20 operating systems and applications software. In order to accomplish this goal, the group provides 24-hour, 7-day coverage to analyze system crashes and to provide appropriate fixes. In addition, it is the group's responsibility to install, debug, and modify both the latest monitor versions and the associated subsystems available from the vendor.

14.2.3 Operations

The operations staff is responsible for operating the computers and overseeing the systems and the environment. There is 24-hour, 7-day on-site coverage at the Marina del Rey facility. One of the primary responsibilities of the group is to provide protection and backup for user files such that there is virtually no possibility of loss of data integrity or significant information. This goal is achieved through a variety of means, including regularly scheduled full and incremental backups of all systems; permanent archivals of requested or infrequently accessed system and user files; tape storage and pointers to all information extant at the time of removal of directories from the various systems; and, perhaps most important, redundant offsite storage of all significant information active on the disk structures or existing on tape within the facility. When a problem occurs, the on-duty staff isolates it and takes appropriate action. On the night and weekend shifts, the operations staff responds directly to user inquiries.

14.2.4 Network Services

Network Services, the ISI customer service group, provides a two-way communication link between the users and consulting staff. This is accomplished by maintaining a 12-hour, 5-day on-duty staff for prompt problem resolution and rapid information exchange, both on-line and by telephone. The group offers introductory training in the use of both the hardware and software tools available on the ISI systems, as well as providing documentation for new users of the ARPANET. Network Services also assists in the formulation of user training programs for large, ARPANET-based military experiments, for example, the U.S. Army XVIII Airborne Division Army Data Distribution System at Fort Bragg, North Carolina, and the C3/ARPANET Experiment at Strategic Air Command Headquarters at Offutt Air Force Base, Nebraska. Appropriate documentation is constantly being generated and distributed to individual users, as well as to remote user-group liaison personnel; this documentation ranges from simple, novice-level explanations to more technical information suitable for experienced users. In accordance with IPTO guidelines, the customer-service group provides regular accounting data to DARPA.

14.3 PROGRESS

The past year has been a very successful one for the Computer Center. Average uptimes were 99 + percent, and several major transactions have occurred without great difficulty. We saw for the first

time the results of the modernization effort undertaken in 1982, made major advances in our local networks implementations, and successfully transitioned to the TCP/IP suite of network protocols.

14.3.1 Environmental Changes

In late 1982 we completed a major facility renovation that included new wiring, conditioned power, air conditioning, and fire protection equipment. During 1983 we saw a substantial decrease in crashes due to environmental conditions. During FY'82 there were 107 environment-related crashes (power fluctuation, power failure, air conditioning failure). During FY'83 there were only 20 environment-related crashes, a decrease of 81 percent. In the last seven months of operation there were no environmental crashes.

14.3.2 Hardware Additions

Over the past two years the ISI Computer Center has undergone a major change in direction. Previously dominated by DEC PDP-10 computers, the facility was a relatively simple support environment consisting of a uniform collection of hardware and software. The current collection of equipment reflects the New Computing Environment project plan of moving towards professional workstations and rear-end servers. The new collection of systems is a substantially more complex support problem involving two local networks, nine different processors, and seven operating systems:

Quantity:	*Manufacturer:*	*Model:*	*Operating System:*
6	DEC	PDP-10	TOPS-20
1	DEC	VAX 11/780	VMS
1	DEC	VAX 11/780	UNIX
7	DEC	VAX 11/750	UNIX
3	DEC	VAX 11/750	VMS
8	XEROX	8010	STAR
3	XEROX	1100	INTERLISP BASED
3	3-RIVERS	PERQ	UNIX
1	SYMBOLICS	LM-2	QLISP BASED
5	SYMBOLICS	3600	QLISP BASED
12	IBM	PC	MS-DOS

In addition, two more DEC PDP-10s were supported by the ISI staff at remote sites (Gunter AFB and NOSC). The current local hardware configuration is shown in Figure 14-1, and the configuration of the hardware at NOSC is shown in Figure 14-2.

14.3.3 System Software Enhancements

During FY'83 the major effort centered upon installation and debugging of the TOPS-20 systems for the conversion from NCP network protocols to the new DoD standard TCP/IP suite of protocols. This work was a joint effort of Digital Equipment Corporation, Bolt Beranek and Newman, Inc., and ISI. ISI developed the bulk of the applications-level programs (SMTP mail, TN, FTP, etc.) and also assisted in the debugging of the two operating system modules.

In addition, during FY'83 backup procedures were developed for SAC scheduling, preparation was made for the ARPANET/MilNet split, the SA10 controller software was enhanced, the last TENEX

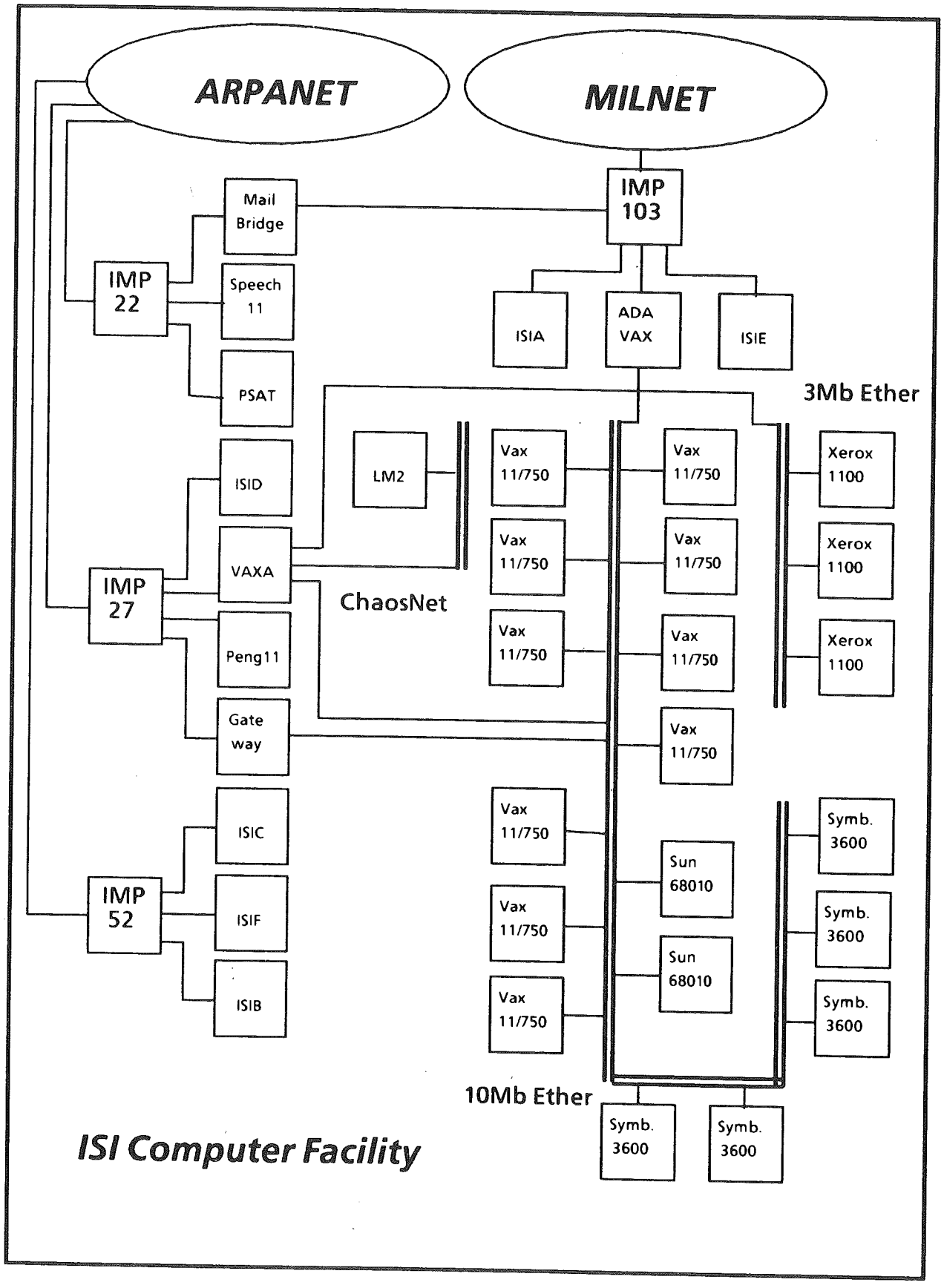


Figure 14-1: Diagram of local ISI ARPANET facility

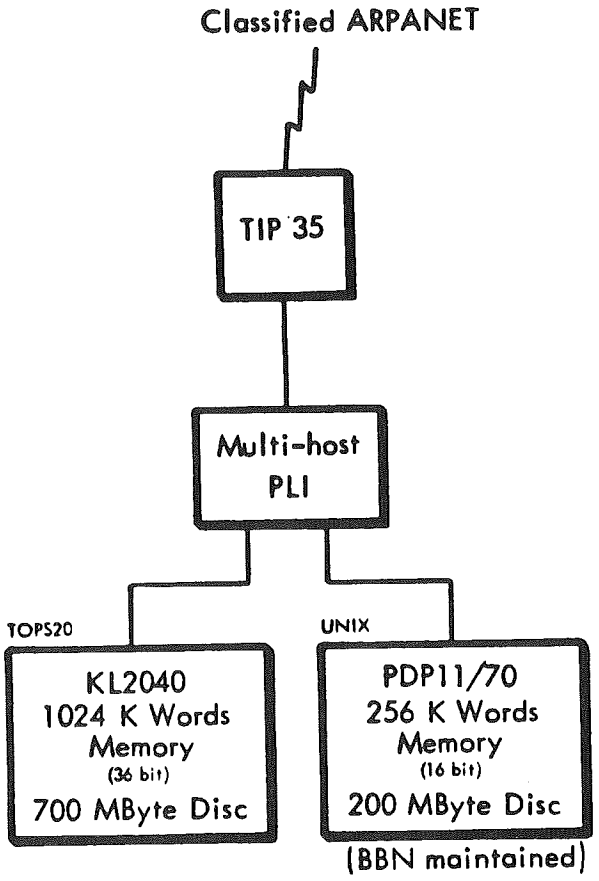


Figure 14-2: Diagram of remote ISI ARPANET facility at NOSC

system was replaced by a KL running TOPS-20, support was added for new tape drives, RSX-20F floppies were developed for Gunter-Adam, dial-out capability was added for the VLSI project and the ISI business office, an accounting system was developed for the ADA-VAX, VMS 3.1 and VMS 3.2 were installed on the ADA-VAX, UNIX 4.1 was installed on the UNIX systems, and numerous bug fixes were made on the System and user Subsystem programs.

14.4 MILITARY IMPACT

ISI's computer centers provide ARPANET cycles 24 hours a day, 7 days a week to the Strategic Air Command, Naval Ocean Systems Center, and Fort Bragg. In addition to supplying machine time, this project has provided additional support in the following areas:

- Training, documentation, and modifications as requested by user groups for NLS.
- Planning support and training in ISI systems software for the installation of an on-site DEC System 2060 at Gunter Air Force Base.
- Support for the production of AFM 67-1 with NLS.

14.5 FUTURE WORK

The Computer Research Support project will continue to provide computing service to the DARPA research community, provide and support software packages for the ARPANET community, and offer a program of technology transfer and user education through the Network Services group.

15. STRATEGIC C3 SYSTEM EXPERIMENT SUPPORT

Research Staff:
Gary McGreal
Victor Ramos

15.1 INTRODUCTION

DARPA has defined an experiment in Strategic C3 systems to be conducted in cooperation with the World Wide Military Command Control System (WWMCCS) System Engineer (WSE) and the Strategic Air Command (SAC). The concept of the experiment is to demonstrate and evaluate the use of new technologies (such as the ARPANET, packet radio, network security, and distributed knowledge-base techniques) for strategic command, control, and communication. Specific goals are to

- Demonstrate and evaluate the survivability of multinode computer-communication networks, including the ARPANET and packet radio, especially for remote access from both airborne platforms and surface sites.
- Explore replication and reconstitution of critical knowledge bases on this network in the face of a loss of a large number of links.
- Demonstrate and evaluate the rapid reconstitution of a network by rapid deployment of packet radio nets to reestablish connectivity between surviving elements of the network.
- Conduct experiments and exercises to evaluate the utility of such a network on a distributed knowledge base to support postattack C3 activities.

The DARPA experiment is defined as having three phases:

1. Phase I is planned to demonstrate air-to-surface packet radio links and gateways into the ARPANET as a first step in evaluating the feasibility of a truly survivable strategic network.
2. Phase II is directed toward creating a survivable message system and data bases through multiple copies of the critical components and data across the ARPANET.
3. Phase III will address the feasibility of rapid reconstitution of a strategic network by deployment of packet radio networks to reconnect surviving elements of the network.

15.2 PROBLEM BEING SOLVED

ISI's major portion of the above plan is to provide an initial core of necessary facilities (ARPANET/MILNET access, host systems, various software tools, Network Services support, etc.) to allow SAC personnel to gain experience with this technology and to ensure the success of the experiment. Specifically, SAC must have fairly broad-based experience with ARPANET-based on-line interactive computing. Installation of modems, installation of 30 or more interactive CRT terminals, user training, system software training and support, and on-site maintenance of equipment are part of the continuing program.

15.3 PROGRESS

The major activity at SAC itself during this period, outside of the normal datacom, experiment, and Network Services support, revolved around the ARPANET/MILNET split. The SAC user community has been growing regularly since its inception. As a result, existing resources have been strained. An additional TAC was installed in December to support the growing user community. This summer the entire ISIE facility was dedicated to SAC use, doubling their available computational capacity.

A SAC data communications plan, developed by ISI and proposed in November 1982, would triple the maximum number of SAC users able to access the ARPANET/MILNET. The wiring plan was accepted and the work will be completed in August 1983.

In addition, during this period new resources were obtained or redistributed to support the VAX van effort (the effort is a portion of the experiment testing the mobility and reconstitutability of a partially destroyed network). Additional ARPANET access was provided to SRI International (which was working the Telecommunications portion of this effort and housed the vans) via 9600 bps communications lines. A plan for reallocation of C3 terminal assets, identified for use on the VAX van by the Headquarters Emergency Relocation Team (HERT) participants, was approved by the Chief of Staff (Monroe W. Hatch) and the Deputy Chief of Staff for AD (James L. Crouch). ISI also supplied three new printers for the vans.

15.4 FUTURE WORK

ISI will continue to assist DARPA in planning this program, working to satisfy the communication and computer resource requirements of SAC headquarters. In particular, ISI will do the following:

- Continue to provide on-site maintenance support for the required equipment.
- Continue to plan and assist in implementing improved SAC connectivity to the ARPANET.
- Install and maintain terminals and communication equipment for the connection of two Air Force bases to the ARPANET to allow participation in the experiment.
- Continue to supply programming support to users at SAC headquarters.

ISI will provide an on-site technician at SAC, who will be responsible for the identification of system malfunctions and for primary maintenance of on-site equipment. He will be supplied with required spare parts and will have maintenance contracts with the equipment vendors. Further support will be available from ISI in terms of additional spare parts, systems expertise, and documentation as necessary. The on-site maintenance technician will also be responsible for the off-site terminals at Peterson Air Force Base, Barksdale Air Force Base, March Air Force Base, and any new locations. The on-site technician will coordinate requests of SAC AD with SRI International and SAC (XPFC) under the supervision of ISI management. The technician will provide training and consulting with backup from ISI as required.

ISI will provide program planning assistance to DARPA. We will continue to investigate the data requirements for SAC Headquarters, the HERT effort, and the ACCESS system (SAC Executive Data Processing System). As specific research tasks are defined, ISI may choose to submit new proposals to address one or more of these tasks.

16. TCP/IP IMPLEMENTATION SUPPORT

Research Staff:
Gary McGreal
Joel Goldberger
Dale Chase
Craig Rogers

16.1 PROBLEM BEING SOLVED

ISI's user base is distributed across the entire continental United States, with a few users communicating via satellite connections to Europe and Hawaii. In the near future, further expansion is expected that will allow access to Japan and Korea. This access is obtained by the connectivity of the ARPANET, the MILNET, and the Internet. These networks are accessed via the Internet Protocol (IP) and the Transmission Control Protocol (TCP).

The Department of Defense has adopted the Internet concept, and the IP and TCP protocols in particular, as DoD-wide standards for all DoD packet networks. The DoD will be converting to this architecture over the next several years.

The role of the TCP/IP Implementation Support project is to assist in placing these protocols into active use in the operational ARPANET and MILNET by installing and debugging host software to support the ISI user community.

16.2 GOALS AND APPROACH

This is a group effort involving programmers and researchers from ISI, Bolt Beranek and Newman, Inc., Digital Equipment Corporation, UC Berkeley, SRI International, Stanford, the Massachusetts Institute of Technology, and other institutions. The ISI TOPS-20 implementation was based on monitor sources developed at BBN. ISI, with some assistance from Stanford and DEC, debugged the monitor code and installed the principal services: Telnet, File Transfer, and Mail.

The UNIX kernel modifications were developed by BBN and Berkeley; ISI installed this software on the local UNIX systems and adapted existing services to the new protocols.

A VMS implementation was adapted from the UNIX code running in the SRI-developed EUNICE environment, which emulates UNIX operations and kernel functionality. This was installed in the ISI VMS hosts using locally developed network drivers.

16.3 PROGRESS

In preparation for the DARPA-directed conversion from NCP to TCP, which took place on January 1, 1983, a major effort was made to convert existing network utilities and develop TCP-specific tools on our six TOPS-20 systems and two VAX 780s running VMS and UNIX. The existing utilities included the mail system, local line printer spooling, system monitoring, user inquiry (FINGER), TELNET, FTP, and several database backup facilities operated for the Strategic Air Command (SAC) and Fort Bragg.

New utilities that were developed include TCP-specific status and monitoring tools and assorted servers (time servers, name servers, and the like). Both before and after the conversion we worked closely with BBN to test and improve new releases of TOPS-20 system and application TCP code. We worked with SRI on the VMS support, and with both BBN and Stanford on the UNIX support. A number of crippling bugs in the TOPS-20 implementation were found and corrected, and performance was improved significantly as a result of this collaboration. We also served as a distributor of a number of TCP utilities to the rest of the DARPA community, making our TOPS-20 sources available to any interested party.

We also imported and adapted the Berkeley 4.1a UNIX implementation of TCP/IP and the MIT Remote-Virtual-Disk protocol to provide a usable environment on seven VAX 750s. Most of this development was done by the New Computing Environment project, but support was provided by the programmers working directly on TCP/IP as well. Another task completed in support of this project was the installation of a DEC PDP-11/23 ARPANET/Ethernet gateway, which provides network connectivity for these VAXes. Several utilities developed to monitor this gateway have proved useful in identifying and diagnosing some of its problems.

In an effort to identify and correct network response problems affecting SAC, Fort Bragg, and IPTO users, we have again collaborated with BBN to install an extensive measurement package in our TOPS-20 systems. The results of this measurement have already been used to direct software improvements. We expect further results and additional improvements in the months ahead.

16.4 IMPACT

ISI currently supports eight DEC TOPS-20 systems connected to the ARPANET/MILNET: six at the Marina del Rey Center, one at the Naval Ocean Systems Center (NOSC) in San Diego and one at Gunter Air Force Station in Alabama. These systems support approximately 2,500 government, military, and subcontractor users. TCP/IP protocols are used to connect these users to the ISI systems.

16.5 FUTURE WORK

We will continue to work with BBN to identify the bottlenecks that are causing poor network response on the TOPS-20 machines. In addition, problems in the TOPS-20 TCP code that are causing both system crashes and throughput degradation continue to be identified and corrected. Effort is also continuing to improve the reliability of the DEC PDP-11/23 Gateway. As we add hosts that use TCP/IP to our Ethernet, we have uncovered problems in both the TOPS-20 and VAX implementations. We continue to correct these problems as they appear. The hosts include IBM-PCs, VAXes, SUN workstations, and eventually Xerox Dandelions. These efforts are drawing on resources from both systems staff and the staff of the New Computing Environment project. Our desire is to provide a unified environment that supports all of the different processors at ISI.

We are also investigating alternate gateway implementations that will allow us greater control over this resource than we are allowed using the implementation supplied by BBN. These alternatives include implementations done by Dave Clark at MIT and Dave Mills at Linkabit.

We are in the process of connecting our TOPS-20 systems directly to the Ethernet using the Massbus Ethernet Interface System (MEIS) developed at Stanford University. This is presently

connected to the 3MBit Ethernet, but Stanford is planning to convert it to the 10MBit Ethernet by the end of this year. To achieve this connection, changes are required in both the hardware (as it came from Stanford) and in the TOPS-20 software. Even on the 3MBit Ethernet we will be able to use this hardware in a number of ways, since we have other processors (VAXes and SUN workstations) also constrained to use this network.

We are anxiously awaiting release of the DEC TCP JSYS interface, which promises to provide a much cleaner interface to the TCP/IP facility than the current BBN implementation. Once this is delivered, considerable effort will be required to convert the existing utilities to make use of it.

ISI will carry out support and development of TCP/IP protocol implementations on the computer systems at ISI as required. Specifically this will include work on throughput problems, coordination with BBN, reliability enhancements, work on the local area net system, support of the UNIX and VMS systems, and work on the TOPS-20 JSYS interface.

17. EXPORTABLE WORKSTATION SYSTEMS

Research Staff:
Gary McGreal
Joel Goldberger
Jim Koda
Dale Russell
Tom Wisniewski

17.1 PROBLEM BEING SOLVED

Portability and survivable communications are key concerns in the command and control environment. To date, portability of data processing equipment has been complicated by the size and power requirements of even the smallest minicomputer. With the advent of single–chip microprocessors (such as the Motorola M68010), it is possible to develop command workstations that are readily portable and can be installed in vans or airborne command stations.

Over the next few years, workstations in the office, in the research community, and in the military will generally be used as part of an interconnected system that will include access to local and national networks, printers, file servers, database hosts, and large timesharing computers supplying special–purpose or volume processing (LISP cycles, fast floating point, peripheral array processors, etc.).

For survivability purposes, it is important that databases (logistics data, weather data, etc.) be redundant and distributed. Local area networks connected to long–haul networks such as DDN (or packet radio networks) will provide communication connectivity to these distributed resources.

The exportable workstation systems will provide a model for this environment and serve as a test site for new hardware and software developed in DARPA–IPTO programs. The Workstation project, by installing these Sun Microsystems, will support IPTO administrative functions. In addition, these systems will serve as a proving ground for software developed by ISI and other DARPA–IPTO contractors.

17.2 GOALS AND APPROACH

The ISI Exportable Workstation Systems project has proposed a testbed workstation for installation at the DARPA–IPTO office in Arlington. This workstation, the SMI SUN (a Motorola 68010–based system with graphics hardware and display), runs UNIX–based software. ISI will install fourteen workstations, which will be connected to a local Ethernet and a VAX file server. TOPS–20 systems will be maintained at ISI in Marina del Rey as remote servers and database hosts. We expect workstations to significantly augment existing technologies and services.

This plan has the following potential advantages:

1. Avoidance of network delays in character transmission.
2. Avoidance of occasional high loads due to overloading or some other problem on the timeshared TOPS–20 hosts.

3. Ability to use either workstation or remote host software tools.
4. Use of a workstation as an agent between the user and redundant databases on multiple hosts.
5. Ability to use high-resolution graphics on the local display.
6. Use of the workstation for security; the workstation could locally maintain keys for encrypted data stored in hosts. The keys need never appear in the host or the Internet.

17.3 PROGRESS

This project was initiated late in the fiscal year, but some progress has been made on the prototype hardware and software. Two prototype systems have been installed, one at DARPA-IPTO and one at ISI. UNIX is operational and the systems are on local Ethernets accessing the Internet via BBN (Bolt Beranek and Newman Inc.) gateways. Primitive administrative programs are available for mail and editing. More sophisticated software is being developed and is on order.

17.4 IMPACT

This project serves as an example and existence proof of command and control, office automation, and communications technology, all of which are critical to the military community. It will demonstrate the viability of communications in an operating Internet environment, the first step towards mobile and distributed command and control systems. It will demonstrate the viability of using redundant databases for backing up applications in the face of partitioned networks and inoperable systems. The system will also show some advantages of using local area networks over using low-bandwidth access to interactive timesharing systems.

17.5 FUTURE WORK

The operating system will be developed, using SMI's UNIX offering as a base and adding functionality and services as required. When possible, we plan to use existing software as the basis for new tools. In adapting existing software and creating new software, we offer a clean structure for building tools, which minimizes cost and provides building blocks for future enhancements.

17.5.1 Redundant Mail Cache

A fully automatic and distributed message system is a very complex and important research problem with broad implications for the military community. An ideal solution will take a good deal of time, research, and development. In order to lower costs and get the system operational as soon as possible, we propose a more modest engineering solution. Our plan is to build a system in which a single host is responsible for maintaining all redundant copies of a particular file; whenever the master copy is updated, the redundancy manager in the master copy's host is responsible for updating all "registered" copies.

In such a system, redundancy guarantees that a recent consistent version of any file is available, but updates are not distributed until the master host for a particular database is up. This is completely adequate for most program libraries; in the case of mail, it means that a user can access all messages that have arrived at his master mailbox *even if the corresponding host is down.*

A DEC VAX will be the master mailbox host. Mail received by the VAX will be "tagged" as received and sent on to the TOPS–20 redundant mailbox. New mail that has been sent directly to the TOPS–20 host will be forwarded to the VAX for processing by the master mailbox. Changes to the mailboxes on either host will not be reflected in the corresponding host's mailbox. In general, the user will manipulate, read, and delete messages from the VAX host. The TOPS–20 mailbox will remain an archive cache of all messages received in the past unless the user chooses to handle them in some special way.

17.5.2 Mail Services

The main component of the mail support on the SMI systems will be a mail preparation program similar to those existing on TOPS–20. We will attempt to mimic the user interface of Hermes, since that system seems to be in widest use on the TOPS–20 systems. As a starting point we will use the "C" based mail handler MH. Substantial changes must be carried out on the user interface, and we plan to add the minimal functions found in even simple TOPS–20 mail handlers such as MSG. We do not plan to add the full range of capabilities found in mail handling "systems" such as Hermes or MM. In general, we feel that the editors, mail handlers, and applications–level utilities will eventually be replaced by programs that take full advantage of the workstation's bit–map display and "mouse" I/O device. Our short–term goal is to provide services commensurate with those already available from TOPS–20 or UNIX, not to augment or heavily invest in programs based on antiquated line–at–a–time user interfaces.

17.5.3 Archive and Backup System

A number of problems are inherent in operating a local network in an office environment. On one hand, it is desirable to avoid the noise, high support costs, need for an "operator," and space used by the traditional tape drive and dump procedures; on the other hand, the requirement for data integrity and systems backup remains the same.

The logical solution is to centralize and remote this capability. In the future, this will probably account for a good deal of network FTP traffic. In this area the IPTO system will be breaking new ground. While some research and even implementations exist (work at Bell Labs, CCA Datacomputer, etc.), a truly tapeless (locally) archive and backup system has not been used operationally.

The archival system will be tied into the TOPS–20 systems at ISI, which will make it possible to share the operator time (and expense) required for tape mounts, etc., across a large existing facility, and possibly over several other local net environments. In addition, existing TOPS–20 software will be used for the bulk of the end user transactions, avoiding a very large development cost.

Each user will have an archive directory on the TOPS–20 system. A series of programs and commands will be written on the UNIX side to move files across the net and to evoke the appropriate TOPS–20 command. A background process will observe when files are retrieved on the TOPS–20 side and FTP them across the net to their appropriate UNIX directories. For example, if the user wants to archive a file on the UNIX system, he will type the following command:

$ archive filenm

The UNIX archive program will open a connection to TOPS–20 and copy the UNIX file /usr/user/filenm into the TOPS–20 file <IPTO–NET.ARCHIVE.USER>filenm..1. Further, the program will initiate an @archive filenm..1 command to TOPS–20. If a user wants to examine his archive file directory, he can give a command such as

$ dir m*

The UNIX–based program will then print out the current list of archived files that start with the letter "m."

To support backup of the file system, programs will be required to do disk–to–disk incremental and full dumps. Given that Berkeley will be writing drivers for the RUA–81 and RUA–60 disk drives (the new de facto DEC standard), this effort should be minimal.

18. NEW COMPUTING ENVIRONMENT

Research Staff:
Gary McGreal
Joel Goldberger
Michael Butler
Dale Chase
Jim Koda
Craig Rogers

18.1 PROBLEM BEING SOLVED

The New Computing Environment (NCE) project's goal is to adapt developing computer technologies to serve the research and military user communities. The resulting model computing environment will serve four purposes; it will

1. provide a very large improvement in languages, system support, and additional computing cycles to the ongoing ISI research effort;
2. serve as a testbed and eventual existence proof for the implemented technology;
3. serve as a proving ground for local computer environments targeted for DARPA and the military community, as well as a device for investigating new research ideas that will eventually be adapted by those communities; and
4. allow for experimentation and realization of command and control requirements for an environment that can exist in mobile command centers, given the small size, portability, and local computing capability of personal computers.

18.2 BACKGROUND

For convenience in formulating a generic model of a computing environment, computational activity can be split into two major types: interactive computing and CPU–bound computing. Our model assumes that the user will have at least the computing capacity in his personal computer (PC) to allow him to do all interactive computing locally. If the PC we acquire is fast enough, it may be able to do CPU–bound computing as well. If the frontend PC is not fast enough, we will need more powerful rear–end servers. All current scenarios for a personal computing environment include some combination of PCs and servers.

Generally, interactive computing is considered to include editing, mail handling, administrative tasks (interactive calculations, spread sheets, calendar maintenance), directory maintenance, conferencing, small program generation/debugging, and system status checking. CPU–bound computing includes LISP programming, large program compiles, large program execution, large file transfer, sorts, searching, and database management.

The guiding principle in the design of this environment is full use of current hardware technology as it is represented by personal computers, bit–mapped displays, and pointing devices (mice). A common failure in the use of new hardware technology is to drag along the prejudices and limitations of operating system software that was suitable for an earlier hardware base. Current applications software rarely takes advantage of the now commonplace features of smart terminals (blinking characters, highlighting, reverse video, etc.). We consider an environment that further perpetuates such a teletype–oriented user interface onto bit–mapped displays to be an unacceptable waste.

Human engineering is an area that has been frequently overlooked in computer system implementations. Historically, operating systems have evolved from a level of low complexity/functionality to high complexity/functionality, not by following a coordinated plan, but rather as the result of the programming requirements of individual users. Functionality has been tacked on, rather than integrated into systems.

Computers are a tool to aid in the solving of problems. Time wasted in dealing with a poorly implemented system detracts from that system's potential benefits. The hidden costs of poor human engineering include time wasted learning non-mnemonic commands, time wasted deciphering bad documentation, time spent tracking down errors, work lost through system malfunction or inadvertent deletion, time spent training new users, and the cost of staff who should use the computers, but who do not use them because of difficulties resulting from bad design.

18.3 GOALS AND APPROACH

Current hardware technology as it is represented by programmer workstations includes bit-mapped displays, pointing devices, relatively high-speed local processing, and a local area network with specialized servers. Servers can be used for high-resolution printing, optical character readers, file servers, plotters, communications, or special-purpose processing (LISP machines).

Users who are already trained on one computer system are frequently frustrated by the design flaws of another. To be successful, NCE must offer a large, tangible improvement over existing systems (particularly TOPS-20) and at the same time minimize retraining of users familiar with other environments. NCE must perform the following functions in support of its users:

- Word processing support
- Network connectivity
- Electronic mail services
- Simple programming capability
- Large program support
- Graphics services
- Administrative support software
- File server and backup support
- System support

Specifically, then, our goal is to create a workstation environment that will allow research and administrative work at ISI to be carried out on a network of personal computers. After our general NCE environment matures, a major goal of exporting this knowledge and technology to a military setting will be attempted. At this stage in the NCE development, we have the following goals and evaluation criteria:

1. We plan to use the current hardware technology to maintain access to various servers and to take advantage of features of the new "smart" terminals.
2. We plan to make full use of recent developments in the understanding of human engineering, and we expect to save significant amounts of time that would ordinarily be wasted on old-style non-integrated functionality, haphazard user training, and non-rigorous system implementations.
3. We expect to offer a total system with tangible improvements over TOPS-20 and similar operating systems, so that the NCE will be the system of choice for most users.
4. We expect to provide a non-Xerox file server for the NCE.

18.4 PROGRESS

To date, a variety of possible hardware bases have been analyzed to determine their suitability to our evaluation criteria. Processors that have been reviewed include VAX 11/730s, Perqs, Apollos, Wicats, Suns, Xerox 1100s (Dolphins and Dandelions), and Symbolics 3600s.

One strong candidate as the processor most suitable for our needs is the Xerox Dandelion workstation, running with the Mesa programming environment and the Star operating system software. This processor offers a combination of high speed, low cost, sophisticated software, and integrated printing peripherals not as easily obtained from other vendors' products.

Moving the entire ISI community to this product presents several problems, some of which may not be resolvable. However, a number of critical basic functions are fully implemented in the Xerox Dandelion–based environment:

- **Word processing support.** This requirement is fully met by the Dandelion/Star system. The bit–mapped display and editor supports a high–resolution, multiple–font environment with an extensive graphics capability. The command interface makes good use of the hardware, including extensive use of the pointing device (mouse).
- **Programming capability.** We have negotiated access to both Interlisp and MESA, sophisticated programming environments that meet the needs of most of the ISI research staff.
- **Graphics services.** There is full support for graphics, windowing, multiple fonts, and access to the mouse. The Xerox Raven printer provides high–resolution laser output.

The decision to base the local area net on a 10MB Ethernet has provided a firm basis for achieving the requirements of the NCE system. The local network is accessible to UNIX–based VAX computers, workstations, and IBM PCs. A diskless UNIX system has been implemented, using Berkeley 4.1a UNIX running on VAX 11/750s with a Remote Virtual Disk protocol developed at MIT. Several projects require access to highly portable software. These UNIX systems support both "C" and "MAINSAIL," and they will eventually be accessible as backend servers. A number of projects are using these systems as single–user Interlisp workstations.

18.5 IMPACT

Portability and survivable communications are key concerns in the command and control environment. To date, portability of data processing equipment has been complicated by the size and power requirements of even the smallest minicomputer. With the advent of single–chip microprocessors, it is possible to develop command workstations that are readily portable and can be installed in vans or airborne command stations.

For survivability purposes, it is important that databases (logistics data, weather data, etc.) be redundant and distributed. Local area networks connected to long–haul networks such as DDN (or packet radio networks) will provide this connectivity.

In addition, local area nets and administrative workstations will gain widespread use in the military over the next few years. A number of major vendors are adapting Ethernet as the basis of their local area network technology. The NCE project will enable these systems to communicate with DoD long–haul networks via the required TCP/IP protocols.

18.6 FUTURE WORK

18.6.1 Electronic Mail

The most important work to be done at this time (assuming that the Xerox Dandelion is our final choice) is integrating TCP/IP protocols into the XNS (Xerox Network System) environment. It would be a mistake to replace all of the existing XNS protocols with TCP/IP, as this would disrupt the integrity of the existing software. We will want TCP/IP to run in individual workstations in much the same way that Xerox has both XNS and PUP (PARC Universal Packet) coresident; that is, we will build a set of tools that allow the workstations to communicate directly with IP hosts rather than having to go through a translating gateway. This will allow access to DoD Internet hosts via FTP and Telnet. In addition, a translating gateway may be built.

Since one of our goals is to provide file service on a non-Xerox processor in order to provide a more robust archive mechanism, we will probably develop a system which will allow this same processor to handle some subset (if not all) of the translation duties. We will obviously need such an intermediary to handle mail from the DoD Internet, since our only viable option with mail on the Dandelion is to use the Star mail system. We may provide a Mesa environment interface to the Star mail, but that has no impact on the problem. We will also require this processor to allow access to the Star file server from the Internet. The TCP/IP module will also be used by the Internet project for the development of Multimedia mail and for general protocol development. We will make inquiries of ACC and Interlan regarding their implementations of XNS protocols on the VAX, and we await details on the current Xerox VAX/XNS effort. We have always been wary of using protocol packages for which we could not acquire the sources, and this would apply in this case as well.

In order to provide mail service, we will need to modify Star mail service to allow simplified addressing of ARPANET mail. That is, we would like to be able to say Postel@ISIF, rather than to specify the address in some other way to direct it to our mail gateway. This would render a mail system rather like the present Hardy/Grapevine system, in use at Xerox, which allows addressing to either ARPANET or Xerox Internet recipients in the style of the appropriate network. Since the easiest interface into the Star mail system is at the user level, we would provide either a Xerox 8000, or more likely a VAX, the ability to poll the mail-server at regular intervals and accept mail bound for ARPANET recipients. The server would then retransmit the mail using the DoD SMTP protocol over the DoD Internet using TCP/IP. Incoming mail would be addressed to User:Domain:Organization@Mail-Gateway, in the same way that mail to people on the Grapevine network at Xerox is presently addressed to User.Registry@PARC-MAXC. The return address would be filled in by the mail gateway as mail left the Star network.

18.6.2 Hardcopy Output

Several text formatting applications presently available on our DEC 20s are unlikely to migrate to Dandelions, and we would like to be able to make use of the new line of printers available for the Star network, all of which use the InterPress format. To this end we need to write conversion tools or modify our existing applications to produce InterPress masters directly on the DEC 20s. These InterPress masters would then be transmitted to the Star printers via our gateway, which would have to be educated to talk to the print server via XNS protocols. We presently have a similar arrangement in place to print files on a Xerox Penguin.

18.6.3 Home Terminal Access

The researchers at ISI are accustomed to being able to access our systems from home, and they are reluctant to sacrifice this facility. We plan to provide for this capability, although the precise mode of connectivity has not yet been decided on; it could be either simple TTY-style interaction, as is provided by the present Star ITS service, or an implementation of the XNS protocols that are used over the phone.

18.6.4 Remoting the Display/Keyboard from the Processor

Since our offices are not adequately air-conditioned and since the Dandelion in its present form is very noisy, we would like to be able to have the display/keyboard remoted from the processor. To allow this, we must acquire sufficient technical documentation to allow us to modify the hardware of the Dandelion. We will pursue negotiations with Xerox to obtain the required documentation and will carry out the necessary hardware development.

18.6.5 Access to LISP Engines

We presently have three Xerox 1100s, two Symbolics LM-2s, and five Symbolics 3600s, all of which have rather elaborate window packages. Because we will never be be able to provide a machine of this power to every researcher who desires one, we will have to use them on a sequentially timeshared basis. It is our desire to make these high-powered LISP engines available via the workstations in people's offices. This requires access at a low level to the window packages on each of these backend machines, and a facility to replicate the windowing functions on the Dandelion. The LISP environment on the Dandelion provides sufficient access to these low-level windowing primitives.

18.6.6 Porting of the Final Product

Our eventual goal is to create and maintain a workstation environment that allows the research and administrative programming at ISI to be carried out on the personal computers. When this occurs, the system and its programs will be available for porting to military environments. It will arrive as a well-tested and reliable environment.

19. PUBLICATIONS

Books, Chapters, and Journal Articles

1. Balzer, R. M., Don Cohen, M. S. Feather, N. M. Goldman, W. R. Swartout, and D. S. Wile, "Operational specification as the basis for specification validation," in D. Ferrari, M. Bolognani, and J. Goguen (eds.), *Theory and Practice of Software Technology*, North–Holland, 1983.

2. Cohen, Don, W. R. Swartout, and R. M. Balzer, "Using symbolic execution to characterize behavior," *ACM Sigsoft Software Engineering Notes* 7, (5), December 1982, 25–32. Working papers from the ACM SIGSOFT Rapid Prototyping Workshop.

3. Cohen, Danny, and V. Tyree, "Quality control from the silicon broker's perspective," *VLSI Design* 3, (4), July/August 1982, 24–30.

4. Cole, E. R., "Packet voice: When it makes sense," *Speech Technology* I, (3), September/October 1982, 52–61.

5. Feather, M. S., "Program specifications applied to a text-formatter," *IEEE Transactions on Software Engineering* SE–8, (5), September 1982, 490–498.

6. Lipkis, T., "Descriptive mapping for explanation production [abstract]," *SIGART Newsletter*, (85), June 1983.

7. Mann, W. C., "Systemic encounters with computation," *Network: News, Views and Reviews* 7, May 1983, 27–33.

8. Mostow, D. J., "Learning by being told: Machine transformation of advice into a heuristic search procedure," in J. G. Carbonell, R. S. Michalski, and T. M. Mitchell (eds.), *Machine Learning*, Tioga, Palo Alto, California, 1982.

9. Sondheimer, N. K., and N. Relles, "Human factors and user assistance in interactive computing systems: An introduction," *IEEE Transactions on Systems, Man, and Cybernetics* SMC–12, (2), 1982, 102–107.

10. Sondheimer, N. K. (ed.), *Tutorial on Natural Language Interfaces,* Association for Computational Linguistics, 1983.

11. Stefik, M., J. Aikins, R. M. Balzer, J. Benoit, L. Birnbaum, F. Hayes-Roth, and E. Sacerdoti, "Basic concepts for building expert systems," in F. Hayes-Roth, D. Waterman, and D. Lenat (eds.), *Building Expert Systems*, Addison–Wesley, 1983.

12. Stefik, M., J. Aikins, R. M. Balzer, J. Benoit, L. Birnbaum, F. Hayes-Roth, and E. Sacerdoti, "Architecture of Expert Systems," in F. Hayes-Roth, D. Waterman, and D. Lenat (eds.), *Building Expert Systems*, Addison–Wesley, 1983.

13. Sunshine, C., "Guest editorial: Protocol specification, testing, and verification," *Computer Networks* 6, (6), December 1982, 375–376.

14. Swartout, W. R., and R. M. Balzer, "On the inevitable intertwining of specification and implementation," *Communications of the ACM* 25, (7), July 1982, 438–440.

15. Weischedel, R. M., and N. K. Sondheimer, "Meta-rules as a basis for processing ill-formed input," *American Journal of Computational Linguistics* 9, (3–4), 1983.

Refereed Conference Proceedings and Papers

1. Balzer, R. M., D. Dyer, M. Fehling, and S. Saunders, "Specification–based computing environments," in *Proceedings of the Eighth International Conference on Very Large Data Bases*, IEEE, Mexico City, September 1982.

2. Bates, R., D. Dyer, and J. A. G. M. Koomen, "Implementation of Interlisp on the VAX," in *Conference Record of the 1982 Symposium on LISP and Functional Programming*, ACM, Pittsburgh, August 1982.

3. Bisbey, R., II, D. Hollingworth, and B. Britt, "A network graphics system for command and control," in *Proceedings of the Symposium on Interoperability of Automated Data Systems*, North American Treaty Organization, The Hague, Netherlands, 1982.

4. Casner, S. L., Danny Cohen, and E. R. Cole, "Issues in satellite packet video communication," in *International Conference on Communications (ICC'83)*, pp. 34–38, IEEE, Boston, June 1983.

5. Cohen, Danny, "The impact of VLSI on peripheral array processors," in *Proceedings of the Conference on Peripheral Array Processors*, Simulation Councils, San Diego, California, October 1982. *Simulation Series*, Vol. 11, No. 2, pp. 33–38.

6. Cohen, Danny, and L. Johnsson, "A formal derivation of array implementation of FFT algorithms," in *Proceedings of the USC Workshop on VLSI and Modern Signal Processing*, pp. 53–63, USC, September 1982. Also available as California Institute of Technology Computer Science report 5043:TM:82.

7. Cohen, Danny, and L. Johnsson, "The impact of VLSI on signal processing," in *Proceedings of the USC Workshop on VLSI and Modern Signal Processing*, pp. 153–156, USC, September 1982.

8. Cohen, Danny, and L. Johnsson, "Algebraic description of array implementation of FFT algorithms," in *Proceedings of the 20th Annual Allerton Conference on Communication, Control, and Computing*, pp. 126–134, University of Illinois at Urbana–Champaign, October 1982.

9. Cohen, Danny, and J. B. Postel, "Gateways bridges and tunnels in computer mail," in *Local Networks: Strategy and Systems, Proceedings of Localnet '83*, pp. 109–123, London Online, Inc., London, March 1983. Also appears in *Local Networks: Distributed Office and Factory Systems, Proceedings of Localnet '83*, pp. 385–400, New York, September 1983.

10. Cuykendall, R., A. Domic, W. H. Joyner, S. C. Johnson, S. Kelem, D. McBride, D. J. Mostow, J. E. Savage, and G. Saucier, "Design synthesis and measurement," in *Workshop Report: VLSI and Software Engineering Workshop*, pp. 6–9, IEEE Computer Society Press, Port Chester, New York, October 1982.

11. Kaczmarek, T., W. Mark, and D. Wilczynski, "The CUE project," in *Proceedings of SoftFair*, June 1983.

12. Lam, M., and D. J. Mostow, "A transformational model of VLSI systolic design," in *IFIP Sixth International Symposium on Computer Hardware Description Languages and Their Applications*, IFIP, Carnegie–Mellon University, May 1983.

13. Mann, W. C., "Anatomy of a Systemic Choice," in *Proceedings of the Ninth International Conference on Computational Linguistics*, COLING, Prague, July 1982.

14. Mann, W. C., "Multiparagraph text generation," in W. R. Swartout (ed.), *Workshop on Automated Explanation Production*, ACM, 1983.

15. Mann, W. C., "A tutorial on text generation," in *Conference Proceedings*, Association for Computational Linguistics, Santa Monica, California, January 1983. Oral presentation.

16. Mark, W., "Natural-language help in the Consul system," in H. L. Morgan (ed.), *AFIPS Conference Proceedings*, pp. 475–479, National Computer Conference, June 1982.

17. Mostow, D. J., "Operationalizing advice: A problem-solving model," in *Proceedings of the International Machine Learning Workshop*, University of Illinois, June 1983.

18. Mostow, D. J., and R. M. Balzer, "A program-transformation approach to VLSI design," in *Proceedings of the 1982 Workshop on Software Engineering and VLSI*, IEEE, 1983.

19. Mostow, D. J., "A decision-based framework for understanding hardware compilers," in *Proceedings of the 1982 Workshop on Software Engineering and VLSI*, IEEE, 1983.

20. Postel, J. B., C. Sunshine, and Danny Cohen, "Recent developments in the DARPA Internet program," in *Pathways to the Information Society, Proceedings of the Sixth International Conference on Computer Communication*, pp. 975–980, ICCC, London, September 1982.

21. Sunshine, C., (ed.), *Proceedings of the Second International Workshop on Protocol Specification Testing and Verification*, North-Holland, 1982.

22. Swartout, W. R., "Gist English generator," in *Proceedings of the National Conference on Artificial Intelligence*, pp. 404–407, AAAI, August 1982.

23. Swartout, W. R., "The Gist Behavior Explainer," in *Proceedings of the National Conference on Artificial Intelligence*, AAAI, 1983. Also available as USC/Information Sciences, RS-83-3, July 1983.

24. Weischedel, R. M., and N. K. Sondheimer, "An improved heuristic for ellipsis processing," in *Proceedings of the 20th Annual Meeting of the Association for Computational Linguistics*, pp. 85–88, Association for Computational Linguistics, Toronto, Ontario, June 1982.

Refereed Technical Reports

1. Hollingworth, D., *C2 Graphics Editor User's Manual*, USC/Information Sciences Institute, TM-83-24, 1983.

2. LaCoss, J., *Large Screen Terminal Project*, USC/Information Sciences Institute, Hardware Development Lab Report, August 1982.

3. LaCoss, J., *KW11-XX Unibus Clock Module*, USC/Information Sciences Institute, Hardware Development Lab Report, August 1982.

4. LaCoss, J., and R. Parker, *Faxie Interface Unit*, USC/Information Sciences Institute, Hardware Development Lab Report, January 1983.

5. LaCoss, J., *Standard Power Supply*, USC/Information Sciences Institute, Hardware Development Lab Report, January 1983.

6. Mann, W. C., *The Anatomy of a Systemic Choice*, USC/Information Sciences Institute, RR-82-104, October 1982. To appear in *Discourse Processes*.

7. Mann, W. C., and C. M. I. M. Matthiessen, *Two Discourse Generators* and *A Grammar and a Lexicon for a Text-Production System*, USC/Information Sciences Institute, RR-82-102, September 1982.

8. Mann, W. C., *An Overview of the Penman Text Generation System*, USC/Information Sciences Institute, RR-83-114, 1983.

9. Mann, W. C., and C. M. I. M. Matthiessen, *Nigel: A Systemic Grammar for Text Generation*, USC/Information Sciences Institute, RR-83-105, February 1983. The three papers in this report will also appear in a forthcoming volume of the *Advances in Discourse Processes* series, R. Freedle (ed.): *Systemic Perspectives on Discourse: Selected Theoretical Papers from the Ninth International Systemic Workshop*, to be published by Ablex.

10. Mann, W. C., *An Overview of the Nigel Text Generation Grammar*, USC/Information Sciences Institute, RR-83-113, April 1983.

11. Merritt, I. H., and R. Parker, *Switched Telephone Network Interface Card*, USC/Information Sciences Institute, Hardware Development Lab Report, March 1983.

12. Merritt, I. H., *Providing Telephone Line Access to a Packet Voice Network*, USC/Information Sciences Institute, RR-83-107, February 1983.

13. Mockapetris, P., *Communication Environments for Local Networks*, USC/Information Sciences Institute, RR-82-103, December 1982.

14. Parker, R., *VLSI Canary Tester*, USC/Information Sciences Institute, Hardware Development Lab Report, July 1982.

15. Shiffman, R., *RS-170 Composite Sync Generator*, Power Supply, Hardware Development Lab Report, July 1982.

16. Sunshine, C., and D. Smallberg, *Automated Protocol Verification*, USC/Information Sciences Institute, RR-83-110, October 1982.

17. Wile, D. S., *Formal Developments: Formal Explanations of Implementations*, USC/Information Sciences Institute, RR-82-99, August 1982.

Informal Project Notes

1. Clements, R., "Who Talks ICMP, Too? - Survey of 18 February 1983," USC/Information Sciences Institute, RFC 844, February 1983.

2. Postel, J. B., "Simple Mail Transport Protocol," USC/Information Sciences Institute, RFC 821, August 1982.

3. Postel, J. B., "Request for Comments on Requests for Comments," USC/Information Sciences Institute, RFC 825, November 1982.

4. Postel, J. B., "The Remote Telnet User Telnet Service," USC/Information Sciences Institute, RFC 818, November 1982.

5. Postel, J. B., and J. Vernon, "Requests for Comments Summary - Notes 600-699," USC/Information Sciences Institute, RFC 699, November 1982.

6. Postel, J. B., and J. Vernon, "Requests for Comments Summary - Notes 700-799," USC/Information Sciences Institute, RFC 800, November 1982.

7. Postel, J. B., "Assigned Numbers," USC/Information Sciences Institute, RFC 820, January 1983.

8. Postel, J. B., "Official Protocols," USC/Information Sciences Institute, RFC 840, April 1983.

9. Postel, J. B., and J. Reynolds, "Telnet Protocol Specification," USC/Information Sciences Institute, RFC 854, May 1983.

10. Postel, J. B., and J. Reynolds, "Telnet Option Specifications," USC/Information Sciences Institute, RFC 855, May 1983.

11. Postel, J. B., and J. Reynolds, "Telnet Binary Transmission," USC/Information Sciences Institute, RFC 856, May 1983.

12. Postel, J. B., and J. Reynolds, "Telnet Echo Option," USC/Information Sciences Institute, RFC 857, May 1983.

13. Postel, J. B., and J. Reynolds, "Telnet Suppress Go Ahead Option," USC/Information Sciences Institute, RFC 858, May 1983.

14. Postel, J. B., and J. Reynolds, "Telnet Status Option," USC/Information Sciences Institute, RFC 859, May 1983.

15. Postel, J. B., and J. Reynolds, "Telnet Timing Mark Option," USC/Information Sciences Institute, RFC 860, May 1983.

16. Postel, J. B., and J. Reynolds, "Telnet Extended Options - List Option," USC/Information Sciences Institute, RFC 861, May 1983.

17. Postel, J. B., "Echo Protocol," USC/Information Sciences Institute, RFC 862, May 1983.

18. Postel, J. B., "Discard Protocol," USC/Information Sciences Institute, RFC 863, May 1983.

19. Postel, J. B., "Character Generator Protocol," USC/Information Sciences Institute, RFC 864, May 1983.

20. Postel, J. B., "Quote of the Day Protocol," USC/Information Sciences Institute, RFC 865, May 1983.

21. Postel, J. B., "Active Users," USC/Information Sciences Institute, RFC 866, May 1983.

22. Postel, J. B., "Daytime Protocol," USC/Information Sciences Institute, RFC 867, May 1983.

23. Postel, J. B., and K. Harrenstien, "Time Protocol," USC/Information Sciences Institute, RFC 868, May 1983.

24. Smallberg, D., "Who Talks TCP? - Survey of 7-Dec-82," USC/Information Sciences Institute, RFC 832, December 1982.

25. Smallberg, D., "Who Talks TCP? - Survey of 14-Dec-82," USC/Information Sciences Institute, RFC 833, December 1982.

26. Smallberg, D., "Who Talks TCP? - Survey of 22-Dec-82," USC/Information Sciences Institute, RFC 834, December 1982.

27. Smallberg, D., "Who Talks TCP? - Survey of 28-Dec-82," USC/Information Sciences Institute, RFC 835, December 1982.

28. Smallberg, D., "Who Talks TCP? - Survey of 4-Jan-83," USC/Information Sciences Institute, RFC 836, January 1983.

29. Smallberg, D., "Who Talks TCP? - Survey of 11-Jan-83," USC/Information Sciences Institute, RFC 837, January 1983.

30. Smallberg, D., "Who Talks TCP? - Survey of 18-Jan-83," USC/Information Sciences Institute, RFC 838, January 1983.

31. Smallberg, D., "Who Talks TCP? - Survey of 25-Jan-83," USC/Information Sciences Institute, RFC 839, January 1983.

32. Smallberg, D., "Who Talks TCP? - Survey of 1-Feb-83," USC/Information Sciences Institute, RFC 842, February 1983.

33. Smallberg, D., "Who Talks TCP? - Survey of 8-Feb-83," USC/Information Sciences Institute, RFC 843, February 1983.

34. Smallberg, D., "Who Talks TCP? - Survey of 15-Feb-83," USC/Information Sciences Institute, RFC 845, February 1983.

35. Smallberg, D., "Who Talks TCP? - Survey of 22-Feb-83," USC/Information Sciences Institute, RFC 846, February 1983.

36. Smallberg, D., "Who Provides the Little TCP Services?," USC/Information Sciences Institute, RFC 848, March 1983.

37. Su, Z., and J. B. Postel, "The Domain Naming Convention for Internet User Applications," Network Information Center, SRI International, RFC 819, August 1982.

38. Sunshine, C., "Protocol Specification and Verification Work at USC/ISI: Summary Report," USC/Information Sciences Institute, IEN 211, August 1982.

39. Westine, A., "Summary of Smallberg Surveys - February 1983," USC/Information Sciences Institute, RFC 847, February 1983.

RESEARCH AND ADMINISTRATIVE SUPPORT

Institute Administration:
Robert Blechen
Monica Boseman
Patti Craig
Kathleen Fry
Gary Lum
Karen Luna
Gina Maschmeier
Lani Upton
Rolanda Valentin
Steve Wagner

Graphic Design:
Curtis Nishiyama

Librarian:
Alicia Drake

Publications:
Jim Melancon
Sheila Coyazo

Secretaries to Directors:
Barbara Brockschmidt
Joyce K. Reynolds

Computing Facilities Training and Information
Chloe Holg
Lisa Moses

Development Laboratory
Robert Parker
Shorty Garza
Bob Hines
Jeff LaCoss
Lee Magnone
Rick Shiffman
Jerry Wills

INFORMATION
SCIENCES
INSTITUTE
4676 Admiralty Way/Marina del Rey/California 90292-6695