

AD-A022 997

KNOWLEDGE WORKSHOP DEVELOPMENT

Douglas C. Engelbart

Stanford Research Institute

Prepared for:

Rome Air Development Center

30 January 1976

DISTRIBUTED BY:

NTIS

National Technical Information Service
U. S. DEPARTMENT OF COMMERCE

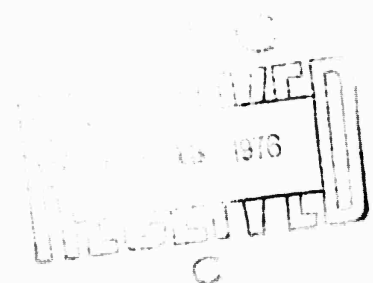
107040

AD A 022997

Knowledge Workshop Development

Augmentation Research Center

30 JANUARY 1976

**DISTRIBUTION STATEMENT A**

Approved for public release;
Distribution Unlimited



STANFORD RESEARCH INSTITUTE
Menlo Park, California 94025 • U.S.A

REPRODUCED BY
**NATIONAL TECHNICAL
INFORMATION SERVICE**
U. S. DEPARTMENT OF COMMERCE
SPRINGFIELD, VA. 22161

186

RADC-IR-75-304
Final Report
30 June 1974
SRI Project 1868

SRI-ARC 30 JAN 76 5 34PM 22133

KNOWLEDGE WORKSHOP DEVELOPMENT

Augmentation Research Center

Stanford Research Institute
Menlo Park, Ca. 94025

Sponsored by
Defense Advanced Research Projects Agency
ARPA ORDER NO. 2853

ACQUISITION BY	
RTIS	☒
DEC	☐
UNCLASSIFIED	☐
DISTRIBUTION	
BY	
DATE	
A	

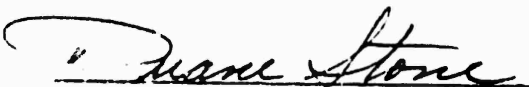
*Approved for public release;
distribution unlimited.*

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U. S. Government.

Rome Air Development Center
Air Force Systems Command
Griffiss Air Force Base, New York

FOREWORD

This technical report has been reviewed and is approved.


Contract Engineer

KNOWLEDGE WORKSHOP DEVELOPMENT

Contractor: Stanford Research Institute

Contract Number: F30602-72-C-0313

Effective Date of Contract: 16 August 1973

Expiration Date of Contract and Amendments: 30 June 1974

Amount of Contract: \$2,270,000

Program Code Number: F07619

SRI Project Number: 1868

Principal Investigator: Douglas C. Engelbart
Phone: (415) 326-6200, ext. 2220

Project Engineer: Duane L. Stone
Phone: (315) 330-3857

*Approved for public release;
distribution unlimited.*

This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense and was monitored by D. L. Stone, RADC (ISIM), GAFB, NY 13440 under Contract F30602-72-C-0313.

ABSTRACT

The oNLine System, Version 8 (NLS-8) developed at the Augmentation Research Center (ARC) was brought to prototype operation. Improvement in fundamental design continued, but new attention was given to applications and technology transfers. Strategies included an effort to involve more and more users outside ARC, establishment of experts in NLS within user organizations, a training program, an online query system to inform users about NLS, revision of the command language, and operation of the Network Information Center. New developments include a simple calculator subsystem, multi-host journal system, a variety of options to tailor the system to different working conditions, file access controls, and a control meta language to make user interface writing easier and more flexible. ARC developed a microprocessor-device, the Lineprocessor, to enhance inexpensive displays for use with two-dimensional display NLS and reduce communication cost. ARC made NLS available on a subscription basis through an information utility with its own computer.

CREDITS

The work from 10 May 1972 to 30 June 1974 involved the following ARC staff (some of whom have since left):

Andrews, Don I.
 Auerbach, Marilyn F.
 Bair, James H.
 Bass, Walter L.
 Beach, Mark
 Beck, Jeanne
 Bondurant, Rodney A.
 Byrd, Kaye
 Cooke, Judith
 Dornbush, Charles F.
 Duvall, William S.
 Engelbart, Douglas C.
 Evans, David
 Feinler, Jake J.
 Ferguson, William R.
 Glenn, Joy
 Guilbault, Carol
 Hardeman, Beauregard A.
 Hardy, Martin E.
 Hopper, J. David
 Irby, Charles H.
 Jernigan, Mil E.
 Johnson, Sandy
 Kaye, Diane S.
 Keeney, Marcia L.
 Kelley, Kirk E.
 Kudlick, Michael D.
 Lane, Linda L.
 Leavitt, Jeanne M.
 Lee, Susan R.
 Lehtman, Harvey G.
 Lieberman, Robert N.
 Limuti, Donald
 Lister, Priscilla M.
 Martin, Karolyn
 Maynard, David
 Meyer, N. Dean
 Michael, Elizabeth K.
 North, Jeanne B.
 Norton, James C.
 Page, Cynthia
 Parsley, Bruce L.
 Paxton, William H.
 Peters, Jeffrey C.
 Prather, Ralph
 Rathliff, Jake
 Rech, Paul
 Row, Barbara E.
 Vallee, Jacques F.
 Van De Riet, Edwin K.
 van Noughys, Dirk H.
 Victor, Kenneth E.
 Wallace, Donald C.
 Watson, Richard W.
 White, James E.

TABLE OF CONTENTS

Section	Branch
ABSTRACT	1
CREDITS	2
TABLE OF CONTENTS	3
INTRODUCTION	4
 CHAPTER I APPLICATION EXPERIENCE	
Aspects of ARC's Technology Transfer Strategy	5
User Training and Development	6
Experience with an Online Feedback Mechanism	7
 CHAPTER II USER INTERFACE	
Issues in the Design of the NLS User Interface	8
A Command Meta Language for NLS	9
First Studies of NLS Command Use and Timing	10
 CHAPTER III NLS SUBSYSTEMS	
The Calculator	11
The Output Processor and Computer Output to Microfilm	12
Recorded Dialog:	13
User Program System and Library	14
Query/Help Software and Data Bases	15
 CHAPTER IV WORKSHOP FOUNDATION	
The Group Allocation System	16
NLS File System	17
Software Engineering	18
TENEX Development	19
System Measurement Tools	20
APPENDIX, HIGHLIGHTS OF THE PREVIOUS REPORT	21

Preceding page blank

INTRODUCTION

TIME COVERED

This report covers Contract F30602-72-0313, which extended from March 1972 through June 1974.

SUMMARY OF WORK UNDER THIS CONTRACT

In this period the central development at ARC, the Online System (NLS), was brought to prototype operation with outside groups. Passing this milestone led us to undertake clarification of our mid-range goals, to make changes in the organization of ARC, and to give more energetic attention to a wide range of users through development of technology transfer, of system features, and of services. We here report on ARC's goals in terms of the "Augmented Knowledge Workshop" -- a computer-based set of tools for people who need to manipulate knowledge in their work. ARC has been organized into a development branch, whose work is of primary concern in this report, and an applications branch, dedicated to offering NLS as an information utility; each is under an assistant director.

The International Conference on Computers in Communications took place in Washington in October of 1972. It was an important event for most of the research organizations associated with the ARPA Network; it was particularly important to ARC. The Network Information Center prepared informative directories of ARPANET participants, and published scenarios of many systems demonstrated at the conference. Half a dozen ARC staff members spent full or substantial part time preparing for NIC services at the conference, and for demonstrating ARC functions through the Network, and took part in a variety of other support functions. Twelve members of the staff were in Washington for the duration of the conference.

Development of NLS

NLS User Interface

We have made the NLS user interface simpler, more flexible, and easier to use. We completed design and implementation of a Command Meta Language and command interpreter system that allows creating commands in terms of what they do rather than in our programmers' language (high-level language command specification). The CML system compiles the high-level terms used to describe commands into a tree of instructions to drive the existing NLS command interpreter, centralizing both command parsing and feedback to the user.

This approach allows experiments with different command language structures and feedback, simplifies building subsystems, and allows users to tailor command languages for themselves.

It also allows NLS "frontend" functions to move to a minicomputer. During this contract period, before the move to the minicomputer took place, the new architecture resulted in more compact source code and more efficient running.

Preceding page blank

4 Introduction

Other Changes

4b3a2

With the creation of the CML and our two years of experience with ARPANET users, we redesigned the command language to make it more consistent, and added features oriented toward novices.

NLS functions were reorganized into cleanly interconnecting subsystems. New subsystems include an arithmetic calculator integrated with NLS text files, the Modify subsystem, which contains automatic editing commands, and the Publish subsystem, which creates references, tables of contents, and the like.

We added a User Profile where a user can specify defaults such as the amount of feedback she gets, function of control characters, size of printout, type of recognition, and so on.

We added help commands that provide either a quick list of alternatives, complete command syntax, or access to complete, queriable documentation at a point related to what the user was doing when she asked. Cues to what the user was doing are derived from the CML.

User programs in the L10 programming language became increasingly important as the world of NLS applications widened, and programs supported by ARC were integrated into documentation and organized into a directory.

A restricted NLS Macro facility based on the command language, but lacking in loops, was implemented.

Provisions have been made to restrict access of NLS files to a list of idsents selected by the file owner.

Dialog Support

4b3b

We integrated the NLS Journal into the ARPA Network Mail System both for input and output; we have taken a leading role in creating a Network Mail Protocol.

4b3b1

We designed a Distributed Journal System and associated network protocols that allow various Journal functions such as distributing, recording, cataloging, storage and retrieval to exist and cooperate on scattered hosts.

4b3b2

We implemented an initial system in which two Network-based PDP-10's cooperate in supporting a common Journal system.

4b3b3

Privacy provisions were added to the Journal. A user may restrict access to a list of idsents she supplies. Private items are not cataloged.

4b3b4

Display Concepts and Terminals

4b3c

Display NLS was made more available to users working through the ARPANET or otherwise working remotely. In 1972 it became available through Imlac terminals; in 1973 we developed an inexpensive microcomputer based box, the Lineprocessor.

4b3c1

The Lineprocessor and associated software allow cheap, mass produced alpha-numeric terminals to display NLS files in the optimum, two-dimensional manner integrated with the Mouse and Keyset. The Lineprocessor does not require modification of the terminal.

4 Introduction

This work included extension of the NLS virtual terminal concept and development of associated communication protocols for the ARPANET. The results were useful in the development of the Network Graphics Protocol.

The Lineprocessors are being produced commercially for about \$2000 per unit and were just coming into use on the Network at the close of the contract period.

Operating System

The backup file archival and dump system, BSYS, developed at ARC, was released to the TENEX community.

A group allocation scheme to control logins to the system was built, put into operation, and released to the TENEX community. It split the users into groups and limits the number that may log in from each group. Allocation may vary during the day. Provision is made for brief "Express" logins over normal allocation.

Changes in TENEX necessary to support the Typewriter and Display versions of NLS became part of the standard BBN release of TENEX to allow future support of NLS on any standard TENEX.

We have found it advantageous to make several changes in our own TENEX, notably in the scheduler, while still remaining in harmony with BBN's standard TENEX releases.

We have built a system, Superwatch, to collect information on the consumption by various procedures and by users of CPU and clock time.

Network Information Center

During the contract period, the Network Information Center was the main source of information about the personnel, computing facilities, and organizations associated with the ARPANET, and of a large volume of related data. It was also an innovative development in providing information to a community of computer users, in online, offline, and mixed form. Service included support and cataloging of online dialog (through the Journal), an online database and query language, dissemination in hard copy of a Resource Notebook, an ARPANET directory, and Network protocols, frequent tours for visitors, and response to questions from the computer public.

At the end of the report period, the operations of the Network Information Center were curtailed from the experimental array of NLS-based information exchange services to maintenance of directories of persons and resources for the ARPA network; a detailed account and evaluation of Network Information services is to be published as a separate technical report [1].

THE ORGANIZATION OF THIS REPORT

Since 1970, the central funding of ARC's work has been a series of ARPA contracts. The resulting series of reports (7101,) (5139,) and (13041,) [2] [3] [4] outlined the evolution of ARC and the development of NLS in those years. By the middle of this report period, however, a prototype Knowledge Workshop existed, and much of ARC's thinking, particularly planning, turned toward defining new goals and opening applications in different directions. The support of

4 Introduction

ARC in the fiscal year beginning in July 1974 is more widely spread than ever before, a trend that we expect to see continue.

One result of the evolution of project emphasis is reflected in the organization of this report. The work during this contract period is reported under four headings:

Chapter I: Application Experience

Chapter II: User Interface

Chapter III: NLS Subsystems

Chapter IV: Workshop Foundation

The detailed descriptions under these four headings are reported in a series of what amount to individual papers. Inevitably this approach, while preserving the work of individual researchers, leads to a certain amount of redundancy. We apologize; however, this format seemed appropriate for work that reflects a core of accomplishment, but is sufficiently diversified that write-ups aimed at specialized audiences are appropriate.

To take advantage of the automatic reference search of our online system, bibliographic citations in this report look a little unusual. They will appear in two forms:

A string of numbers and letters in parentheses or angle brackets [e.g., <9a1>] cites some other part of this report as identified by the statement numbers printed to the right of the page. Online, a reader may cite such an address and move automatically to the appropriate part of the report.

A number in square brackets (e.g., [2]) cites a reference that is listed at the end of that particular section in which bibliographic information about these documents is supplied in the usual way. Each reference in turn cites the statement where the reference has originally been cited. The four or five digit number at the end of the reference citation itself is the ARC catalog number. All of the documents cited in this report are either online or archived, and an online reader may move to that file automatically.

A glossary of NLS-8 terminology and associated concepts has been published [5].

REFERENCES

- [1] (4b3e2) Michael D. Kudlick. Network Information Center. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. June 1975. (25088,)
- [2] (4c1) (4c4b) John B. Postel (UCLA-NMC). Official Initial Connection Protocol (Document No. 2). Network Information Center. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 11-JUN-71. (7101,)
- [3] (4c1) Douglas C. Englebart, and Staff of ARC. Computer-Augmented Management-System Research and Development of Augmentation Facility. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. APR-70. (5139,)

4 Introduction

- [4] (4c1) SRI-ARC. Online Team Environment / Network Information Center and Computer Augmented Team Interaction. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 6-MAR-73. (13041.) 404
- [5] (4c5) SRI-ARC. NLS-8 Glossary. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. June 1975. (22132,) 405

Chapter I: APPLICATION EXPERIENCE

Aspects of ARC's Technology Transfer Strategy (by Richard W. Watson, Douglas C. Engelbart, and James C. Norton)

INTRODUCTION

By 1972, following the connection of the ARC computer system to the ARPANET and the establishment at ARC of the Network Information Center (NIC), we began to actively plan for and carry out an explicit technology transfer strategy [1]. Previous experience had indicated traditional approaches to technology transfer--publishing papers and reports, giving demonstrations at conferences and at SRI, making movies, and giving slide shows and talks--while useful, were not enough to achieve technology transfer at the rate desired. Additional mechanisms were needed, particularly, hands-on experience by target groups. This chapter outlines some of the additional mechanisms being used and considerations for their selection.

Discussion

At the heart of our views on technology transfer is the belief, based on experience, that the type of information system we are developing can only be developed and evolve in an environment with real users doing their everyday work on the system. We at ARC had been the prime users of the system over the first decade of its development, but in the last three years have begun to seriously enlist outside users from a variety of organizations. The importance of obtaining views and feedback from the users with a variety of needs from many organizational environments is vital to the ongoing healthy evolution of a flexible and general-purpose knowledge workshop. Based on this premise, we have taken four steps:

- 1) We organized our internal activities during this contract period into three areas that we call:
 - a) Analysis
 - b) Development
 - c) Applications

The functioning of these three parts as a harmonious whole constitutes our research process. Development creates new user features, system organizations, and usage methodology based on experience and anticipated needs. Applications provides computer and other services, such as training to real users, both internally within the project and to outside groups. Analysis studies, at many levels, the ongoing system evolutionary process.

- 2) We have set up an ARPANET-connected facility managed by Tymshare at their Cupertino, California, computer center to serve as a reliable utility for delivery of workshop computer services developed at ARC and elsewhere. The present PDP-10 system is called Office-1 and is accessible through the ARPANET and directly through low-speed or high-speed phone lines. As part of the delivery system we have also developed a low-cost unit called a Lineprocessor (now commercially available) to support the display version of NLS from low-cost commercially available alphanumeric CRTs [2].

I Application Experience

5 Technology Transfer

The ability to offer reliable computer service is crucial to the Development-Application-Analysis strategy. Staff and facilities with the know-how and motivation to create such a facility are not easily maintained by a highly Development- and Application-oriented organization such as ARC. Therefore, an important decision was made in 1972 to subcontract computer facility management to a corporation like Tymshare that has the staff and physical facilities for providing the needed services. Tymshare is responsible for hardware and operating system reliability. ARC is responsible for all services at higher levels.

This has been a valuable and trend-setting move within the ARPA R&D computer community.

3) We are asking each subscribing organization to provide what we are calling a "workshop architect," whose prime loyalty is to the using organization (preferably a person from the using organization, although we will provide a person for that role if necessary) to plan and conduct a staged evolution of the technology and training appropriate to his organization.

The importance to successful technology transfer of having a person within the target organization who is familiar with his organization's needs and the outside technology has been clearly demonstrated in the works of Allen [3][4][5][6][7]. Allen has called such a person a gatekeeper, and has shown that most technology transfer occurs through such people, usually operating on an informal basis. We are trying to formalize and make explicit this role.

On the ARC side we have created the roles of architect liaison -- whose function is to help define and shape the subscribing organization's basic level of service; and applications liaison -- who assists in developing those specific applications suitable to each client. Both are to be generally knowledgeable about ARC technology and outside user needs. It is across these overlapping liaison and workshop architect roles that we hope to achieve effective transfer, while being supported by other technical, analysis, and training personnel as well.

4) The technology was originally developed on the assumption that it would be used as the everyday working environment of its users and that therefore the users would quickly be of the expert category. Experience has shown that a) it will probably be some time before this is the case, and b) even where it becomes the case, there is a critical transfer phase. Therefore, we have begun during the past year to pay much more attention to levels of documentation, usage scenarios, help, novice language features, etc., to provide a spectrum of functions from new to experienced users.

Our experience indicates that conscious attention to technology transfer by an R&D group affects:

- A) Its organizational structure
- B) The types of skills and roles needed
- C) Its R&D strategy.

I Application Experience

5 Technology Transfer

LET US NOW LOOK AT THE FIVE TECHNOLOGY TRANSFER ISSUES THAT LEAD TO THESE STEPS:

1) Need for demand pull versus technology push.

We feel that successful transfer takes place only when a real need is met. Just to have a clever new toy is not sufficient for a technology to stick. It must meet a real need at a cost appropriate to the user's value in order for transfer to be realized.

This need to understand real needs in the outside world and to try to determine how well we are providing value leads to the creation of an Analysis function to study needs and analyze how well we are meeting them. We brought in an experienced operations research person with little interest in the technology as a thing in itself to provide this perspective.

Because of changes in funding levels and pressing needs for trained personnel within the Applications group, we have temporarily halted the Analysis function. Recruiting people with the appropriate interests, training, experience, and motivation for the important Analysis function is a difficult task. It is a highly interdisciplinary function and is not easily filled by the present orientations of academic computer science, operations research, or psychology departments.

Usage by real users with work and applications to do other than build the NLS system is providing us with the feedback and contact with real needs that we feel are necessary to help us operate more on the need-pull side of the technology pull-push spectrum.

2) One has to know where one is with respect to the two questions:

Is one trying to show something is feasible? OR

Is one trying to show something meets a need and should be continued?

We feel the former was accomplished and that we are in the latter area, thus requiring a shift in emphasis from technology-push to need-pull.

3) The ease of technology transfer is proportional to the risk and cost to the user in terms of total system, organization, work habit, and training he has to undergo to adopt the new technology. Technology transfer has been described as more of a battle than just a matter of communicating an idea. Our experience confirms this view.

To meet this issue we are asking user organizations not to try to adopt our technology on a broad scale, but to find a subgroup to try first, learn the advantages and problems, and then develop people trained in its use to take the next steps.

4) Transfer of our type of technology is most successful by transfer of people. Studies at MIT of developments done at MIT and their transfer to industry found that on the order of 90% of the successful transplants were achieved by students or faculty going to work for the organization, obtaining an understanding of the organization's problems, and then bringing in the technology he was familiar with. Industrial firms transfer many of their people periodically for just these reasons.

1 Application Experience

5 Technology Transfer

However, it is not easy to transfer people from SRI to outside groups, nor do we have enough people to do that. This problem, when coupled with the motivation of the gatekeeper concept, supports our establishment of the workshop architect role [3][4][5][6][7].

In the future when we have our experiment off the ground, we may try to transfer ARC people to user groups for six months to one year, and vice versa. For the past three months and for the coming months, we have stationed one ARC person in Washington, D.C., where a number of NLS user organizations are clustered, to provide an approximation of such a role. We have found this close contact useful and important to the transfer process.

We would like internally to move our people through the Development, Analysis, and Application areas to help them obtain several points of view, as our technology transfer effort matures.

5) To transfer a system such as ours, and even many of its ideas, requires much more than publishing papers and reports. One needs a gut feeling that only a demonstration or, better yet, hands-on experience can give. This has led us to encourage visitors to ARC and to set up the NLS utility to provide service to real users. One problem we have faced is the task of finding suitable low-cost, commercially available display terminals for NLS use. Thus, most outside users to date have had to use the typewriter version, which has quite different user characteristics and feel from the display version they see in use at ARC. To make the display version more widely available, we have developed a special microcomputer-based box for use with commercially available alphanumeric terminals that enables them to be used without modifications as true two-dimensional display NLS workstations [2].

CONCLUSIONS

Experience to date indicates that the elements of a technology transfer strategy have put us on the right track, although there is much yet to be learned about the process. It has shown us that technology transfer can be made an explicit, conscious process and that the efficiency and effectiveness of technology transfer can be improved as a result.

REFERENCES

- [1] (5a2) Douglas C. Engelbart, Richard W. Watson, James C. Norton. The Augmented Knowledge Workshop. In AFIPS Proceedings, Vol. 42, 1973 National Computer Conference, pp. 9-21, 1973. (14724,)
- [2] (5a6) (5b5) Don I. Andrews. Lineprocessor: A Device for Amplification of Display Terminal Capabilities for Text Manipulation. In Proceedings of the National Computer Conference, 1974, p. 257-265. (20184,)
- [3] (5a7a) (5b4a) Thomas Allen. Technology transfer to developing countries. The International Gatekeeper. In ASIS Proceedings, Vol. 7, The Information Conscious Society, 33rd Annual Meeting, 1970, p. 205-210. (13959,)

I Application Experience

5 Technology Transfer

- [4] (5a7a) (5b4a) Thomas Allen. Technology transfer to developing countries: The International Gatekeeper. Massachusetts Institute of Technology. Feb-71. (13859,) 5 d 4
- [5] (5a7a) (5b4a) Thomas Allen. Roles in Technical Communication Networks. Massachusetts Institute of Technology. 1970. (13977,) 5 d 5
- [6] (5a7a) (5b4a) Thomas Allen. Performance of Information Channels in the Transfer of Technology. Massachusetts Institute of Technology. 1966. (15538,) 5 d 6
- [7] (5a7a) (5b4a) Thomas Allen. Information Flow in Research and Development Laboratories. Massachusetts Institute of Technology. Mar-69. (15539,) 5 d 7

User Training and Development (by Dirk H van Nouhuys and James H Bair)

BACKGROUND

At the beginning of this report period, efforts to train both internal and external users, for the purpose of transferring knowledge of NLS, consisted of training courses offered through the Network Information Center (Journal, 13041, 5g10), distribution of a Reference Manual, Journal dialog [1] ARC and external users, workbooks, and informal but frequent question answering by members of the ARC staff, particularly through online links.

EXPERIMENTAL PERIOD

The first year of this contract was a time of experiment in media and techniques as shown in Table 1 and also in goals and organization. We had to construct answers to such questions as:

Was it our function to train users local to ARC, NIC users, or all comers?

Should we train users to the limit of their ability, enough to get them started, enough to perform some specific task, or should it vary from user group to user group?

Should training materials and courses be aimed at, for example, bright computer professionals, managers, secretaries, or should we, at considerable effort, develop a suite of training modes to fit various groups?

The system that evolved from our attempt to answer these questions is described below. It does not seem worthwhile to summarize the internal debate that took place. It is however important to reflect that a group such as this faces these kinds of questions and debate along with a shortage of criteria to guide conclusions. When we attempted to analyze results in order to choose our future course we found the analysis demanding, and that in many cases we had frequently gone ahead on the basis of our experience.

In the summer of 1973, our training was substantially reorganized, partly on the basis of experience, partly because of new expertise imported into the group, and because a substantial part of the training effort shifted to another contract. Table 1 summarizes our experience up to that time according to medium: with some comment on the enduring value of the methods.

LIST OF EARLY TRAINING METHODS AND MEDIA

Hardcopy training aids

TNLS, Dex, L-10, and Journal "User Guides".

Use: Internal reference, distributed widely through the NIC for reference at NIC stations, as homework before courses, and to give an impression of NLS to interested parties.

Current State: Kept up-to-date except for NLS-8, not distributed or used in courses, available on request.

Preceding page blank

I Application Experience

6 Training and Development

Comment: Really reference documents; exhaustive, hard to read and maintain except for restricted subsystems.

6c1a3

References: [2][3][4][5][6][7]

6c1a4

A set of flip charts of NLS commands and allied subjects for the TNLS course.

6c1b

Use: As prompts for instructor and visual aids for students.

6c1b1

Current State: No longer used due to their unwieldy bulk in traveling and the difficulty of displaying them at client sites. Handouts, viewgraphs, and blackboards are more practical.

6c1b2

Comment: 8.5 by 11-inch color copies of the most important charts were distributed; now out of date and not reprinted.

6c1b3

A series of workbooks (like Sullivan Readers).

6c1c

Use: A person learning alone at an online terminal has exact, keystroke-by-keystroke guidance showing operation of most NLS features.

6c1c1

Current State: Available on and offline for NLS-7, little used.

6c1c2

Folklore which was reprinted and sent out to supplement the User Guide from time to time.

6c1d

Use: Updating manuals and providing information about procedures that were customary rather than command rules.

6c1d1

Current State: No longer used. NLS-7 does not change and HELP (See Section 12), provides information about NLS-8.

6c1d2

Comment: It was very difficult to get users at distant sites to shelve properly or use folklore. Traveling trainers can update a Site Notebook containing the latest documentation for the users' reference.

6c1d3

References: [9][10]

6c1d4

Journalized response to journalized questions.

6c1e

Use: Very extensive for answering questions from other network sites.

6c1e1

Current State: Largely replaced by user feedback systems described below (6d).

6c1e2

Comment: An important medium of exporting techniques of NLS use.

6c1e3

References: [11] for example.

6c1e4

A wallet-sized card of the viewspecs and mouse and keyset codes.

6c1f

Use: People keep it beside their terminals.

6c1f1

Current State: In use.

6c1f2

Comment: Visitors frequently take the cards as a physical embodiment of the special qualities of NLS.

6c1f3

References: [12]

6c1f4

I Application Experience

6 Training and Development

A command cue card that folds to shirt pocket size.

Use: For reference when working alone and in class; handed to visitors.

Current State: In use.

Comment: Two-color.

References: [13]

Documents that describe the envelope of philosophy and procedures around NLS.

Use: By people organizing use of NLS at other sites.

Current State: Available through Journal hard copy.

Comment: Little read to our knowledge.

References: For example, [19] is thoughtful; [1] is general, whereas [17] and [16] deal with daily details. [15] and [14] are more recent examples.

Online Training Aids

The TNLS HELP Command ["?"] at any point in a command lists the legal command terms at that point.

Use: A frequent and important method of learning for people who already know a little.

Current State:

In use in TNLS. Never implemented in old Display NLS. Expanded in NLS-8 in Display and TNLS; supplemented and coordinated with HELP system.

Comment: Also useful for more advanced users to remind them of a command term for a particular use.

The Userguides, workbooks, and procedural documents described above.

Use: In general the same as the hardcopy versions for people when the system is available.

Current State: The Userguides and workbooks are kept up to date.

Comment: Userguides read on the Utility.

References: [7][6][5][4][3][2][18][11][17][16][15][14]

Demonstration via linked terminals.

Use: Usually the result of an inexperienced user seeing the name of an experienced user of her acquaintance on the system and asking a question.

Current State: This practice continues at the utility although the feedback system partially replaces it.

Comment: An important medium of technology transfer: there is nothing quite like seeing it done.

A series of seminars in NLS (Continuing NLS Education) based on remote online demonstration with some linking and shared images with other sites [19].

I Application Experience

6 Training and Development

Use: Teaching new or out of the way features to people who already know basic NLS.

6c2d1

Current State: Not used.

6c2d2

Comment: Connection with other sites was difficult technically at the time; it might be easier now. Local users paid lip service to developing their skill, but in practice preferred to do work and stayed away from seminars.

6c2d3

References: [19]

6c2d4

Face-to-face Teaching

6c1

A series of courses at ARC in TNLS for local people and people from the network and for network people at other sites.

6c1a

Use: Conceived as the main medium of exporting knowledge of NLS. The interaction of people talking, framing and reframing questions and answers, helps a lot in communicating the novel concepts and language of NLS.

6c1a1

Current State: Continues for subscribers to the Utility.

6c1a2

Comment: Many people learned NLS this way, but only if they continued to use it after the course was over. It served as a showcase for NLS to others.

6c1a3

References: [20]

6c1a4

A series of training sessions for people in more or less clerical positions at ARC.

6c1b

Use: To increase the skills of people with little or no formal training whose work did not demand innovative use of the system.

6c1b1

Current State: Pressure on resources forced discontinuance of special courses. Clerical personnel now take the same courses that are offered to clients.

6c1b2

Comment: All clerical staff at ARC are trained in TNLS.

6c1b3

Local question answering.

6c1c

Use: An effective way of augmenting the teaching of NLS locally.

6c1c1

Current State: Very active due to the availability of knowledgeable users in the open, common terminal area.

6c1c2

Comment: Time-consuming for experienced users.

6c1c3

Teaching L-10/NLS, and TENEX programming.

6c1d

Use: To maintain the skills for development and maintenance of our system.

6c1d1

Current State: Aside from the limited introductory documentation, teaching how to program in our system consists entirely of tutoring by experienced programmers.

6c1d2

Comment: Perhaps the situation is tolerable as long as the group of programmers is neither too small nor too large.

6c1d3

I Application Experience

6 Training and Development

Video tapes

Use: Dissemination of general impressions of ARC work; teaching small parts of the system.

Current State: After a lapse in use, more ambitious production of overviews of NLS activity is going on under another contract.

Comment: One overview and one live sequence from a TNLS class were produced. Very time-consuming for staff. Technical quality fair to poor. The training tape is little viewed and now out of date.

References: [21][22]

Computer-aided instruction

Use: Might be used someday for online teaching of NLS.

Current State: BBN has a contract to develop a course in basic TNLS on their SCHOLAR system.

Comment: Very expensive -- impractical given NLS and the state-of-the-art in CAI.

References: [23][24]

Control files which permit you to record an NLS session and play it back in real time.

Use: We have always believed control files might be useful in training but have never used them.

Current State: Available but not presently used for training.

PRESENT USER TRAINING AND DEVELOPMENT STRATEGY

Background

The evolution of augmentation technology has included offering NLS as an experimental service. As a consequence, learning how to support the users of such a service has become a major goal of ARC. To that end, positions have been established to provide those non-computer and para-computer services that are seen as necessary to make such a utility service viable. These positions have responsibilities in the areas of instructional development, training, front end analysis, user interface, documentation and feedback coordination. They function in cooperation with other areas (such as operations management, programming development and debugging, and marketing) with the goal of facilitating the use of NLS in varied user environments for multiple applications.

Areas of Work

Training

The scheduling, custom tailoring of courses, and the collection of data on the training and implementation process are to be centrally managed. This will permit the systematic development of training methods during the evolution of the user population.

1 Application Experience

6 Training and Development

Instructional Development

6d2b

Instructional development includes:

6d2b1

1. The development of NLS Graduated Courses for a user oriented progression through NLS commands, syntax, and procedures.
2. Design recommendations on software development for NLS modules based on an analysis of user experiences from a psychological viewpoint.
3. Development of training packages and "application scenarios" to guide the accomplishment of particular tasks and applications.
4. Inputs to documentation concerning user needs, including document content, layout, and structure.

User Interface Analysis

6d2c

Review of the command language, error messages, and other aspects of the user interface from a psychological viewpoint in order to make recommendations to the Software Development team. The goal is to aid in rendering NLS as intuitive and straightforward as possible to non-programming users.

6d2c1

Feedback Coordination

6d2d

Operation of the mechanism to sort, route, and respond to direct user inquiries.

6d2d1

Analytical Reporting

6d2e

1. Training reports describing specific training processes and noting the result.
2. Writing reports and gathering research relevant to the foregoing and technology transfer in general; sharing and interfacing with other experts in this area.
3. Reporting the effect of any reaction to NLS by the user population.

6d2e1

6d2e2

6d2e3

User Development Handbook

6d2f

Maintain an offline record of courses, reports, feedback, etc., for reference.

6d2f1

User Profile Data Base

6d2g

Organizational and individual user profiles in a structured database available for reference.

6d2g1

Accomplishments

6d3

Instructional development

6d3a

Numerous courses have been given in NLS to various groups of individuals over the years. However, no formal course had evolved that could be utilized by a trainer. A formal course has the advantage of graduating the exposure to NLS in such a way to maximize the user's progression from a minimal capability to the highest level he wishes to attain. Those elements (concepts, commands, syntax, procedures) that are most appropriate to the situation, give a basic operating capability, and eliminate the need for alternatives, can be selected in view of past training experience, psychological considerations, and logical relationships within the system.

6d3a1

I Application Experience

6 Training and Development

The system tools permit the development of graduated formal courses by invoking content filters to filter out those elements not to be covered in a session, and representing the relationship of the elements in a hierarchy. Courses in TNLS and DNLS were developed during this period and refined as a result of use in varied environments:

6 d 3 a 7

1. The five-level graduated progression through TNLS
2. The filtered progression through DNLS

The course outlines ideally can be self-documenting by providing the exact command syntax to accomplish straightforward operations, such as editing and printing. Such a course was designed and tested in two user environments, and published for general use in training situations. It includes a command summary with those commands that were selected to represent the most basic level of NLS skill and usage, [25].

6 d 3 a 7

User Profile Data Base

6 d 3 b

The rapidly expanding user population has become large and complex enough to warrant an ongoing record of the individuals who have been exposed to NLS or trained. The record includes notation of the training delivered, the position in the organization, and the relationships in and among organizations.

6 d 3 c 1

The hierarchical structure of NLS is ideally suited to represent the relationships in the database, and to facilitate the frequent updating that is necessary.

6 d 3 d 1

The Feedback Mechanism

6 d 3 c

This was established [26] to ensure that the users of the experimental service have a mechanism whereby they can submit problems, comments, suggestions, etc. To that end, a directory ("Feedback") was implemented to serve as a depository for inquiries made via the system. The operation of the mechanism includes sorting the inquiries, consulting the appropriate expert, and responding within one or two working days via the same system channel used for the initial inquiry. Each item received is treated individually, and then sorted on the basis of the action taken.

6 d 3 c 1

An analysis of the stored inquiries and responses (collocated) is made periodically, including a frequency count of the number of inquiries relevant to a particular issue or problem. Future analysis should examine the input more thoroughly and sort the responses with the inquiries, to provide a database from which to draw conclusions about software development.

6 d 3 c 2

Operation of the mechanism has involved three general areas:

6 d 3 c 3

- 1) Instruction: Answers to inquiries that require pedagogical responses. When these are of general interest they are distributed to additional users as Training Memos.
- 2) Software repair: bug fixing by contacting the appropriate programming staff.
- 3) Hardware problems and acquisition: Providing consultation or repair to make equipment reliable and consistent (e.g., noise in telephone connections).

I Application Experience

6 Training and Development

Analytical and Training Reports

1. User Development Report: Training Tour 1 to 11 May 74
Location: [27]
2. User Development Report: Training Tour 11 to 20 Mar 74
Location: [28]
3. User Development Trip to Bell Canada & Preapplication Analysis
Location: [29]
4. User Development Trip to RADC, 19 Nov 73
Location: [30]
5. Bair/Norton Trip to Bell Canada and RADC, Dec. 73
Location: [31]

ACKNOWLEDGMENTS

Many members of the ARC staff participated in training, preparation for training, and preparing documents and other aids during the contract period. Foremost are Marilyn F. Auerbach, Harvey Lehtman, Kirk Kelley, Beauregard Hardeman, Michael D. Kudlick, Richard Watson, and Carol Guilbault.

REFERENCES

- [1] (6A1) (6b4a8d) (6b4b2d) SRI-ARC. Online Team Environment / Network Information Center and Computer Augmented Team Interaction. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 6-MAR-73. (13041.)
- [2] (6b4a1d) (6b4b2d) No Author. DNLS ENVIRONMENT. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 16-JUN-72. (10704.)
- [3] (6b4a1d) (6b4b2d) No Author. FILES. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 16-JUN-72. (10705.)
- [4] (6b4a1d) (6b4b2d) No Author. ADDRESSING IN DNLS - JUMPING AND LINKS. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 16-JUN-72. (10706.)
- [5] (6b4a1d) (6b4b2d) No Author. VIEW CONTROL OPERATIONS. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 16-JUN-72. (10708.)
- [6] (6b4a1d) (6b4b2d) No Author. DNLS/EXEC. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 16-JUN-72. (10713.)
- [7] (6b4a1d) (6b4b2d) No Author. DNLS Preliminary User Guide. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 19-JUN-72. (10703.)

I Application Experience

6 Training and Development

- [8] (6b4b2d) Susan R. Lee. Exercise File for Text Editing (Network Version). Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 26-JUN-73. (17352,) o*18
- [9] (6b4a4d) Marilyn F. Auerbach. Folklore. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 1-MAR-73. (14771,) o*18
- [10] (6b4a4d) Marilyn F. Auerbach. DOCUMENTATION CHANGES. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 5-MAR-73. (14890,) o*18
- [11] (6b4a5d) David H. Crocker. [Question about L10 Functions]. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 9-JUL-73. (17726,) o*18
- [12] (6b4a6d) Marilyn F. Auerbach. NWG/RFC 496 #1 A TNLS QUICK REFERENCE CARD IS AVAILABLE. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 5-APR-73. (15496,) o*18
- [13] (6b4a7d) Marilyn F. Auerbach. NWG/RFC 496 #1 A TNLS QUICK REFERENCE CARD IS AVAILABLE. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 5-APR-73. (15496,) o*18
- [14] (6b4a8d) (6b4b2d) N. Dean Meyer. Procedure for Sending Messages. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. MENLS;2,. (22893,) o*18
- [15] (6b4a8d) (6b4b2d) Jeanne M. Beck. Procedure for maintaining the Userguides directory. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 9-MAR-74. (22363,) o*18
- [16] (6b4a8d) (6b4b2d) Jeanne B. North. Some Procedures for People Support of an Online Information System. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 11-SEP-72. (13037,) o*18
- [17] (6b4a8d) (6b4b2d) No Author. Technical Support for RADC Use of Augmentation Technology. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 5-MAR-73. (14567,3) o*18
- [18] (6b4b2d) SRI-ARC. Online Team Environment / Network Information Center, and Computer Augmented Team Interaction. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 6-MAR-73. (13041,) o*18
- [19] (6b4a8d) (6b4b4d) (6b4b4) Douglas C. Engelbart, Richard W. Watson, James C. Horton. The Augmented Knowledge Workshop. In AFIPS Proceedings, Vol. 42, 1973 National Computer Conference, pp. 9-21, 1973. (14724,) o*18
- [20] (6b4c1d) Dirk H. van Nouhuys. NLS Continuing Education. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 12-JAN-73. (13783,) o*18

1 Application Experience

6 Training and Development

- [21] (6b4d4) Dirk H. van Nouhuys. Notice of NIC TNLS Course at SRI-ARC, 7-8 February 1973. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 10-JAN-73. (13755,) 6471
- [22] (6b4d4) James C. Norton, Richard W. Watson, Dirk H. van Nouhuys, Marilyn F. Auerbach, Harvey G. Lehtman. Augmentation Research Center and Network Information Center: Video Tape. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. Hard. (15365,) 6472
- [23] (6b4e4) Dirk H. van Nouhuys. Live at the ICCC/ A Video Tape. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. Hard. (15481,) 6474
- [24] (6b4e4) Dirk H. van Nouhuys. NLS-SCHOLAR: Current Status. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 3-AUG-73. (18201,) 6474
- [25] (6c3a3) Kirk E. Kelley. Status of SCHOLAR with respect to Office-1 and NLS-8. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 22-JAN-74. (21562,) 6474
- [26] (6c3c1) James H. Bair. The Basic TNL Course. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. COURSENSW.NLS;17. (22858,) 6476
- [27] (6c3d1) James H. Bair. The Utility Feedback Mechanism. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 6-APR-74. (22642,) 6477
- [28] (6c3d2) James H. Bair. User Development Report: Training Tour 1 - 11 May 74. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. TRIPMAY.NLS;7. (23133,) 6478
- [29] (6c3d3) James H. Bair. User Development Report: Training Tour 11 Mar -- 20 Mar 74. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 8-APR-74. (22656,) 6479
- [30] (6c3d4) James H. Bair. User Development Trip to Beil Canada & Pre-application Analysis. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 30-MAR-74. (22536,) 6480
- [31] (6c3d5) James H. Bair. User Development Trip to RADC. 19 Nov 73. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 30-MAR-74. (22535,) 6481

Experience with an Online Feedback Mechanism (by Susan R Lee and Kirk E Kelley)

INTRODUCTION

In order to better assess the new command language, Analysis has developed a procedure for collecting and handling feedback from ARC users. Because the user group is of an experimental nature, it has been possible to try several plans of collecting, organizing, and responding to the data.

Data were collected both online and offline and data submitted online were submitted via the Journal (recorded) and via Sndmsg (unrecorded). People were encouraged to, and in fact did, send most items via the Journal. The Journal proved to be the best method for collecting data because the data arrived in an easily manipulated state and items were easily referenced and retrieved when some action had been taken.

Several attempts were made to organize the data to enable different people to use the file for various purposes. Different sections were organized chronologically, categorically, and according to a set of priorities. Suggestions for future features were eventually organized categorically to have similar and related suggestions in close proximity. Other schemes fragmented the subject matter too much. Specific tasks assigned to a programmer were ordered by programmer, since this section was used mainly by each individual. Priorities seemed to be too difficult to assign and keep up to date in this circumstance.

Response to users warrants more exploration; the two methods tried were offline (orally) and online. Online responses were limited to notification when a bug had been fixed, answers to specific questions, and notification that a suggestion had been rejected. Online responses were most desirable because they did not disturb a person as an offline interruption did. Furthermore, the responses were collected and easily accessible by all users, enabling those who were curious to peruse the file and learn new things.

The method described below has proven to be both easy to use and helpful to people dealing with feedback.

DESCRIPTION OF MECHANISM

Users are encouraged to submit all comments online, using either Sendmail or Sndmsg. Reports of bugs, suggestions for new features, and attitudes, both positive and negative, are solicited. All such inputs are routed to a master file in the Feedback directory.

The information collected is organized into the following categories.

- 1) Unclassified Items (Sendmail automatically delivers items to this section.)
- 2) Bugs (reports of features that are not working as advertised).
- 3) Praise (features people like).
- 4) Future Needs and Possibilities (suggestions for new features and changes to old ones).

I Application Experience

7 Online Feedback

5) Rejected Suggestions (suggested changes that Development has rejected).

6) Assigned Tasks (bugs or other suggestions assigned to a specific system developer--organized by developer).

7) Implemented Items (fixed bugs or implemented suggestions).

Items move from one section to another by the following algorithm.

1) Unclassified items are placed in one of the succeeding categories.

2) The bugs section is reviewed several times a week by a systems developer who assigns each item to the appropriate person.

The bugs are then moved to corresponding sections under Assigned Tasks.

3) The Assigned Tasks section is also reviewed periodically for completed tasks, which are then moved to the Implemented Items section.

4) The Future Needs and Possibilities section is reviewed by Analysis approximately once a month and a report is submitted to Development listing issues to be resolved and subsequently either rejected or assigned to a person.

When action is taken on an item or when for some reason its status changes, the user who submitted the item is notified by means of an online message.

EXPERIENCE

The feedback mechanism has been used extensively and viewed positively, primarily because of the knowledge that a problem submitted to the file will be considered and some response will be forthcoming. Approximately 10 to 15 reports a week have been received since January 1974.

System developers have found a list of complaints and problems to be an aid in scheduling work and establishing priorities. The mechanism also seems to act as a buffer between developer and user, softening the negative aspects of user frustration while still assuring that action is taken.

The database consisting of user inputs has proven to be a valuable source of ideas for modifications to improve the command language and to improve the methods used for accomplishing work. It is structured categorically to be of use during the design of future command languages and subsets of command languages. Many problems encountered by a group of people learning a new command language have been recorded and categorized. These should prove helpful when the command language is introduced to other new users.

An online mechanism such as this seems to be most suitable for handling bugs and suggestions for future changes. In both cases the user usually has sufficient motivation to make the effort to submit the item. When a collection of attitudes and feelings is desired, especially positive ones, the motivation level is low and an online mechanism is probably not as effective as a method involving personal interviews or questionnaires.

Chapter II: USER INTERFACE

Issues in the Design of the NLS User Interface (by Richard W Watson)

INTRODUCTION

The user interface has two sides: the input side by which the user inputs information, indicating by various conventions and controls what he wishes accomplished; and the output side by which the machine provides feedback and other assistance to the user in command specification, and provides various forms of information portrayal. Man has many motor and other capabilities that could be the basis for input and command specifications; similarly he has his full range of senses that could be targets for system output.

To date, computer information systems make use of only a few motor and sensory capabilities in their man-machine dialog. An important area of research involves exploring the advantages to be gained and the techniques to be used to extend this range. There is interesting research going on in areas of speech, eye movement, brain wave control, hand written script, and video graphics that will undoubtedly be integrated into the truly multimedia systems to be built in the near future.

We call the user's collection of input-output equipment, and arrangement of work tables and work space, the workstation. At the present time, input centers around various types of keyboard devices: standard typewriter-type, function button, keyset (chord), and graphical pointing devices (mouse, electronic pen-tablet, light pen, joystick). The dominant output means are printers and displays of varying capabilities.

The present NLS user interface has been developed around this equipment, although many of the principles used in its design can be easily extended for use with other media [3]. The prime motivation for the use of the mouse for pointing and two keyboards (standard typewriter-like and keyset) as the input devices for the display version of NLS-7 (DNLS), are described in references [2][3]. NLS can also be used from typewriter terminals (TNLS). In this chapter, we concentrate on describing some of the motivations behind the design of the NLS command language and the forms of information portrayed to assist the user in command specifications. Forms of general NLS information portrayal are described in reference [1].

NLS is a prototype collection of tools in a growing workshop of tools and services to aid knowledge work [1][4], and we expect the number of tools and vocabulary that controls their use to grow. We further expect that the use of such a workshop will spread throughout those occupations involved with information in various forms and that there will be infrequent and casual users of such systems, along with many people who will spend large fractions of their day using such workshops. One goal is to match the speed of system responsiveness to the natural speed and flow of man's thought processes. It is from these basic expectations that our user interface work has developed. The sections below enumerate several assumptions and areas of

II User Interface

8 Design Issues

concern around which the NLS user interface has developed to date. A key point to mention is that we do not consider the NLS user interface a static, finished product. It will change, based on analysis of usage experience, and the technology and media available.

HIGH LEVEL ASSUMPTIONS UNDERLYING THE DESIGN OF THE NLS USER INTERFACE

First we describe a few high-level assumptions that affect the user interface design and then discuss some of the lower level issues and the specific techniques used to deal with them.

1) Coordinated Set of User Interface Principles

There will be a common command interaction discipline, over the many application areas in the workshop, that shapes user interface features, such as the language, control conventions, methods for obtaining help, and computer-aided training.

This commonality has two main implications. One, it means that while each domain within the core workshop area or within a specialized application system may have a vocabulary unique to its area, this vocabulary will be used within language and control structures common throughout the workshop system. A user will learn to use additional functions by increasing vocabulary, not by having to learn separate "foreign" languages. Two, when in trouble, he will invoke help or tutorial functions in a standard way.

2) Grades Of User Proficiency

A once-in-a-while user with a minimum of learning will want to be able to get at least a few straightforward things done. In fact, even an expert user in one domain will be a novice in others. Users will be clerical workers, information specialists, executives, engineers, and others. Attention to novice-oriented and to tutorial help features is required.

Users also want and deserve the reward of increased proficiency and capability from improvements in their skills and knowledge, and in their conceptual orientation to the problem domain and to their workshop's system of tools, methods, conventions, etc. "Advanced vocabularies," short concise control notation and conventions in every special domain will be important and unavoidable.

A corollary feature is that workers in the rapidly evolving augmented workshops should be involved continuously with testing and training in order that their skills and knowledge may most effectively harness available tools and methodology.

3) Ease of Communication Between Subsets and Addition of Workshop Domains

One cannot predict which domains or application systems within the workshop will want to communicate in various sequences with which others, or what operations will be needed in the future. Thus, results must be easily communicated from one set of operations to another, and it should be easy to add or interface new domains to the workshop. A corollary is that the total workshop may contain a very large number of tools and services. Some users may have access to only a subset of its capabilities while others will have access to many or all capabilities.

II User Interface

8 Design Issues

As described below, we expect the workshop to be embedded in a computer network and thus communication between tools and between users must take place across both process and host boundaries according to well specified conventions and protocols [5][6].

4) User Programming Capability Or User Interface Extensibility

There will never be enough professional programmers and system developers to build or interface all the tools that users may need for their work. Therefore, it must be possible, with various levels of ease, for users to add or interface new tools, and extend the language to meet their needs. They should be able to do this in either a variety of programming languages with which they may have training, or in the basic user-level language of the workshop itself.

5) Range of Workstations and Symbol Representations

The range of work stations available to the user will increase in scope and capability. These work stations will support text with large, open-ended character sets, pictures, voice, mathematical notation, tables, numbers, and other forms of knowledge. Even small portable hand-held consoles will be available. The multiplicity of possible terminals indeed raises the question of whether a consistent set of control and portrayal conventions is possible.

As hardware decreases in cost, more and more capabilities will be placed in the work station both in the form of user interface aids and facilities, and in the form of frequently used tools.

6) Distributed Nature of The User Interface Processes

The collection of facilities to support interfaces with the system of tools can be conceived of as a single service as seen by the user. These facilities may all reside in a processor in the work station or be distributed in two or more processors, depending on the level of their sophistication and state of the art with respect to cost, hardware capability, and so forth.

7) Tools Embedded in a Computer Network

The computer-based tools of a knowledge workshop will be provided in the environment of a computer network, such as the ARPANET [7]. For instance, the core functions will consist of a network of cooperating processors performing special functions, such as editing, publishing, exchanging documents and messages, data management, and so forth. Less commonly used, but important functions, might exist on a single machine. The total computer-assisted workshop will be based on many geographically separate systems.

Once there is a "digital-packet transportation system," it becomes possible for the individual user to reach out through his processor to other people and other services scattered throughout a "community." The "labor marketplace" where he transacts his knowledge work will be literally independent of geographical location.

Specialty application systems will exist in the way that specialty shops and services now do--and for the same reasons. When it is easy to transport the material and negotiate the service transactions, one group of people will find that specialization can improve their cost/effectiveness, and that there is a large enough market within reach to support them.

II User Interface

8 Design Issues

And, in the network-coupled computer-resource marketplace, there will be a growth of specialty shops, such as application systems specially tailored for particular types of analyses, or for checking through text for spelling errors, or for doing the text-graphic document typography in a special area of technical portrayal, and so on. There will be brokers, wholesalers, middle men, and retailers.

Bb1g3

The key point to emphasize is that even when hardware costs decrease to the point where a user can perform 90% of his work using tools and information that operate in the processor in his work station, he will want to have access to a computer network to.

- a) Communicate in various forms with others
- b) Access very large or special databases
- c) Access special tools that run elsewhere

Bb1g4

8) Problem Orientation of the Command Language and Tolerance for Ambiguity

Bb1g5

The user has a task that he wishes performed by the system. Depending on the nature of the task and operations available to him on the system, he may be able to express what he wants accomplished in a single "statement" or command to the machine, or it may require a series of commands.

Bb1g6

One of the goals of the designers of the command language and system is to understand the nature of the user's application domain so that the user can express his needs with words that are similar to his natural problem solving vocabulary and language forms. The machine should then break down the request into smaller steps as required.

Bb1g7

If there is ambiguity in the user's command, the machine should recognize it, if possible, and prompt appropriately for clarification. There is still much research and development required to fully meet this goal.

Bb1g8

Many people hope to allow novice users or users in certain applications to use natural language in making statements to the machine. This capability will require models of the user and task domains for understanding.

Bb1g9

Even when systems are able to interpret commands given in natural language, the precision and usage efficiency of appropriate artificial languages will make the latter's continued use preferable, especially for skilled users.

Bb1g10

Given the above general considerations as background, we can move on to examine features of the NLS user interface in more detail.

Bb1g11

MORE DETAILED DISCUSSION OF THE NLS USER INTERFACE

Bb1g12

A command language must allow unambiguous specification of what the user wishes accomplished. The operation to be performed, and the entities or information items (arguments) to be acted upon, or used to determine what is to be acted upon, must be specified. These can be specified in a variety of ways: by typing them in in full or in some form of abbreviation, by pointing at them on a screen, by pronominal reference, by implication from context, or by use of default values where appropriate. The order of their specification, the syntax or grammar of the language, can have various forms. For example, operational command-words can be specified, followed by the arguments, or vice versa. Arguments can be in fixed positions or explicitly

II User Interface

8 Design Issues

named and occur in any order. Some arguments or command-words can be optional and require special characters to indicate their presence. Arguments or command-words can have defaulted values under certain conditions. Pronominal references can be allowed to refer to previous occurrences. Arguments may be given types by the system and language designer for more extensive error checking and feedback.

Arguments and keywords can be specified by complete or partial typein (there are a variety of forms of command recognition that are discussed later) or designated by pointing to representations on a display or by use of specially coded function keys. Or, the machine may ask questions and the user just fill in the blanks.

Depending on the characteristics of the computer and communications system, it may or may not be possible to provide command word or keyword completion, prompts or other feedback, argument checking, default value fill in, and so forth, during the command specifications.

For example, in line-at-a-time, half-duplex systems, the user usually must complete the entire specification of the command before transmission to the system, while in character-at-a-time, full duplex systems, the system can react to each character received and provide more extensive aids to the user during command specification.

The above discussion outlines just a few of the many choices available to the language designer. As the purpose of this section is not to be a complete tutorial on all possible choices available and their advantages and disadvantages, the following discussion gives only the main NLS command language features and the motivation for their adoption.

THE NLS COMMAND LANGUAGE

The NLS command language generally has the following form, where angle brackets group meta symbols:

< operation specification > < operand specification > < command completion >

The fields in a command are of a fixed order, although some commands have optional fields that can be specifically requested. Other fields can have a system-supplied default value. Because NLS operates from a character-at-a-time, full-duplex system, several levels of help are available, as described later, for giving cues and prompts, explicitly listing options or syntax, and giving full documentation on what the system expects next during command specification. It was not felt that much would be gained for novice users by allowing fields to be specified in any order by using explicit field names. Novice users do not need to be aware of optional fields.

As much as possible NLS makes the operational specification of the form verb-noun followed by arguments and possibly other keywords. We have also tried to maximize the fullness of the verb-noun matrix.

This approach seemed to be natural, and follows normal English imperative forms to aid learning. The choice of verb-noun form seemed to fall out naturally when considering such important areas as editing. A given verb, such as DELETE, can naturally be applied to many entities, such as statement (: paragraph, title, equation), character, number, text, file etc. Learning is easier if the user can form a model of how the system works that can be

II User Interface

8 Design Issues

consistently applied. In this case, a user can learn n verbs and m nouns and understand that generally, if it is meaningful, they can be used in pairs. Having learned $n+m$ vocabulary terms, he can apply them in the form of $n \times m$ commands. 8d7a

We have tried to pick command keywords that have normal usage related to the operation described. A synonym capability would be easy to implement. 8d7b

Four forms of command keyword recognition are provided to enable the user to choose the one most appropriate to his terminal type, system response, previous system experience, and present NLS experience level. We have worked to pick an operational vocabulary for the present system that guarantees keywords to be unique in a maximum of three characters: 8d7c

1) A single-character mode allowing high-speed single-character recognition of the most commonly used commands; less commonly used commands require an escape character followed by enough characters for unique recognition: With large and expanding command sets one cannot choose keywords with mnemonic value and guarantee uniqueness with the first character. This mode is generally preferred by experienced users because of the simplicity and speed with which frequently used operations can be expressed. We find that experienced users are very concerned that commands be formed with the minimum number of input operations, and that commands have the richness needed to specify adjective or adverb type operations as needed. There is thus some conflict in certain commands between these goals for the experienced user and the need for command simplicity for the novice. 8d7d

2) A demand mode requiring a right delimiter to initiate recognition: This has proved to be popular for new users of typewriter terminals, particularly those with experience using the TENEX operating system. 8d7e

3) An anticipatory mode requiring the user to type enough characters until the command is uniquely specified; the system then automatically fills in the remainder. 8d7f

4) A fixed mode that guarantees recognition on entry of three characters. 8d80

Given the implementation approach outlined later, it is quite easy to add other recognition modes, such as allowing the user to choose keywords from a menu displayed on the screen. However, experiments have shown that the time it takes to point at some item on the screen is equivalent to several keystrokes and thus would be disadvantageous to skilled users, although possibly of value to novices [2][3]. 8d81

Modes 3 and 4 have not turned out to be heavily used. 8d82

Operand argument specification is contained in a number of fields that are variable with the type of command. All commands of a similar type have had the order of the operands made as consistent and as natural (relative to normal English usage) as possible. Infrequently used operand fields are optional and novice users need not be aware of their existence. 8d83

Related to argument specification is the problem of choosing argument delimiters. One can recognize the following delimiting functions. 8d84

- 1) Delimiting command words
- 2) Delimiting arguments
- 3) Delimiting optional arguments, selection type, or command word fields.

II User Interface

8 Design Issues

4) Delimiting commands

5) Selecting arguments off a display screen, and confirming the selections

Bd 3a

One could choose separate characters (codes) to represent each of these functions. To do so seemed to us to add an unnecessary complication for the user and so, except for using a special character to indicate an optional argument, selection type, or command word, a single code is used for the other function in NLS. We call this code "Command Accept" (CA) even though it is used for other purposes as well. The system allows the user to define which keyboard character is to serve this function if he finds the system's default to be inconvenient. One of the buttons on the mouse also serves this function.

Bd 4b

Arguments can be typed in, defaulted where appropriate, or specified by pointing to appropriate entities on the display screen.

Bd 4c

There are three flavors of command completion.

Bd 5

1) Completion of the command indicating execute the command and return to the base state to await input of the next command: The default indication for this form is one of the buttons on the mouse in DNLS, which is translated into a control character. Command completion is defaulted to be CR in TNLS. The use of CR in TNLS is quite natural and generally does not conflict with textual input as most text in NLS is typed in without explicit CRs and is appropriately formatted by the system for various output devices. If the TNLS user wishes to input an explicit CR in his text file, he must precede it with an escape character. If he has need to enter many CRs in his text string, he can redefine the completion character, Command Accept, to be some other character.

Bd 5a

2) Completion of the command and return to an appropriate point for quick repetition of the command. Repetition mode continues until explicitly commanded to delete out of it. This mode is very useful when a delete or other operation is repeated several times.

Bd 5b

3) Completion of the command and entry to insert-statement mode for addition of new paragraphs or other text statements: This mode is like command repeat above except that it always takes you to the insert command. It is used frequently when one adds, replaces, or moves text, and then wants to follow it with new statements. It speeds text input when inserting sequences of paragraphs.

Bd 5c

The system is to be used from a variety of terminal types, including both typewriter-type terminals and displays. The two-dimensional displays are to be the preferred work station types whenever a design decision must be made between language forms possibly favoring one type or the other.

Bd 6

It was decided to make the command language syntax for the typewriter (TNLS) version and the display (DNLS) version as close as possible, except where the difference between the one-dimensional and two-dimensional media clearly prohibits this or would seriously limit one or the other version. This decision was made to allow people working in environments consisting of both typewriter and display terminals to be able to move back and forth with ease.

Bd 6a

II User Interface

8 Design Issues

The system has been organized into clearly defined subsystems with uniform rules for their entry and exit. Any subsystem can be entered from any other, either to "execute" a single command with automatic return or to perform a chain of commands. The user can return, either to a specifically named subsystem in the path of subsystems traversed or enter a new subsystem. The issue of how to group commands into subsystems has to do with training and patterns of use rather than system constraints. It relates to learnability and, to some extent ease of command specification using single characters, and to "knowing where you are" in a command or operational space.

One could construct a system where all commands were in a single subsystem. Study of the command set of a large system particularly conceived of as a set of tools shows that operations tend to group together such that to perform a given task, such as sending a message or calculating a budget, generally require several related suboperations. Certain operations, such as moving in information space or seeking help, tend to be used as suboperations of many or all tasks. This latter observation led to "universal" commands available from within any subsystems. One can also imagine certain commands to be needed frequently in just two or more subsystems and thus implemented in each subsystem having the need. There are now no instances of this case in NLS. The ability to execute a single command in another subsystem with automatic return has been very useful.

Provision has been made for options the user can control as he wishes for the amount of prompting, feedback, and for setting other user interface parameters whenever it seemed a standard interface might not be appropriate to some significant class of users.

A mechanism is implemented that enables the user, or someone acting in his behalf, to create a file stating what options he wants to run with. The system thereafter automatically sets these options when he enters. This facility can also be used with small extensions to subset commands. This user option capability, when coupled with the ease by which the user interface can be redefined using the Control Meta Language described below, makes possible tailoring the user interface to specific users or groups of users.

All operations that have a natural inverse command have been given one (although NLS still does not have an "undo" facility). A general undo/redo facility has a number of technical difficulties and its value can be questioned. However, the ability to undo or redo the last one, two, or three commands would clearly be useful.

User Programming: As indicated earlier the ability of the user to extend the system himself is important. There is a tradeoff between ease of extension specification and operational efficiency. In providing such a facility one does not have to be deeply concerned with efficiency if the task handled by the extension is performed infrequently. If the operation is performed frequently, then it should probably be inserted as a system feature and implemented efficiently by professionals. This area is ripe for much additional development. The extensions must be specified in some language to indicate what sequence of events is to take place, what arguments to collect, and so forth, when a given user action is performed.

NLS now offers two forms of extensibility. The first allows users with some basic programming knowledge to write programs in the Algol-like L10 language in which the system is implemented, calling on NLS system primitives as needed. They can use the Control Meta

II User Interface

8 Design Issues

Language to specify a user interface if desired. These programs can be installed by the user as one of his default subsystems, loaded as subsystems as needed, or used as content analyzer patterns [8].

B a 1 C a

The user can also write sequences of NLS commands and have these sequences executed at will. A specific sequence of commands can be automatically invoked when the user first enters NLS.

B a 1 C b

HELP, STATUS, AND PORTRAYAL FACILITIES

B e

Let us now consider each of the information spaces and the type of feedback, help, and other status information available to the user.

B e 1

The user interface must implement a man/machine dialog. In this section, we discuss issues from machine to man. The discussion centers around the use of displays, with comments on how the problem is dealt with for typewriters. Let us examine some of the types of information that the user needs in order to keep his bearings.

B e .

There are three main areas or dimensions along which the user needs information to help him a) to know where he has been, b) to know where he is, and c) to know where he can go from here. Clearly the command language and user interface must offer provisions to move in these spaces as well as obtain status.

B e 4

1) Information Space

The user needs to know where he is in his information space, and what view or portrayal of the many possible is being displayed to him. Generally he arrived at his present position from previous points and he may want to be able to backtrack to previous points or views as well as to move on.

B e 3 a

2) Subsystem or Tool Space

In workshops containing many tools and commands, the user needs to know which tool is active and possibly needs to know which ones he was in previously and their order, and which ones he can enter from here.

B e 1 b

3) Command Syntax Space

During the specifications of a command, the user may need to know what he can or is expected to do next and how to back up to a previous point.

B e 3 c

The NLS display screen is organized into windows as described in some detail in [9]. These windows are arbitrary rectangles. Windows can be displayed essentially all the time or overlaid with others. Windows can grow dynamically. Some windows are allocated and displayed or not displayed under system control for status and feedback information. Others can be created and manipulated by the user for display in his information space. With typewriter terminals, one does not have this two-dimensional random display capability and while the same information can be given to the user, less can be given automatically or at least must be given in an altered form.

B e 4

1) Information space

B e 4 a

The present NLS information space is hierarchically organized. A user has a directory or directories within which there are files. A file can contain notes on many subjects stored

II User Interface

8 Design Issues

under various headings, his mail, or single documents. Files in turn are hierarchically organized as a tree of information nodes (now text strings but soon to be generalized to include illustrations and other entities).

Files can contain cross citations to specific points within other files or the same files, thus creating networks. NLS has appropriate commands for moving within and between files and for obtaining a display of the path over which one has traveled and commands for backtracking along this path [1].

Display screens have a limited number of lines within which to display information, and typewriters, even at 30 chars/sec or higher, cannot quickly and easily print out large documents. Also, the user often wants to see a summary or overview of a document or have it formatted in special ways to aid his understanding. To meet this need for easy control of information portrayal, NLS has a concept called "view specification." The user can change his "view" within the commands for moving in information space or by separate command. So that he can be reminded of his current view, the most commonly used view parameters are fed back to him in a small window in the upper right hand corner of the screen. When he is at a point in a command where it is permissible to change views, this fact is feedback both by prompt (if prompts are turned on) and by enlarging the characters in the view-feedback window. For more discussion on moving, viewing, and portrayal in NLS see [1][4].

2) Subsystem or tool space

NLS is viewed as a collection of tools (subsystems) that can be used cooperatively or stand alone. Each subsystem contains a number of logically related commands and has a name, such as Base (the collection of editing and file manipulating commands), Calculator, etc. All the tools work on information in the same file structure and the user can move from one tool to another, or execute commands on a single command basis in any tool from any other tool, as mentioned earlier. The user can receive a display of subsystems available to him or an ordered list of the subsystems in which he has previously been.

The name of the current subsystem within which he is operating is fed back in a small window in the upper left-hand corner of the screen in DNLS and as a four-character prompt in TNLS.

3) Command syntax space

There are several levels of feedback and Help available to the user in formulating a command to the system (15). Each is described below. The Help database clearly is also generally useful for understanding the system as a whole.

a) Command-word recognition:

The options here were described earlier and this mode is primarily useful in minimizing keystrokes and in triggering additional feedback.

b) Noise words:

When the system recognizes a command-word or field it generates what we call "noise words" set off in parentheses so the user can distinguish between what he has input and what the system has added. The noise words aid the user in remembering what to do next.

II User Interface

8 Design Issues

Novice users report that noise words are one of the most useful initial aids. As more experience is gained, the other aids take on more importance. This is an important point to note: users at different levels of experience value different forms of feedback. Usefulness is not only determined by the inherent characteristics of the aids, but also, by how they are implemented.

c) Prompts:

When the user completes the specification of a field in a command, he is prompted with some terse characters indicating the type of thing expected next and the alternatives available to him for specifying, selecting, or addressing the needed argument. Users can turn prompts off, which some users of TNLS do when they reach a certain level of proficiency, although many highly skilled users always operate with them on. DNLS users tend to always operate with them on because the high speed of the display does not slow down work while providing useful information. Users can also specify terse prompting in which case optional fields are not prompted for. Beginning users have indicated that prompting is useful, but would like prompts to be more mnemonic and of English type and word length.

d) Next Options and Syntax:

If the noise words and prompts are not sufficient to jog a user's memory about what options are available to him next, he can strike a ? or a <Control-S>. If he strikes a ?, the system displays, in alphabetical order, all the command-words that are legitimate for the next field or more extensive information than is available in the prompts for other fields. If he strikes <Control-S>, the system prints out the syntax of the command from his present position to the end of the command. The ? facility is extensively used and is very useful in refreshing one's memory about infrequently used commands or new commands for a user with only a basic knowledge of command system concepts and vocabulary. The <Control-S> feature does not seem to be extensively used at present and may indicate that the ? facility is sufficient.

e) Help, Data Base:

If the above facilities are not sufficient because of uncertainty about a basic concept or vocabulary word or the user wishes more information about the effects or use of a command, he can enter the the Help tool. Entry can be from the basic command level or from any point during command specification. In the latter case, the system utilizes the information input at this point to take the user to an initial point that describes the command and field where he is located. (15)

Once in the Help Database, a simple set of command conventions and the organization of the database allow the user to easily move to reference related subjects or move to new subjects or back up to higher level descriptions (15).

f) Active Tutorial Help:

The next level of Help facility would be an active tutorial facility. We have not yet implemented such a facility but can see its value. An example of such a facility is the work going on at BBN on the NLS-Scholar system [10].

II User Interface

8 Design Issues

ERROR MESSAGES AND RECOVERY

Error messages indicating an incorrectly spelled file name or improperly specified entity are fed back to the user in a window at the top of the screen. The user is left at an appropriate point within the command specification or where necessary he must start over again to respecify the command. The text of error messages is important and should be as specific to the problem as possible. This has implications within the system design for trapping error conditions as early as possible and determining the appropriate message for the specific error and total context of the user. While we have made progress in this area, there is much more that could be done to meet the need stated above.

There are now no automatic error correction mechanisms built into the system, such as spelling correction or "Do What I Mean" type facilities. These would probably be useful to add when resources permit.

EDITING AND BACKUP DURING COMMAND SPECIFICATION

The user can perform certain simple editing and backup operations during command specification. At any point during command specification he can do a "command delete," which will take him back to the basic command level. This is useful if he gets confused and wants to return to a known state or changes his mind about which command to perform next.

The user can delete the last character input or last selection made on the screen with a "backspace character" keystroke or button push on the mouse. He can repeat this process and continue the incremental backup process to the basic command state.

He can also delete the last word input, or the field specified to date, with a "backspace-word" keystroke or button push on the mouse. He can also repeat this process backwards to the basic command state as well.

IMPLEMENTATION

The mechanisms and databases needed to implement the user interface have been modularized and isolated as a "Frontend" that can run on a separate computer, such as a mini-computer close to the user, and communicate with the basic tool information processing routines ("Backend") over a communication network. The Frontend consists of terminal handling capabilities [9], a command language interpreter (9), and two databases: a Grammar representing the language syntax and noise words; and a User Profile indicating how the user wants various parameters set for him, such as his prompt and command recognition modes, keyboard key translations, etc. The Grammar is generated from a high-level description of the user interface written in a language special for this purpose we call Control Meta Language (9.).

Given this particular system organization it is very easy to tailor, subset, or modify the user interface for individuals or groups, or to create interfaces for new tools.

Furthermore all the levels of help information, except the Help Data Base, are derived from the Grammar, which guarantees their correctness as the system changes and is debugged. Various forms of hard copy documentation, such as command summaries, are also derived from the Grammar representation.

II User Interface

8 Design Issues

ACKNOWLEDGMENTS

The NLS user interface has evolved over many years and reflects thousands of console hours of user experience. Contributions have been made in this area by many members of the ARC staff during this period.

REFERENCES

- [1] (8A4) (8A5) (8e4a2) (8e4a3) Douglas C. Engelbart, William K. English. A Research Center for Augmenting Human Intellect. In AFIPS Proceedings, 1968 Fall Joint Computer Conference, Vol. 33, Part I, p. 395-410, 1968. (3954.)
- [2] (8A4) (8D3E) William K. English, Douglas C. Engelbart, Melvin A. Berman. Display-Selection Techniques for Text Manipulation. In IEEE Transactions on Human Factors in Electronics, Vol. HFE-8, No. 1, March 1967, p. 5-15. (9694.)
- [3] (8A4) (8D3E) Douglas C. Engelbart. Design Considerations for Knowledge Workshop Terminals. In AFIPS Proceedings, Vol. 42, 1973 National Computer Conference, pp. 221-227, 1973. (14851.)
- [4] (8A5) (8e4a3) Douglas C. Engelbart, Richard W. Watson, James C. Norton. The Augmented Knowledge Workshop. In AFIPS Proceedings, Vol. 42, 1973 National Computer Conference, pp. 9-21, 1973. (14724.)
- [5] (8B1C2) James E. (Jim) White. Version 2 of the Procedure Call Protocol (PCP). Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. PCP-COVER.NLS;5.. (24590.)
- [6] (8B1C2) Jonathan B. Postel and James E. (Jim) White. Notes on a Distributed Programming System. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 21-MAR-75. (25613.)
- [7] (8B1G1) Lawrence G. Roberts and Barry D. Wessler. (University of Utah, Computer Science Department). The ARPA Network. Advanced Research Projects Agency, Information Processing Techniques, Washington, D.C. MAY-71. (7750.)
- [8] (8d10a) No Author. L10 Users' Guide: Content Analyzer. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. L10.NLS:7.. (24426.)
- [9] (8e4) (8e4H1) Charles H. Irby. Display Techniques for Interactive Text Manipulation. In Proceedings of the National Computer Conference, 1974, p. 247-255. (20183.)
- [10] (8e4e6) M. C. Grignetti et. al. An Intelligent Online Assistant and Tutor--NLS Scholar. AFIPS Conference Proceedings, Vol. 44. Anaheim, California. MAY-75. (25054.)

A Command Meta Language for NLS

(by Charles H Irby, Charles F Dornbush, Kenneth E Victor,
and Donald C Wallace)

CONTROL META LANGUAGE -- CML

INTRODUCTION AND MOTIVATION

To facilitate the easy formal description, implementation, and modification of the user interface to a range of interactive application programs, the Augmentation Research Center (ARC) has developed the Control Meta Language (or CML). This was an outgrowth of earlier efforts to accomplish the same goals at ARC [1][2][3][4][5].

The goals of this development were the following:

- 1) Provide a means for easily changing and experimenting with the user interface to an interactive application program.
- 2) Allow for the independent manipulation of
 - a) the commands available to the user and
 - b) the interaction methodology and techniques that are used to specify commands.
- 3) Provide builders of new interactive application programs with a facility for easily creating the user interfaces for their new programs.
- 4) Provide the user with consistent and coherent command language features across a collection of application programs, or what might be termed "tools."

Independent of the tool to which the user is giving commands, he does so using the same methods for specifying which commands he wishes executed, the same methods for specifying arguments or parameters to commands, gets the same type of prompting and requests help in the same way, always. In addition, the general syntactic form(s) of commands should be the same from tool to tool unless there is good reason for the tool to deviate from the standard. Of course the particular commands and vocabularies will vary with the tool and in fact the same verbs may be used with quite different semantics in different tools, but at least most other aspects of the command language (including asking for help and being prompted for the proper type of input) should stay the same across tool boundaries.

- 5) Provide tools with well-formed commands.

Many operating systems and application programs have elected to use half duplex, line-at-a-time terminals because of the increased computer efficiency provided by this approach. Other operating systems and application programs have chosen, instead, to utilize character-at-a-time full duplex terminal disciplines because of the opportunity this provides for utilizing a more human-engineered command language.

Preceding page blank

II User Interface

9 Command Meta Language

The CML system is an attempt to combine these two approaches into a COMMAND-AT-A-TIME system, where the application programs do not directly interact with the terminal, but rather receive fully specified commands from the Frontend. At the same time, the CML interpreter will attempt to provide the user with the best possible human-engineered command language discipline.

Although initially this was done by issuing direct procedure calls on tools (requiring that the CML interpreter and tools be written in the same language and link-loaded together), it is proposed that this eventually be done by issuing "remote" procedure calls to "external" procedures in the tools to actually execute commands. This will be accomplished through the Procedure Call Protocol [6].

6) Provide a terminal-independent interface to the tools.

Because the CML interpreter handles all terminal interaction, it will present to the tool a small number of virtual terminal classes. Thus, once a tool is developed, little attention need be given to the type or particular characteristics of the terminal the end user may choose to employ while using the tool. In fact, the cost of creating new tools should be considerably reduced because of these facilities.

This means that even though the creators of a tool envisioned the user sitting at a typewriter terminal, the user who happens to be using a display terminal with a pointing device may be able to interact with the tool in a two dimensional sense, pointing to arguments on his screen instead of typing them.

For tools which wish to make more extensive use of a display terminal if the user has one, the CML interpreter presents primitives for allocating windows on the display and allows the tool to write/delete/move/make invisible items displayed within the windows.

7) Make asynchronous operation possible.

In some instances, it may be possible for the execution of the user's commands to be accomplished in parallel with subsequent command specification and execution. This frees the user to do other things while a lengthy command is being executed by a tool.

8) Provide standard mechanisms for presenting status or error conditions to the user.

an error should consist of the following:

a human-readable error message

a code indicating whether this error caused the command to be aborted, completed or undefined and whether the tool is now in a state to receive more commands or should be restarted.

9) Provide the user with enhanced, consistent help facilities while using any such tool.

10) Allow for a common statistics gathering point for analysing user interaction characteristics such as error rates, frequency of issuing given commands or groupings of commands, and average user-observed execution times for commands.

11) Provide a convenient way of subsetting the commands available to the user.

II User Interface

9 Command Meta Language

In the current case, a hypothetical computer was postulated that had as primitives instructions that interacted with a human user. A "program" for this machine forms a tree-like data structure, that we call a Grammar. The two addresses of an instruction are called the "successor" and "alternative" addresses. The successor address points to another instruction to be executed if this instruction succeeds (i.e., is "TRUE"). The alternative address points to another instruction(s) to process in parallel with this one. That is, a number of instructions are processed in "parallel" such that when any one of the instructions in the current set of alternative instructions succeeds, then the Program Counter is advanced to its successor. That instruction and its alternatives are then processed.

At any point in time, the machine is attempting (presumably by interacting with a human user) to choose a path through the tree. An illustration might be helpful here. At a certain point in time the Program Counter might point to an instruction to recognize a command word, a "reserved" word in the command language. This instruction might have alternatives which are attempting to recognize other command words. These command words might represent the verbs of commands the user can give to the system or might represent refinements to a command already partially specified. The machine picks a path through these alternative command words, although how this is accomplished is left purposely unspecified. For it is precisely the "HOW" of this machine's path finding that embodies the human-factors considerations and human-machine interaction disciplines, which can, and in our case do, vary from user to user. Thus, how the system interacts with the user is independent of the commands the user has available -- one can be changed while the other remains constant.

Given the existence of the model for this hypothetical machine, we then developed a formal language (CML) and compiler for this machine. It is with this formal language that the staff of ARC now specifies the user interface for the NLS Knowledge Workshop tool system [2] we have been developing for several years. The form of this language is the chief topic of this paper. The object code produced by the CML compiler is called a control language grammar (or simply a Grammar).

We have developed and used for several months (on a production basis) a simulation of this computer which we call the Command Language Interpreter (CLI). Embodied in this implementation are the principles for human-computer interaction that have evolved through many years of usage and evaluation of NLS and other systems.

It is this interpreter that interacts with the user to help him specify commands for the system to execute. It prompts him for the type of input required (if the user wants it to), shows him the syntactic form of specific commands on request, shows him his actual alternatives at any point in the specification of a command on request, and can invoke a semantic help facility if the user requests. This semantic help is derived from a structured database provided by the implementers along with the user interface (CML description and grammar) which attempts to describe in English the intended use of the various commands and the tool as a whole. This database is highly structured to allow the user to get the information he needs quickly without wading through pages of output (8,15).

Thus, a tool now consists of three parts: 1) a CML description of the user interface, 2) a semantic help database, and 3) an execution module that carries out the commands specifiable by the user.

II User Interface

9 Command Meta Language

To allow the user interface to be individually tailored, we have added a database called the "user profile" which describes to the CML interpreter how much prompting and feedback the user wants, what recognition scheme he wishes to use to select command word alternatives, and many other idiosyncratic features of the user interface (8). There is a special set of commands for modifying this database and consequently the behavior of the system.

And finally, to facilitate user analysis, we have added a user statistics database in which the interpreter can record which commands were used, whether or not errors were made in the specification of the commands, the execution time of the command, and other statistics.

We are now involved in a second generation CML system which will provide a Frontend system for the National Software Works (NSW) program sponsored jointly by ARPA and the Air Force. In this second generation system the execution functions that implement the semantics of the commands are called through the Procedure-Call Protocol (PCP) and the Multi-Process Support System [6]. Thus, unlike the current CML system, the execution functions may be written in any language which can be interfaced to PCP. In addition, certain aspects of the language will be improved as discussed in the conclusions below.

The intention with the NSW system, as with NLS, is to provide the user access to a number of general or specialized tools in such a way that the command discipline he uses remains constant even though the particular vocabulary changes from tool to tool as appropriate to describe that tool's functions. In the case of the NSW and for future releases of NLS, this user interface will reside not only on a PDP-10 but will also be available on a dedicated Frontend computer (PDP-11 or other mini-computer) for better responsiveness and less expense. We anticipate that heavily used tools or commands will, in time, actually be executed in the Frontend computer. In addition to increased system responsiveness, this will reduce network communication and will afford users a certain amount of insulation from network or large-computer unavailability.

As described above, the Frontend system consists of the following:

- 1) A formal language (CML) for specifying NSW user interfaces
- 2) A compiler for that formal language that runs under TENEX as a subsystem or from NLS
- 3) Tool grammars, products of the CML compiler or any other such program
- 4) A CML interpreter that processes a CML grammar in order to work with the user in specifying syntactically correct commands to the NSW.
- 5) A user profile database that is used by the CML interpreter while interacting with the user. This database allows the Frontend to be tailored to the individual preferences of the users.
- 6) A user statistics database, where, if desired, statistics can be accumulated on commands used by a user, error rates, etc.
- 7) Access to a semantic help tool which is employed by the Frontend when the user requests semantic level help with a tool or a command. It is presumed that each tool, in addition to supplying the Frontend with a grammar will also supply it with the name of a

II User Interface

9 Command Meta Language

help database whose structure and content, as with the grammar, are the sole responsibility of the tool builder/supplier.

This help tool could also be kept informed of the user's dialog with the Frontend and could have access to the tool grammar, the current parse state of the user, and the user's profile.

The rest of this section describes in more detail the Control Meta Language and the CML interpreter.

As discussed above, the Control Meta-Language (CML) is a vehicle for describing the syntax of the user interface to application programs. The syntax is described through the tree-meta alternation (denoted by /) and succession (denoted by juxtaposition) concepts [7][8][9][10]. The semantics are introduced via built-in functions, semantic conventions, and parse functions.

No attempt is made to allow for the full semantic description of any command via CML, but it is hoped that the Frontend interface (parsing and feedback operations) may be explicitly accommodated with these facilities. It will still be necessary, and desirable, to use execution functions to perform the low-level semantics of the command. We call the collection of these execution functions and their support routines and data structures the tool "Backend." The CML describes how the command "looks" to the user, rather than what it does inside the tool.

The CML supports zero look ahead, phrase structured, context free control languages.

USE OF CML

The user interface for a tool is defined in the CML specification language. This CML "program" is then compiled by the CML compiler (written using ARC's tree-meta compiler system [9][10] to produce object code (called a Grammar) which is interpreted by a Control Language Interpreter (CLI). The Control Language Interpreter is cognizant of the device dependent feedback and addressing characteristics of the user's terminal through an appropriate interface to a terminal control module described in [11].

SYNTAX NOTES

The following meta symbols are used in this discussion of the CML:

.ID	An Identifier.
.SR	A quoted string.
\$	Zero or more occurrences of the following element.
/	Denotes alternatives. A/B means A or B.
#	At least one occurrence of the following element.
%	Brackets comments.
()	Used for grouping to control precedence.
[]	Used to denote optional elements.
.	Precedes literal characters.

II User Interface

9 Command Meta Language

"	Encloses literal strings.	data
# <...>	At least one occurrence of the following element, separated by whatever ... represents.	data
\$ <...>	Zero or more occurrences of the following element, separated by whatever ... represents.	data

ELEMENTS OF CML

PROGRAM STRUCTURE

The basic compilation structure of a CMI program is described by:

```
file          = "FILE" .ID $dcls $rule #subsys "FINISH";
```

Explanation:

The "file" construct brackets the definition of control language subsystems.

Declarations of variables, execution and parse functions, and external identifiers may be made at this level. In addition, global parsing rules may appear here and be invoked in commands by simply specifying their names.

```
subsys      = "SUBSYSTEM" .ID % subsystem handle %
              KEYWORD .SR % recognition string %
              #(command / rule) "END.";
```

Explanation:

The "subsystem" construct brackets a set of rules or commands (generally a set of related commands that the command language designer wants to cluster together). Commands beginning with the command-word **COMMAND** are linked together to form a command language subsystem.

```
command     = ("COMMAND" / "INITIALIZATION" /
              "TERMINATION" / "REENTRY") rule ;
```

```
rule        = .ID '= exp ` ; ;
```

Explanation:

The subsystem may include a rule preceded by the reserved **INITIALIZATION** or **TERMINATION**. If specified, these rules will be executed once upon subsystem initialization/termination, respectively. This enables, for example, a tool to open and initialize a work file when it is started and to close it after the user's last command has been issued.

The subsystem may include a rule preceded by the command-word **REENTRY** which will be executed upon reentry in the subsystem after executing commands in other subsystems.

The command Language Interpreter allows the user to freely move among subsystems. Thus, the user may give commands to one subsystem for a while, then give commands to another, and finally return to the first. The **REENTRY** rule will be executed when the

II User Interface

9 Command Meta Language

user resumes giving commands to the first subsystem. This might be necessary, for example, to ensure that a work file or data structures were still in a consistent form.

Each rule/command is named with an identifier. This name may be used as a term in any other rule, indicating that the named rule is to be invoked at that point in the parse.

DECLARATIONS

Declarations are used to associate attributes with identifier names which are used in CML programs. If not declared, identifiers are defined by their first occurrence according to the following rules.

- 1) Identifiers appearing on the left hand side of an assignment statement are defined as "VARIABLES."
- 2) Identifiers followed by a subscripted list are assumed to be of type "FUNCTION."
- 3) All other undefined identifiers are assumed to be names of parse rules or commands.

The syntax of the declare statement is given by:

```
dcls = ("DCL" / "DECLARE")
        ( ["VARIABLE" / "FUNCTION" / "PARSEFUNCTION" /
          "EXTERNAL"] # <'> .ID / "EXT-KEYWORD" # <'> .SR);
```

If a declare attribute is not given, type VARIABLE is assumed. Identifiers which are implicitly defined as type FUNCTION or PARSEFUNCTION are EXTERNAL symbols and will be linked by the loader to externally defined symbols with that name.

Semantics of the declare attributes:

VARIABLE:

a cell which holds pointers to CML records

FUNCTION:

arbitrary processing function usually invoked to carry out all or part of the execution of a command

PARSEFUNCTION:

a function which is used to extend CML. Such a function processes input, and is called in "parsehelp" and "parseqmark" mode to supply a prompt string and a ? string, respectively.

EXT-KEYWORD:

precedes a list of command-word strings (# <'> .SR) and indicates that the named command-words are globally defined elsewhere in the system.

EXTERNAL:

associates an external symbol with the named rule/variable permitting separately compiled programs to reference the named rule/variable.

II User Interface

9 Command Meta Language

RECOGNIZERS

Keyword Recognition

The process of command-word recognition is independent of the description of the command-words for CML. In the CML description, each command-word is represented by the full text of the command-word. The algorithm used to match a user's typed input against any list of alternative command-words is known as command-word recognition, and is a function of the Command Language Interpreter and is independent of the CML description of the command.

Keywords are written in the meta language as upper-case identifiers enclosed in double quote marks optionally followed by a set of command-word qualifiers.

command-word = .SR [' ! #qualifier ' !]

The qualifiers serve to control the recognition process for the command-words and to supercede the system supplied internal identification for the command-words.

qualifier = "NOTT" % Not available from a typewriter terminal %

/"NOTD" % Not available from a display terminal %

/"L1" % first level command-word (to be recognized by its first letter) %

/.NUM % explicit value for command-word %

If the user has specified that he wants some (supposedly frequently used) command words recognized based on their first letter and the rest only after typing an escape character, the CML interpreter attempts to accommodate him. The command language designer has control over which command words will be available to such a user via first letter recognition through the L1 qualifier.

Selection Recognition

Three types of selections are built into CML. They are Destination Selection (DSEL), Source Selection (SSEL), and Literal-typein Selection (LSEL).

The literal-typein selection is used to collect literal typein from the user, although it might also allow him to point to text on his display instead of typing it.

A destination selection is used to allow the user to select one of several items the tool has presented to him. This can be done by pointing to it using a pointing device at a display terminal or by typing characters which the tool will interpret. For example, a tool may manipulate textual or graphical representations of data stored in a file. The tool might have a delete command and would use a destination selection to allow the user to specify the line in the drawing or the word in the text to delete. Thus, when the tool put the display image on the screen, it did so using primitives in the Frontend that supplied identifiers for elements of the display [12]. When the user points to an object on the screen, the identifier for it is returned to the tool.

II User Interface

9 Command Meta Language

A source selection is similar to a destination selection but also allows the user to supply the argument as a literal-typein.

Basically, these are recognizers which require some entity type as an argument and they return a data structure which represents the selection. The entity type is obtained either by some previous invocation of the recognition function for some list of command-word entities, or use of the VALUEOF (or #) built in function (see example in Appendix 2).

The DSEL, SSEL, and LSEL functions perform all evaluation and feedback operations associated with the selection operations. The command language designer may define new types of selections and define the data structure that is built as a result of the selection.

selection = ("SSEL"/ "DSEL"/ "LSEL") '(param ');

Command Confirmation

The process of command confirmation is represented in CML by a built-in parameterless function.

confirm = "CONFIRM"; % command confirmation %

simple question answering

The process of simple question answering is represented in CML by a built-in parameterless function.

answer = "ANSWER"; % YES/NO answer to a question
(TRUE if YES) %

Other Recognizers

Other recognizers may be added through the use of parse functions as described below.

FUNCTION EXECUTION

Functions may be invoked at any point in the parse by writing a name of some routine and enclosing a parameter list in parentheses. All functions invoked by the interpreter must obey the ground rules set up for interpreter routines. The actual arguments are passed by address, rather than value, and two additional actual arguments are appended to the head of the argument list.

control = .ID % routine name % '(\$ < ', > param ');

param = factor % expression element %

/ "VALUEOF" '(.SR) % command-word value %

/ '# .SR % same as VALUEOF %

/ "TRUE" % boolean TRUE value (one) %

/ "FALSE" % boolean FALSE value (zero) %

/ "NULL"; % null pointer value (zero) %

II User Interface

9 Command Meta Language

PARSING FUNCTIONS

Functions which are declared with the PARSEFUNCTION attribute are assumed to be parsing functions. They are called in "parsehelp" mode (described below) and when so called, are passed the address of a string as a third argument. The parsefunction routine then supplies a prompt string which tells what the parsing function does. In addition, the parse function should, in a like manner, be prepared to generate a more verbose help string to be used when the user asks to see his current alternatives and a terse syntax string for when the user asks for the syntax of a command.

FEEDBACK CONTROL

The feedback control elements of CML are used to provide feedback in addition to the normal feedback generated by the recognizers. This is used to implement additional "noise words" and help feedback.

1) adding feedback to the command feedback.

A string may be added to the current command feedback by enclosing the quoted string in angle brackets.

```
extra feedback = '< .SR '>;
```

2) replacing the last string in the command feedback.

If the user's terminal allows, it is possible to replace the last string in the command feedback line by using the string replace facility. This is similar to (1) above except the previous string in the command feedback is deleted before adding the new string.

```
replace extra feedback = '<"..." .SR '>;
```

A function is also provided to initialize the command feedback mechanisms and clear the command feedback area.

```
clear feedback = "CLEAR";
```

EXPRESSION DEFINITION

CML is an expression language. Commands are defined to be a single expression and expressions are composed of successive/alternative expression factors. Alternative paths are indicated by the character '/' in the expression.

The nesting of expressions may be explicitly defined with parentheses, and brackets are used to delimit optional expression elements.

```
exp      = # <'> alternative;
```

```
alternative = #factor;
```

```
factor     = terminal/ '[ exp ' ] / '( exp ' );
```

```
terminal   = subname %id/ assign/ function%  
            / confirm %command confirmation%  
            / feedback %noise word feedback%
```

II User Interface

9 Command Meta Language

/ recognition %built-in recognizers%
 / loop; %looping facility%

The looping facility permits repetition of a parse rule until an exit condition is met.

loop = "PERFORM" .ID "UNTIL" '(exp);

The .ID following the command-word PERFORM is a name of a parsing rule which is to be repeated. This rule is evaluated and then the expression following the UNTIL command-word is evaluated. If the expression returns TRUE, then the loop is exited and the next factor in the rule is evaluated. If the expression returns FALSE, then the parse is backed up to the head of the PERFORM, and the named rule is invoked once again.

COMPLETE FORMAL SYNTAX OF CML

file = "FILE" .ID \$dcls \$rule
 subsys "FINISH";

subsys = "SUBSYSTEM" .ID % subsystem handle %
 "KEYWORD" .SR % recognition name %
 #(command / rule) "END";

command = ("COMMAND" / "INITIALIZATION" / "TERMINATION"
 / "REENTRY") rule;

rule = .ID '= exp ';

dcls = ("DCL" / "DECLARE")
 ("VARIABLE" / "FUNCTION" / "PARSEFUNCTION"
 / "EXTERNAL") # <'> .ID / "EXT-KEYWORD" # <'> .SR);

exp = # <'> alternative;

alternative = #factor;

factor = terminal/ '(exp)/ '[exp];

terminal = subname/ confirm/ answer/ feedback/ recognition/ loop;

subname = .ID ['- param/ '(\$ -',> param ');

confirm = "CONFIRM";

answer = "ANSWER";

recognition = command-word/ builtinrec;

command-word = .SR [' ! #qualifier '];

qualifier = "NOTT"/ "NOTD"/ "LI"/ .NUM;

builtinrec = (("SSEL"/ "DSEL"/ "LSEL") '(param));

feedback = "CLEAR"/ '< ["..."] .SR '>;

II User Interface

9 Command Meta Language

```
control      = .ID '($<',>param ');
param        = factor/ ("VALUEOF" '(' .SR ') / '# .SR)
              /"TRUE"/ "FALSE"/ "NULL";
loop         = "PERFORM" .ID "UNTIL" '(' exp ');
```

THE OBJECT CODE PRODUCED BY THE CML COMPILER -- THE GRAMMAR

Each instruction of the object code consists of the following fields: OPCODE, SUCCESSOR, ALTERNATIVE, ADDR, CTL, and VAL.

The ALTERNATIVE and SUCCESSOR fields

These contain the addresses of an alternative instruction to execute in parallel with the current one and the address of the instruction to execute should this one succeed. Null paths are indicated by 0 valued pointers.

The OPCODE, ADDR, CTL, and VAL fields

OPCODE is an operation code. CTL contains control bits used by the interpreter (reflecting the NOTD, NOTT, and LI qualifiers). VAL contains an integer token or zero. ADDR is the address or principal value for the function.

Possible OPCODES

RECOGNIZERS

KEYGP -- command-word recognition.

CTL = control bits for level 1 commands, Display commands, and TNLS commands.

ADDR = address of command-word literal string

The current input text is matched against the command-word string specified by the current node and all alternatives of the current node. This function performs command-word recognition on all of the alternative nodes of the current node simultaneously.

This function cannot fail. Control remains in the command-word recognition function until appropriate input is recognized or until the control is abnormally wrested via backup or command delete functions.

The value returned in the argument record is a single word containing the address of the string corresponding to the command-word actually recognized.

CONFIRM -- process command confirmation characters

This function interrogates the input text for one of the command confirmation characters. Control remains in this routine until a proper confirmation is recognized, and command termination state is appropriately set. This function always returns TRUE.

The value returned is a single word containing a command completion code which identifies the completion mode.

ANSWER -- process yes/no question answer

II User Interface

9 Command Meta Language

This function interrogates the input text for one of the yes/no question answer characters. Control remains in this routine until a proper response is recognized.

SSEL -- get a source selection

ADDR = not used

The sselect routine is invoked to process a source type selection. The return record generally contains two text pointers which delimit the selected entity.

DSEL -- get a destination selection

ADDR = not used

The dselect routine is invoked to process a destination type selection. The return record generally contains two text pointers which delimit the selected entity.

LSEL -- get a literal selection

ADDR = not used

The lselect routine is invoked to process a literal type selection. The selection type is passed as an actual argument. The return record generally contains two text pointers which delimit the selected entity.

VIEWSPECS -- process viewspecs information

The viewspec input routine is called to process the input stream for viewspec characters. The return record contains the two updated viewspec control words. This function always returns TRUE.

LEVADJ -- process level adjust information

The level adjust input routine is called to process the input stream for level adjust characters. The return record contains a single word which indicates the relative level adjust value (u = +1, d = -1, etc.). This function always returns TRUE.

CONTROL FUNCTIONS

EXECUTE -- transfer of control to another point in the tree.

ADDR = address of root of tree for transfer of control

The current point in the tree is marked and control is transferred to the node pointed to by the address field. Control remains in the descendant node until it has been completely parsed, at which time control returns to the successor of the EXECUTE node.

CALL -- subroutine invocation

VAL = number of actual parameters

ADDR = address of the subroutine

The appropriate number of actual arguments are popped off of the evaluation stack and passed to the routine whose address is contained in ADDR.

The result from this routine is pushed onto the eval stack if it returns TRUE.

II User Interface

9 Command Meta Language

PFCALL -- parsing function invocation

VAL = number of actual parameters

ADDR = address of the subroutine

The appropriate number of actual arguments are popped off of the evaluation stack and passed to the routine whose address is contained in **ADDR**.

The result from this routine is pushed onto the eval stack if it returns **TRUE**.

This function is also called in "parsehelp" mode to find out what it does.

OPTION -- test for an optional construct.

If the next input character is the **OPTION** select character, then it is read and control is transferred to the node at address **ADDR**. If the next character is not the **OPTION** character, then control passes to the successor path of the current node.

FEEDBACK ELEMENTS

FBCLEAR -- clear the contents of the feedback buffers.

The feedback state information and command feedback line are set to their initial or empty position.

ECHO -- appends a noise-word string to the command feedback link

ADDR = address of the text string to be appended.

RECHO -- replaces the last noise-word string in the command feedback line

ADDR = address of the text string which is to replace the last item in the command feedback buffer

VALUE MANIPULATIONS

LOAD -- loads a pointer to an argument record into the top of the eval stack.

ADDR = address of the variable containing the pointer to the argument record.

The pointer value contained in the variable whose address is contained in **ADDR** is pushed onto the top of the eval stack.

STORE -- saves a pointer to an argument record in a variable

ADDR -- address of the variable

The address of an argument record is fetched from the top of the eval stack and is saved in the variable at address **ADDR**.

ENTER -- enters a constant value into the argument record pointed to by the top of the eval stack.

ADDR -- value to be entered (18 BITS only)

II User Interface

9 Command Meta Language

The value is taken from the ADDR field of the instruction and is entered into the argument record for the ENTER node in the path stack (whose address is at the top of the eval stack).

PROBLEMS WE HAVE ENCOUNTERED WITH CML

The principal problem we have encountered is that some of the recognizers (command-word recognition, command confirmation, LSEL, SSEL, DSEL, and so forth) cannot fail. This is purely an implementation decision that was made regarding the CML Interpreter and, consequently, does not impact the language itself. In addition, the CML Interpreter was implemented as a stack machine and would better serve our needs as a machine with an accumulator and an argument stack.

Also, the manner in which the user-input prompt, the current alternatives, and the syntax of commands is generated should be more standardized to avoid some of the problems and anomalies we have encountered to date. These problems have chiefly been caused by the knowledge the interpreter has of some functions and lack of knowledge about others.

In order to serve the needs of a wider range of tools (application programs) we feel that the declaration facility of CML should be expanded to allow the command language designer to define how to handle many special things such as collection of parameters of a form specific to the tool. In addition, we would like to make the CML Interpreter system available through an interface [6] that does not require the tool to be written in the same language as the Interpreter.

USING THE CML SYSTEM

WRITING CML PROGRAMS

Source programs for the CML compiler are free form NLS or TENEX sequential files. Comments may be used wherever a blank is permitted and the structural nesting of the source file is ignored by the compiler.

COMPILING CML PROGRAMS

CML source programs are compiled into REL files with the Compile File command in the PROGRAMS subsystem. CML is the compiler name for the CML compiler.

RUNNING CML PROGRAMS

A complete interactive subsystem usually consists of three distinct parts: (1) The syntactic description for the subsystem command language. (2) The parser interface routines ("X" level parsing support routines). (3) Core execution functions.

If a CML subsystem is to be run as a user program, then the rel-files for the syntax, parsing support, and execution functions are loaded into the user programs buffer with the Load Program command.

After loading the rel-files the user's subsystem is connected to the set of available subsystems with the Attach Subsystem command. The name specified in this command is the name of handle for the subsystem (the .ID appearing on the SUBSYSTEM statement of the CML program).

II User Interface

9 Command Meta Language

The user's subsystem may then be invoked by using the GOTO command, as the system will now know about the new subsystem.

FUNCTION INTERFACE PROTOCOL

The syntax of the function call in the CML meta-language is similar to that of most programming languages: the name of the function is followed by a list of expressions enclosed in parentheses. In the CML system however, there are some strict rules which apply to all execution functions invoked by the interpreter. These rules are enumerated below:

1) Additional actual arguments

Preceding any actual arguments which appear in a function reference in CML, the interpreter supplies two additional actual arguments. These are:

- 1) a pointer to the "function state record"
- 2) an integer which defines a parsing mode
 - = parsing: normal execution mode
 - = backup: backup after a FALSE path is taken
 - = cleanup: resetting of state after completion of command
 - = parsehelp: policing prompts string (parse functions only)

These additional arguments must be used by all execution functions to determine what they are to do. The pointer to the "function state record" is used to return values from the function and to save state information associated with a particular invocation of the function. The length of the function state record is 10 words and this record may be formatted in any manner appropriate to the function.

If 10 words is not sufficient space to record all of the state associated with a particular invocation of a function, then the function must use a storage allocator to allocate the additional storage and record the handles to the allocated storage in the function state record. Note that if this additional "local state" storage is required, then it is the responsibility of the execution function to de-allocate the local state storage when called in backup or cleanup modes.

2) Returning parse failure

All execution functions are passed a pointer to their function state record. If the function processes normally, then it returns the same pointer as its only return value. If the function decides that the parse should fail at a given point, then it returns FALSE.

3) Passing arguments by address

All of the actual arguments in a function call on an execution function are passed by address rather than by value. The values actually passed are pointers to the function state records corresponding to the actual arguments. The format of the function state records are defined by the execution functions which manipulated them, and thus the location of parameter values in these records is determined by convention, the caller and callee having previously agreed to a particular layout for the function state record.

II User Interface

9 Command Meta Language

4) Order of control

An execution function will always be called in parsing mode before it is called in backup or cleanup modes.

A function routine which saves state information in the function state record must initialize its state record to some consistent state before it calls any subroutines which may cause SIGNALS or otherwise cause control to abnormally pass above the execution function.

FLOW OF CONTROL IN THE INTERPRETER

At any point in the process of parsing, the control pointer for the interpreter points to a structure word in the grammar. A path stack also exists which shows the nodes from which TRUE returns have been achieved. Some operations mark the path stack for halting the backup process. The parser has four distinct control states defined as follows:

- 1) parsing: recognition state where input text is compared with grammatical constructs to determine the parsing path in the parse tree.
- 2) backup: A FALSE return has been obtained from some execution/recognition function. The path stack is backed up until a non-NULL alternative path is found, at which time the parse mode is set to parsing, and recognition of the alternative path is attempted. If no non-NULL alternative path is found, then the parse fails and the interpreter returns FALSE.
- 3) cleanup: A terminal parse has been achieved and control is passed to each execution routine to reset any state informations set by the routine.
- 4) parsehelp: (used only with parsefunctions) Before calling a parsefunction in "parsing" mode, the function is called in "parsehelp" mode to solicit a user prompt string.

The general flow of control is:

- 1) An initial path stack entry is constructed, and the parse mode is set to parsing. The execution function for the current node is evaluated. A pointer to the "function state record" is passed to the routine. The state record contains the return values for the function as well as a record of any state information saved by the function (for backup purposes).
- 2) A prompt string is generated for the user indicating in a terse fashion what his current alternatives are. If he wishes expansion on this he may ask for his current alternatives or for the syntax of the rest of the command.
- 3) If the function returns TRUE, then the successor to the current node becomes the current node. If this is NULL, then the PTRSTK stack is backed up until a non-NULL successor path is found. If none is located before the bottom of the current parse state is reached, then the root of a parse tree has been reached, and a command has been successfully executed. In this case the command reset operation is performed and the interpreter is set to "parsing" mode once more.
- 4) If the function returns FALSE then the parser mode is set to "backup" and a non-NULL alternative path is sought.

II User Interface

9 Command Meta Language

After a command has been executed, the parsing path for the tree is re-evaluated in "reverse order" beginning with the terminal node of the path. Each execution function is re-invoked, in "cleanup" mode, and is passed the handle for the state information record which it generated on the forward pass through the grammar. Each execution routine has the responsibility of resetting any state information which it wishes to do at the termination of a command. Cleanup continues until a "starting point" is reached in the parse. This is generally the beginning of the command. At this point, the interpreter "shifts gears" and goes into forward or recognition mode and begins back down the grammar for the language.

The same backup mechanism is also used during command specification in order to back up the parse to allow the respecification of all or part of the command. The command delete function backs out of the parse tree until the beginning of the command is reached.

The same backup mechanism may be adapted to control the partial backup required for executing commands in "repeat mode" where at least one of the alternatives is defaulted to its current value.

SAMPLE CML PROGRAM

FILE nlsexample

SUBSYSTEM nlseditor KEYWORD "BASE"

% COMMON RULES %

% PARAMETER TYPE DEFINITIONS %

editentity = textent / structure;

% TEXT PARAMETER TYPE DEFINITIONS %

textent =

"CHARACTER" / "WORD" / "VISIBLE" / "INVISIBLE" / "TEXT" / "LINK" /
"NUMBER";

% STRUCTURE PARAMETER TYPE DEFINITIONS %

structure = "STATEMENT" / "GROUP" / "BRANCH" / "PLEX";

COMMAND %replace%

zreplace =

"REPLACE"

type ← editentity

% The rule EDITENTITY defined above is evaluated. The one chosen (via user input) is stored in the variable TYPE. %

<"at"> destination ← DSEL(type)

% The user is presented the noise word "at" and requested to supply a destination of the

II User Interface

9 Command Meta Language

type chosen from EDITENTITY. The user must then identify the item to be replaced. The representation of this item is stored in the variable DESTINATION. %

<"by"> source ← LSEL(type)

% The replacement is collected from the user and stored in the variable SOURCE. %

CONFIRM

% Have the user confirm that he wants the replacement to take place as specified. %

xreplace(type, destination, source);

% call the primitive in the application program that performs replacements. Pass it the type of thing to replace, the specific instance of that type to be replaced, and the replacement. %

COMMAND %load%

zload =

"LOAD"

type ← ("FILE"/"PROGRAM")

% this command allows user's structured text files and programs to be loaded into NLS for user manipulation and execution, respectively. %

filename ← LSEL("#"OLDFILENAME") CONFIRM

% Collect the name of an old file from the user. The file may be the one to load or it may contain the program to be link-loaded. %

xload(type, filename);

% pass the application program's load primitive the type of load and the file name. %

COMMAND %interrogate user to help him send mail to other users%

interrogatecmd =

"INTERROGATE"

CONFIRM

% User wants to be interrogated for needed info to send mail to other users. %

CLEAR <"distribute for action to:">

content ← LSEL("#IDENTLIST")

setfield("#ACTION", content)

% CLEAR causes Carriage Return Line Feed on a typewriter-like terminal and causes the command area to be cleared on a display. The application function setfield is called to set the "action" field in the current message header to the list of user-recipients supplied by the user and stored in CONTENT. %

CLEAR <"distribute for information-only to:">

II User Interface

9 Command Meta Language

```

content ← LSEL("#IDENTLIST")
setfield("#INFORMATION", content)
CLEAR <"title:"> content ← LSEL("#TEXT")
setfield("#TITLE", content)
CLEAR <"type of source:">
(
  "MESSAGE" type ← "#STATEMENT"
  content ← LSEL("#TEXT")
  % Message is the same as statement. %
  / type ← "FILE"
  content ← DSEL("#CHARACTER")
  % The user may specify any character in the file. %
  / type ← structure
  <"at"> content ← SSEL(param)
  % Since this is an SSEL, the user may type it or specify its location in one of his files. %
  / type ← "OFFLINE" <"document">
  <"located at"> content ← LSEL("#TEXT")
  % If it is an offline hardcopy document, simply have the user describe where it is being
  stored. %
)
setfield(type, content)
CLEAR <"show status?"> (ANSWER showstatus() / DUMMY)
% If the user answers "YES", call SHOWSTATUS to present the current specification of
the mail to the user. %
CLEAR <"send the mail now?"> (ANSWER xdoit() / DUMMY);
% If the user answers "YES", call XDOIT to send the mail as specified, otherwise simply
let him use other commands to change the specifications and send it. %
END.
FINISH

```

II User Interface

9 Command Meta Language

SAMPLE INTERPRETER PARSEFUNCTION ROUTINE

Assume that in some command we want the typein of a number to appear as an alternative of some set of command-words. We can accomplish this by defining a parsefunction (call it looknum) which looks at the next input character and succeeds if the next character is a digit and fails otherwise. If we write this function as the first alternative in some command, then control will pass from the interpreter to the parsefunction before it passes to the command-word interpreter.

Suppose our command looks like:

COMMAND sample =

"INSERT"

% determine the type of insert %

(looknum() < "number" > type ← # "NUMBER"

/ type ← ("TEXT" / "LINK"))

% the variable TYPE now contains NUMBER, TEXT, or LINK. We now use the LSEL function to get a selection of this type and store it in the variable SOURCE %

source ← LSEL(type)

% p-, a command confirmation to make sure user wants this done %

CONFIRM

% now invoke the insert execution function passing as arguments the entity type and the selection of that type %

xinsert(type, source);

Now take a look at the parsefunction looknum which is called by the interpreter both when prompting the user and also during the actual parse of the command.

% LOOK FOR A NUMBER %

(looknum) PROC(

% looknum looks at the next input character, if it is a digit, then a true return is taken else FALSE is returned %

% FORMAL ARGUMENTS %

resultptr, % ptr to the function state record %

parsemode, % parsing mode for the interpreter %

string); % ptr to prompting string %

REF resultptr, string;

%-----%

CASE parsemode OF

II User Interface

9 Command Meta Language

```

= parsing:
CASE lookc() OF
IN ['0', '9]:
NULL;
ENDCASE RETURN (FALSE);
= parsehelp:
*string* ← "NUM: ";
ENDCASE;
RETURN (&resultptr);
END.

```

REFERENCES

- [1] (9a1a) Douglas C. Engelbart. Design Considerations for Knowledge Workshop Terminals. In AFIPS Proceedings, Vol. 42, 1973 National Computer Conference, pp. 221-227, 1973. (14851,)
- [2] (9a1a) (9a1e) Douglas C. Engelbart, Richard W. Watson, James C. Norton. The Augmented Knowledge Workshop. In AFIPS Proceedings, Vol. 42, National Computer Conference, pp. 9-21, 1973. (14724,)
- [3] (9a1a) SRI-ARC. Online Team Environment / Network Information Center and Computer Augmented Team Interaction. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 6-MAR-73. (13041,)
- [4] (9a1a) Susan R. Lee. A Review of the SUPARS Report. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 12-NOV-73. (20182,)
- [5] (9a1a) Douglas C. Engelbart. Human Intellect Augmentation Techniques. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. JUL-68. (3562,)
- [6] (9a1b56) (9a1k) (9c3) James E. White. Version 2 of the Procedure Call Protocol (PCP). Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. PCP-COVER.NLS:5,. (24590,)
- [7] (9a1p) Charles H. Irby. Tree Meta References. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 7-MAY-73. (16310,)
- [8] (9a1p) Marilyn F. Auerbach. Notice of Online Tree Meta Reports. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 23-JAN-73. (14052,)
- [9] (9a1p) (9a2a) SRI-ARC. Tree Meta Report -- Preliminary Draft. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 23-JAN-73. (14045,)

II User Interface

9 Command Meta Language

- [10] (9a1p) (9a2a) SRI-ARC. Tree Meta Report -- Preliminary Report, Formal Description. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 23-JAN-73. (14046,)
- [11] (9a2a) Charles H. Irby. Display Techniques for Interactive Text Manipulation. In Proceedings of the National Computer Conference, 1974, p. 247-255. (20183,)
- [12] (9a4c2a2) Douglas C. Engelbart. Coordinated Information Services for a Discipline- Or Mission-Oriented Community. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 12-DEC-72. Paper given at Second Annual Computer Communications Conference, San Jose, California, 24 January, 1973. 13p. (12445,)
- [13] Douglas C. Engelbart, William K. English, A Research Center for Augmenting Human Intellect. In AFIPS Proceedings, 1968 Fall Joint Computer Conference, Vol. 33, Part I, p. 395-410, 1968. (3954,)

First Studies of NLS Command Use and Timing (by Susan R Lee)

INTRODUCTION

Two special studies have been made to determine the efficiency of the system and how it is being used. Studies of system efficiency have been made on several levels, including the timesharing system as a whole, NLS as a text editor, and execution times for individual commands.

Various methodologies have been utilized to achieve these goals. Programs have been written to collect and reduce data in a systematic way, and other available means for collecting data have been used. A description of methods used for some typical studies follows.

SUPERWATCH AND SYSTEM USAGE

Superwatch is a program designed to record statistics on such things as the usage of the drum and disk, paging overhead, load on the system, number of users, percent of system used by individual users, and much more. See (20b) for a more detailed description of this program.

Superwatch has been used both for measuring overall system performance and for daily operational control, both at ARC and the Utility.

After correlating load average with number of users, one conclusion reached was that with the existing computer configuration, the system could reasonably serve a maximum of twenty users at any one time. This conclusion helped set the maximum number of users in the group allocation scheme (16). After further analysis of use by users, number of users, and individual use patterns, it was determined that the average DNLS user used 4% of the system, while the average TNLS user used 2% of the system. These facts led to the conclusion that the system could support 10 DNLS users and 10 TNLS users, or 15 DNLS users, or 30 TNLS users, or some other combination allocating 60% of the machine to users (approximately 40% is needed for overhead functions and idle time as shown in the tables below).

In order to arrive at conclusions such as the ones described above, there is a requirement for data collection of the type done by Superwatch. The following tables give samples of the output from Superwatch showing the distribution of CPU time for May, 1974.

Overhead	32.5%
Percent used	56.8%
Idle	10.7%

Total	100.0%

Overhead and percent used (%U) or percent of CPU used by users may be further broken down as follows.

Preceding page blank

II User Interface

10 Command Use

I/O wait	16.0%
Scheduling	6.6%
Process clocks	4.8%
Garbage collection	2.7%
Network overhead	2.4%

Total Overhead	32.5%
----------------	-------

10b1b1

NLS	46.9%
Hardcopy output	11.9%
Journal	2.5%
Network	8.9%
Support	20.8%
Systems development	6.7%
Fortran development	2.3%

Total Percent Used	100.0%
--------------------	--------

10b4b1

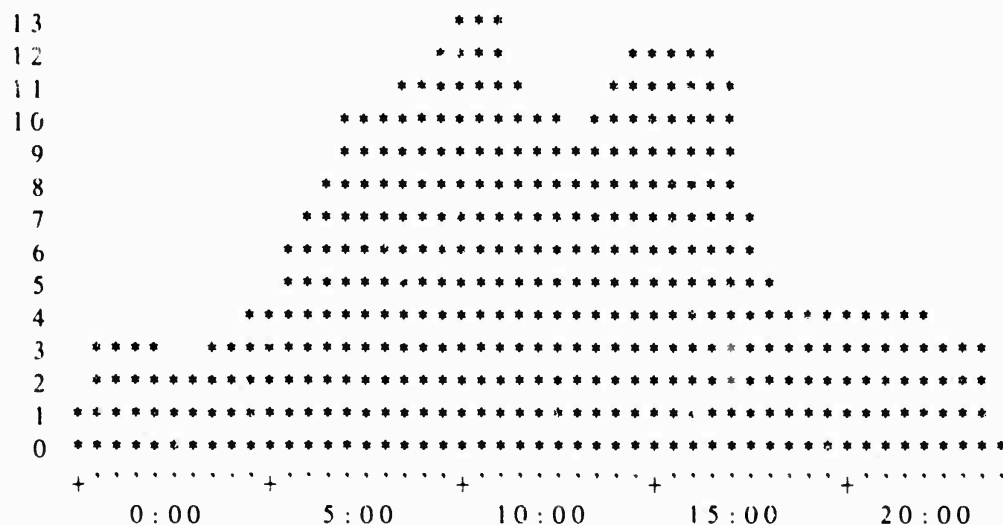
Graphs showing such things as amount of time system is idle, load average, number of users, and percent of system used by users have facilitated the rescheduling of jobs such as catalog production, and have allowed users to pick times to work when the system is not loaded so as to maximize the use of their time. An example follows of a graph showing average variation of number of users over a 24-hour day.

10b5

TIME PLOT OF AVERAGE NUMBER OF USERS FOR MAY 1974

x axis labeled in units of hr:min, x unit = 30 minutes

10bfa1



11 User Interface

10 Command Use

Projections of the effect of changes in the system are possible through the use of Superwatch. For example, during the later part of 1973, the system was heavily loaded and local users were encouraging a change in system configuration or allocated slots to improve service. A check of network usage indicated that 25% of the CPU allocated to users went to network users who would soon depart for the Utility. This fact satisfied local users.

COMMAND FREQUENCY COLLECTION AND NLS USAGE

Summary of Results

In order to improve the efficiency of NLS commands that were used most often, a program was used to collect data on command frequency. Statistics were collected from all users of DNLS for one work-week. A compilation of the results showed that a total of 10,814 commands were issued during the five day period, accounting for 134 different commands.

It should be noted that the collection was done on a macro level, i.e. only the top level commands were recorded. Commands within subsystems or executed via the repeat command feature were not recorded. The data, however, give an overview of usage.

Jump commands accounted for 40% of all commands and text editing commands accounted for 34%. Twenty commands accounted for 75% of all used.

Early studies of command timing led us to the conclusion that to be cost effective, commands would have to be made more efficient. For example, 10 msec were required to insert one character in TNLS and 30 msec were required to insert one character in DNLS. The list of most used commands was used as the basis for choosing commands to concentrate on, and after attempts to improve the Insert Character command and other high frequency commands, 3 msec were required to insert one character in TNLS and 1 msec were required to insert one character in DNLS.

Examples of Data

The most frequently used commands were ordered to produce a list of 20 commands that account for approximately 75% of all commands used. The tables on the following page show the complete list, and a list of commands by functional category.

II User Interface

10 Command Use

Command	Frequency	Percentage	Cumulative
Jump Item	140 ¹	13	13
Insert Character	562	6	19
Jump Link	495	5	24
Insert Statement	492	5	29
Jump Jump*	425	4	24
Jump Successor	352	4	37
Replace Word	350	4	41
Delete Character	316	3	44
Replace Text	296	3	47
Replace Character	282	3	50
Update New	277	3	53
Jump Origin	254	3	56
Jump Content	240	3	59
Jump File	225	2	61
Goto Display	220	2	63
Delete Text	199	2	65
Delete Statement	196	2	67
Update Old	195	2	69
Load File	189	2	71
Jump Return	189	2	73

*The frequency of a user typing "JJ" indicates the number of times a user was in the Jump mode and retyped "Jump" unnecessarily. "JJs" account for 10% of the Jump commands.

Commands were divided into 13 functional categories to further differentiate usage patterns.

Category	Number	Percentage
Jumping	4205	39
Text Editing	3642	34
File Manipulation	853	8
Text Creation	836	8
Structure Editing	333	3
Viewing	279	2
Other	666	6
Total	10814	100%

II User Interface

10 Command Use

JOURNAL USAGE STATISTICS

The Journal is an integral part of the NLS system, and it was therefore felt that the Analysis group should gather data on the Journal to better understand its usage.

Throughout 1972 and 1973, Journal usage increased steadily until mid-1973 when the number of items sent each month reached about 450 per month. One-third of these items originated at a site on the ARPANET other than ARC. A table and graph detailing this growth are shown on the following page.

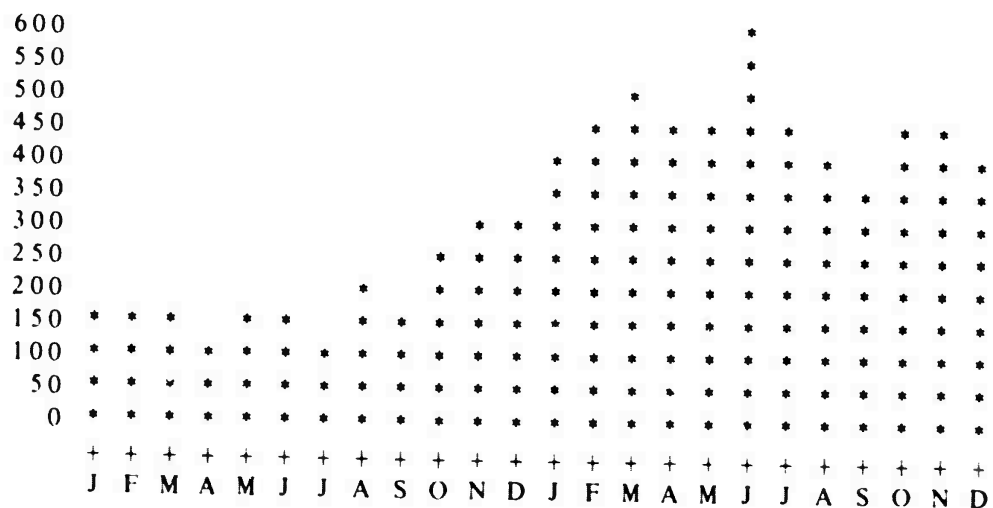
II User Interface

10 Command Use

Monthly tabulation of Journal items sent in 1972 and 1973.

	1972	1973
January	127	404
February	156	432
March	131	505
April	118	474
May	174	437
June	129	608
July	105	461
August	185	405
September	164	355
October	251	448
November	297	473
December	315	385
Total	2152	5387

Graph of Journal usage for 1972 and 1973.



II User Interface

10 Command Use

The majority of items are distributed to a relatively small number of people. Studies show that 40% of all items were sent to one person, and 70% were sent to five or fewer. Distribution patterns are detailed below.

Type of Distribution	Percent	Cumulative Percent
Individual idents		
0 individuals*	5%	5%
1 individual	40%	45%
2 individuals	10%	55%
3 individuals	6%	61%
4 individuals	5%	66%
5 individuals	4%	70%
6-10 individuals	6%	76%
More than 10 individuals	3%	79%
Group idents		
SRI-ARC	12%	12%
One group	6%	18%
One group plus 1-5 individuals	3%	21%
TOTAL	100%	
*(test or for the record)		

The ability to look at the number of times a Journal item has been accessed indicates that files distributed to a group of 15 or more people are accessed in the following manner (accessed implies the text of the item was either read or glanced at):

Day 1 - 9 accesses
 Day 2 - 7
 Day 3 - 3
 Day 4 - 1
 Day 5 - 1
 Day 6 - 1
 Day 7 to Day 20 - 0

Generally, 40% of all access occurs the first day, and 90% has occurred by the end of the fourth day.

Chapter III: NLS SUBSYSTEMS

The Calculator (by Elizabeth K Michael)

BACKGROUND

Old Calculator

A calculator system was implemented in the XDS 940 version of NLS which had two primary operating modes: regular desk calculator mode (add, subtract, multiply, and divide) and function mode (with named variables, etc).

Although some effort was devoted toward rewriting the Calculator in L10 for conversion to the DP 10, this version was never implemented on the PDP-10. The strategy of the current calculator design was to take as many useful features as possible from prior work while adding those new features that provide additional usefulness and power.

New Environment

Extensions to NLS added since the XDS-940 version made it possible to give the new calculator features that were not available in the older version. The most important of these was the split screen capability. This feature allowed the user to display one or more files to be used as targets for results of calculations or to obtain input for calculations. In addition the user could display simultaneously a file in which every calculation was recorded.

Increased power of both TNLS and DNLS addressing made the extraction and insertion of numerical data from and into files much more flexible.

DESIGN SPECIFICATIONS FOR L10 ENVIRONMENT

Introduction

The Calculator is planned to evolve in three successive major phases. The Phase 1 Calculator was designed to provide the capabilities of a relatively simple desk calculator in combination with the file editing features of NLS.

Design of Phase 2 (a compiler capability) and Phase 3 (graphics capability) was not completed. These are described briefly under Planned Extensions.

What it Does

The Calculator is a self-contained subsystem of NLS. It adds, subtracts, multiplies, and divides like a desk calculator. Operands may be selected from NLS files using standard NLS addressing or selection modes, or they may be entered from the keyboard either directly as numbers or indirectly as simple arithmetic expressions (e.g., 2x4.6/5.123). Results may be easily stored in any NLS file. The user may leave the Calculator, use other NLS commands,

III NLS Subsystems

11 The Calculator

and then return to the Calculator and continue work there, just as though his original session had not been interrupted. It is this close integration of numeric work with NLS textual capabilities that is the unique and powerful innovation of the NLS Calculator. 11479

Each operation is recorded in a special Calculator file, created for the user by the system. The results may be stored both in the special Calculator file and in other user files. The user may request various number formats (e.g., have numbers edited with commas, preceded by a dollar sign), keep as many as ten running totals in separate accumulators, and enter arithmetic expressions as if they were numbers. 11485

When the user leaves the Calculator, his accumulator values and number format specifications are saved for him in the Calculator file. When he next enters the Calculator he is given the option of beginning exactly where he left off before or starting with all accumulators set to zero. 11490

Arithmetic operations are performed on the value in the current accumulator. An arithmetic operation requires as input an optional operator followed by a signed or unsigned number or an accumulator designation. The number may be entered directly from the keyboard, indirectly as an arithmetic expression, selected by typing an address expression, or, in DNLS, selected by the cursor (Bug) from any of the displayed files. 11495

There are commands in the system that allow the user to change the current (active) accumulator, display the values in all accumulators, clear the accumulators, change the format of stored numbers, store values in files, and copy the special Calculator file. 11500

Planned Extensions 11505

Planned enhancements to the Phase 1 Calculator included adding a broad range of mathematical functions: exponentiation, root extraction, and trigonometric functions. Most of the basic mathematical routines to accomplish this have been written and tested. Only the user interface routines remain to be done. 11510

The Phase 2 Calculator was to provide a compiler with a user-available algebraic language that would allow the user to evaluate functions and write complex interactive mathematical programs. 11515

The plan was to interface NLS, through the Calculator subsystem, to a compiler (initially Basic) already available on the PDP-10. The task consisted of the following areas: 11520

- 1) Transferring control between NLS and the compiler subsystem. 11525
- 2) Passing control information to the compiler subsystem. This is concerned with passing commands to the subsystem as to what action is to be taken, where to get input, where to place output, etc. 11530
- 3) Transferring data between NLS and the compiler subsystem. This involves taking data in internal NLS format and converting it to a meaningful format for the compiler and accepting compiler output and converting it to NLS format. 11535
- 4) Providing a smooth and simple user interface to accomplish all of the above. 11540

III NLS Subsystems

11 The Calculator

These problems exist in relation to any other subsystem and NLS, and it was decided that the best solution would be to wait for BBN's implementation of stream files in TENEX. The proposed feature would provide a communication system responsible for moving data between user processes, the file system, various devices, etc. : b b d

The Phase 3 Calculator was to provide a graphics capability and was to be coordinated with future plans for more general graphics features for non-calculator, NLS use. : b b e

IMPLEMENTATION : :

The Phase 1 Calculator was implemented initially under L10. This version was in use for about a year during which time user-reaction was observed. When the Calculator was converted to NLS-8 with CML, a number of the Calculator features and defaults were changed on the basis of these observations. : b b f

The system was used extensively in the preparation of internal ARC measurement, budget and accounting reports, and for similar reports prepared at RADC. : b b g

The Output Processor and Computer Output to Microfilm (by Dirk H van Nouhuys, Elizabeth K Michael, N Dean Meyer)

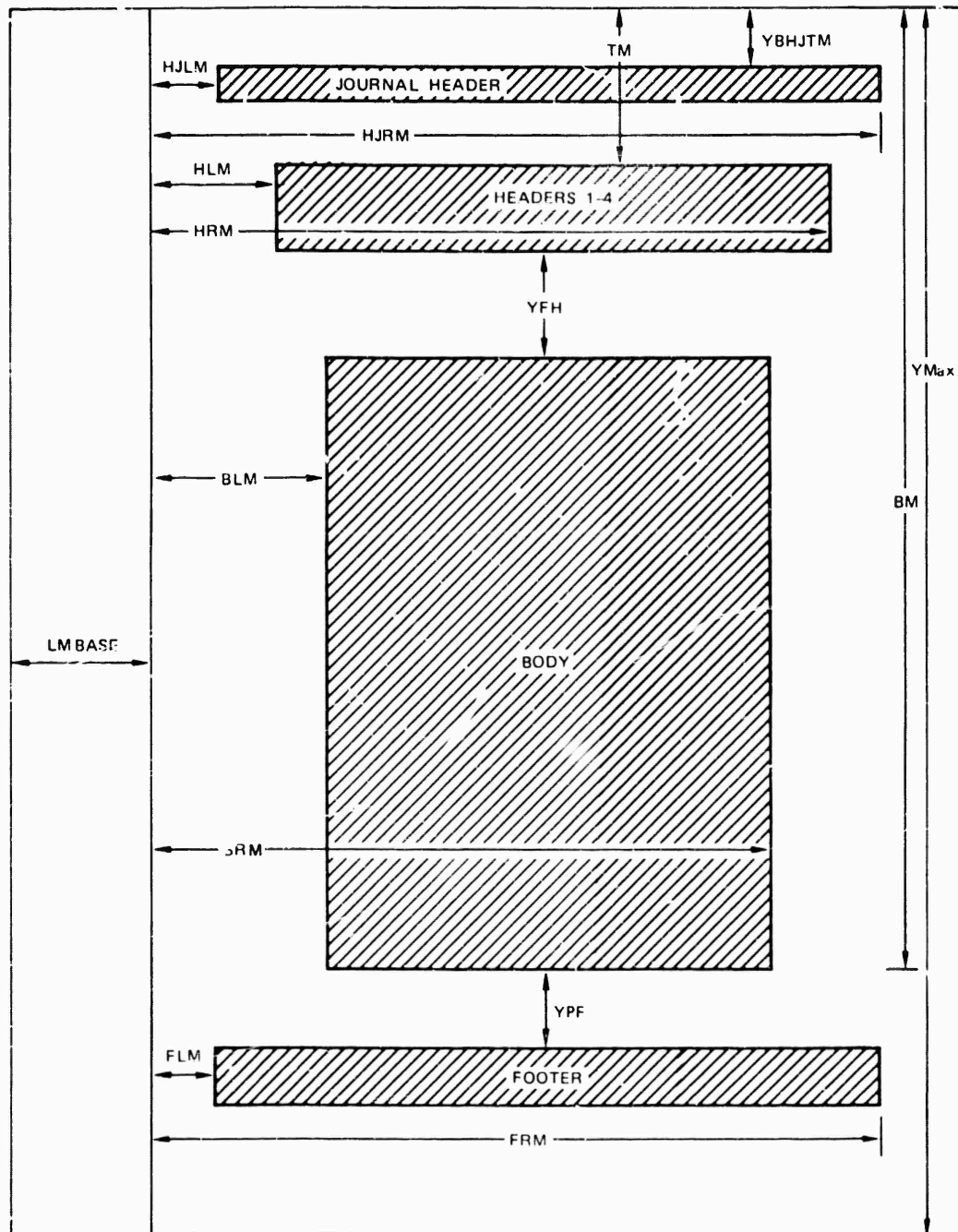
HOW THE OUTPUT PROCESSOR WORKS FROM THE USER'S VIEWPOINT

The NLS Output Processor allows the user to control printed format by inserting brief directives in the text. Such directives normally have the form of a capital letter, some lower case letters, an equals sign, and a number (if required). Figure 1, on the following page, illustrates some of the basic directives and the formatting they control.

III NLS Subsystems

12 Output Processor

Figure 1. Basic Formatting Directives



NOTE: LM sets HLM, BLM, and FLM
 RM sets HRM, BRM, and FRM

SA 3074-1R

III NLS Subsystems

12 Output Processor

Figure 1 illustrates the operation of a few basic instructions. All values in the figure are defaults which the user may change.

- 1) Rm places the right margin, 72 characters to the right. 12a-4
- 2) H creates a header. The user may specify one or more running headers by the instruction H="text". This report has three headers most places. It is possible to set their margins separately as in this report. 12a-5a
- 3) Tm sets the number of blank lines to be left at the top of each page, in this example three. 12a-5b
- 4) Brm sets the right margin of the body of text 65 characters to the right. 12a-5c
- 5) Blm sets the left margin of text 12 spaces to the right. 12a-5d
- 6) Ilev sets the indentation per level of hierarchy. 12a-6
- 7) Lbl puts blank lines between lines. 12a-7a
- 8) Lbs puts blank lines between NLS statments. 12a-7b
- 9) Sn writes statement numbers at the beginning of statements. 12a-8
- 10) Snf writes statement numbers any place in the first line of a statement, by default near the right margin. 12a-9
- 11) Ymax measures out length of the whole page. 12a-10
- 12) BM sets the bottom margin of the body text. 12a-11
- 13) Center centers a given number of lines. 12a-12
- 14) F sets a footer. By default the footer is a page number that users may generate separately with the directive Pn. 12a-13

Other directives control, for example, tabs, position of statements according to level in hierarchy, numbering according to level of hierarchy, unusual types of indentation, and, in the case of COM (Computer Output to Microfilm) printing, type face, type size, columnation, and spacing for illustration. Directives may redefine or suspend other directives. 12a-14

HOW THE OUTPUT PROCESSOR WORKS INTERNALLY 12b

From the viewpoint of NLS system structure, the Output Processor runs as a process under NLS which, whenever the user gives NLS a command that requires its services, is loaded into a transient area. 12b-1

When a user gives NLS an Output Processor command, NLS loads the program and calls the Output Processor's initialization routine. A table of addresses and values, which enables the Output Processor to use NLS routines and to know about things like current viewspec settings and statement levels, is passed to the initialization routine. 12b-2

The Output Processor consists of the following primary elements: 12b-3

A set of tables that contain information about directives, such as minimum and maximum

III NLS Subsystems

12 Output Processor

values, default values, and the names of the appropriate interpreter rules to parse the directives. 12b0a

These tables are 'compiled' by a trivial compiler called DINIT that makes them available to the directive recognizer and executor. 12b0a1

The directive recognizer and executor parses all directives and stores appropriate values in tables and flags that are referred to by the remaining processors. 12b0b

The directive recognizer and executor is written in a language called Tree Meta [1][2] and is compiled by the Meta compiler. 12b0b1

The values in the separately compiled tables identify directives, test their validity and determine which interpreter rules apply to the directive. 12b0b2

The remaining code, written in L10, comprises the initialization and termination routines, and routines that format each statement paying attention to the directive values established by the directive recognizer. 12b0c

The Device dependent code has been carefully restricted to the initialization and post processing routines so that it is relatively straightforward to expand the Output Processor to accommodate new output devices. 12b0c1

The NLS file statements are processed one at a time. Each is scanned for directives and formatted accordingly. Processed statements are stored in a one-dimensional array and written a page at a time on a Printer or COM file. 12b0d

Files to be sent to the line printer are fully formatted for printing. The text of the directive itself is removed from the print image of the statement unless the user specifies otherwise. 12b0d1

The output file format is quite different for a document to be produced using COM. 12b0e

The Output Processor interprets the user's specifications of font, character size, leading, columns, etc.; the COM file it produces includes all the information a virtual COM device needs to place characters on the page. 12b0e1

We have tried to make a simple way for users to produce documents with the format range of most textbooks without concerning themselves with the numerical locations of points on a COM device dot matrix. 12b0e2

This simplified subset has the following characteristics: 12b0e3

The user normally views the COM device as a machine for composing 8 1/2 X 11 inch pages of text.

The user specifies physical dimensions, such as character size and margin settings, in units of one-thousandth inch. The page has a coordinate system in which (0,0) is the upper left corner of the page and (8500, 11000) is the lower right corner. Character size may be optionally specified in points.

Software in the COM device converts character size specifications to "scope points."

The COM device offers eight line widths (CRT spot sizes) and eight levels thus giving 64 possible character densities. It is our feeling that only three densities are actually

III NLS Subsystems

12 Output Processor

distinguishable in a final offset-printed product. Therefore, we offer only light, medium, and bold.

Although the output file is different, the COM format is controlled by the same set of directives for the source file control that controls format for the lineprinter or other non-spaced devices. In the majority of cases the system supplies a default COM value where needed in a directive couched in terms of a monospaced page, e.g., it translates a margin given in centimeters into spaces. When that occurs it is always possible to insert a value for COM that differs from the value for monospace devices. A certain number of directives are meaningful only in the context of COM: those that change type face. In that case, the Output Processor ignores them when producing a file on monospace devices.

The Output Processor produces a sequential file consisting of 7-bit bytes, packed five to a 36-bit word, left-justified (the low-order bit is wasted). This file is then copied to tape in Los Angeles via the ARPA Network for processing on the Comp80.

Text is divided into line segments (an unbroken string of characters in the same font, size, etc.) each of which is preceded by one or more command bytes followed by arguments.

The command bytes are:

- "1" : Output Line Segment
- "2" : New Page (advance frame)
- "3" : Insert Figure (e.g., photograph)
- "4" : End of Document

The Output Line Segment command byte has the following arguments:

- a) y-coordinate of line segment (2 bytes)
- b) x-coordinate of left margin of line segment (2 bytes): This is the x-coordinate (in thousandths of an inch) of the left edge of the leftmost character position in the line segment.
- c) x-coordinate of right margin of line segment (2 bytes): This is the x-coordinate (in thousandths of an inch) of the right edge of the rightmost character position in the line segment.

The x-coordinates actually address the position that lies between two adjacent character positions. In other words, the x-right of one line segment may have the same value as the x-left of the following line segment.

The Justification Code (one byte):

- "0" : Set Line Segment left flush (ragged right)
- "1" : Set Line Segment right flush (ragged left)
- "2" : Center Line Segment between left and right margins
- "3" : Full justification (vary space between words to make both right and left edges of the line segment flush with margins)

III NLS Subsystems

12 Output Processor

Number of text segments in this line segment (one byte): A line segment may contain up to 125 text segments, each of which may be in a different face, size, weight, etc. These text segments are combined into a single line segment to permit consistent justification by the Comp8C.

Each text segment is preceded by the following information:

Type face (one byte)

"0" : Messenger

"1" : Directory

"2" : File

"3" : OCR-B

"4" : MNA Microfont

"5" : News Gothic

"6" : Times Roman

Type Size (two bytes)

The spacing in thousandths of an inch between the base lines of two adjacent lines.

Type Style (one byte where each bit, b1,b2...b7, has a unique meaning.)

Boldness (b6, b7)

"00" : medium (normal)

"01" : light

"10" : bold

Slant (b5): If b5 is on, the characters in this text segment are slanted. Slanting approximates the appearance of italics.

Underlining (b4): If b4 is on, a line is drawn under the text in this segment with a boldness appropriate to that of the text.

Monospacing (b3): If b3 is on, the characters in the text segment are monospaced. If b3 is off, the characters are proportionally spaced (if allowed in that font) or monospaced as appropriate.

Character Count (two bytes)

Text (one byte per character)

The Insert Figure command byte has the following arguments:

a) Figure Number (three bytes): This is the unique (ARC Catalog) number assigned to a photograph or drawing which is submitted in hard copy. The figure is merged into offset-printed copy.

b) y-coordinate of center of figure on the page (two bytes)

III NLS Subsystems

12 Output Processor

c) x-coordinate of left edge of figure (two bytes)

d) x-coordinate of right edge of figure (two bytes)

The resulting magnetic tape is seven track, contains one file per document with files separated by EOF tape marks, and end-of-tape indicated by two consecutive EOF marks.

Each file consists of any number of fixed length (512 36-bit words) records. Each word is broken into six 6-bit packets, each of which has one parity bit (using odd parity)

NEW FEATURES SINCE THE PREVIOUS REPORT

Coding the Output Processor was essentially complete at the time of the last report (13041.4de3). Development since then has consisted of limited refinements in directives, improving the interaction with our COM suppliers as noted below, and efforts to make the full power of the Output Processor more available to occasional users.

Sendprint

The network demonstration room at the 1972 International Conference on Computers and Communication exhibited a large variety of terminals both hard copy and display [4]. In various ways we attempted to prepare our system to operate in that unusual environment. The most substantial special programming was Sendprint. We anticipated that users at displays would want to be able to print files at various teletypes and printers in the room without interrupting their work.

We wrote Sendprint as a TENEX subsystem. In the past, a user of our Output Processor normally sent a processed file directly to the printer at ARC. But she has the option of putting it in a file instead. The result is an ASCII sequential file in which the Output Processor directives have been translated into spaces, carriage returns, etc., or character strings in the case of, for example, headers, and page numbers. The user of Sendprint can address TENEX and ask that the file be printed either at the terminal she is working or at a captured TTP port (usually belonging to a printer local to the user).

In that case Sendprint allows the user to suspend action until she can reach the printer and to send form feeds or control the arrival of the text page by page.

NLS formats Journal items by means of Output Processor directives. Sendprint has been used extensively, in the last two years by sites, such as Bolt Beranek and Newman and Rome Air Development Center who use the Journal extensively, and also by sites, such as Rome, generating documentation. It has been used [5] to create offset plates of NLS documentation in London.

Sendprint removes certain control characters needed by line printers of the type used at ARC. The output of Sendprint may be routed to a file and that file is then suitable for certain operations that require ordinary sequential files, notably TENEX Sendmessage. Through Sendprint it is possible to format a file in NLS and forward it through Sendmessage.

In the revised command language [6] Sendprint has become a normal NLS command (Output Remote).

III NLS Subsystems

12 Output Processor

Userguide

When the Output Processor directives were frozen in the middle of 1972 we published an Output Processor User's Guide, [7] [8]. This guide introduces the operation of the Output Processor, thoroughly describes the purpose and effect of each directive, and offers summary tables of their operation and examples of seven type faces in regular, slanted, boldface, light, and a range of from 6 to 24 p. Several hundred copies have been distributed to those interested in NLS and computer publication and is the most complex printing job attempted through NLS in terms of type faces.

Automatic Formatting

NLS COM puts tools that approach in complexity the font box of a typesetter into the hands of users accustomed for the most part only to the format control that a typewriter offers. The inexperienced user can acquire these tools gradually as his work requires them; but to allow him the benefit of our more experienced users, some standard formats have been developed. To enable someone to lay a complex format on a file without mastering all the directives involved we have developed a series of formatting programs which insert directives to produce a given appearance.

Users may apply such program by hand, or larger NLS systems may call them as procedural steps in automatic documentation production.

The first and most widely used formatter is called by the Journal, and gives Journal documents their characteristic appearance. Formatters are a step in the catalog production process [9] where they produce oversized pages for photoreduction.

We have recently established a format library where users may look at documents, choose a format, and call a program through the Programs subsystem to insert the required directives. The library includes diverse layouts, such as an Air Force Manual, which was designed for microfiche dissemination and which varies widely from normal NLS usage.

APPLICATIONS

Via Lineprinter

Production of documents through our Output Processor has been routine at ARC during the period of this contract. Examples include the previous report [3] including the headmatter, which was produced through COM; the ARPANET Directories [14] and [13], the catalogs of the NIC collections [12], the TNLS User's Guide [11], and the Network Resource Notebook [10]. The catalog, the Resource Notebook, and Userguides are documents of several hundred pages with elaborate and, in the case of the catalog, highly various formats, distributed to several hundred people, and on certain occasions updated in a controlled manner.

Via COM

Development Problems

Our files produced by the Output Processor to be printed via COM go as tapes to a vendor, Data Dissemination System Incorporated in Los Angeles. Development of output from

III NLS Subsystems

12 Output Processor

NLS to COM has been a joint venture with DDSI. In August of 1972 we completed a set of specifications [15], which we believed would allow them to interpret our output. Both sides expected that routine production would be possible within a few months. In fact, although a number of satisfactory documents have appeared and at reasonable cost to us, production you could call routine is still just over the horizon.

On the whole, we see the problems during this report period mostly with DDSI. We briefly discuss a few of them below. Note in their defense that DDSI is a producer of large-scale copies of engineering data, blueprints, parts lists, etc., that are highly repetitive in format and undemanding in quality. For them we are a small customer asking for fancy work.

Problems Arising from Our Specifying Format

Numerous vendors mint from computer tapes. In general, the CPU at the vendor formats from a character stream supplied by the customer. As noted, we supply a character stream that contains all the formatting information needed to create copy. The advantage to us lies in retaining control of a highly flexible formatting system. The disadvantage is that the vendor must interpret our formatting and suspend his own. For example, DDSI's hardware vector generator was built on the assumption that users would be required to know the actual sizes of the characters it creates. But our code makes up justifications; it has to know sizes exactly. In initial setup and in the changes in software at DDSI it has proved difficult for them to communicate the actual size and shape of the characters we specify but they form.

The format information in our text stream has also discouraged competition. On occasions we have tried to interest JPL [16] and Alphanumeric, probably the leading service group in this area, [17] in taking our jobs. In each case they have given up because, in part, of the expense both in programming time and at run time of working around their formatting code to use ours.

Local Drafts

Transportation

ARC is near San Francisco and DDSI is in Los Angeles. There has been a continuing problem of getting drafts back to authors quickly enough to check formats, particularly in the case of documents that were not formatted automatically, or in the case of trial runs of automated formats.

Transportation to Los Angeles runs quickly and smoothly through the ARPA network. Files processed at ARC go by the File Transfer Protocol, usually in the evening, to USC's Information Sciences Institute, which is only a few blocks from DDSI. At the Information Sciences Institute an operator puts them on tape and a messenger from DDSI picks them up, usually the following morning.

At various times throughout the relationship, DDSI has committed itself to different minimum turn-around times, from 48 to 96 hours. DDSI's equipment normally produces 35mm microfilm. For draft purposes, DDSI makes a Xerox copyflow proof which is mailed to ARC, preferably by United Parcel. Returning proofs may take two days in the mail and have frequently been misaddressed. Sending proofs to East Coast users takes correspondingly longer.

III NLS Subsystems

12 Output Processor

Proposals for Special Devices.

We have considered the possibility of an LDX transmitting from Los Angeles or of a low-quality front-end processor at ARC for proof purposes which could easily be constructed based on DDSI equipment [18], but the volume of business has never justified it.

Other Exemplary Problems

Missing Lines

Early samples from DDSI showed missing lines. The problem appeared intermittently. They asserted that the problem lay in the hardware character generator. Several attempts to fix the character generator in place failed and in December of 1972 [19] they demanded that the manufacturer, Information International Incorporated, redesign the hardware. At that time DDSI stopped paying rent to III pending this upgrading. Whatever the cause, it was April of 1973 before DDSI's hardware printed reliably all the lines we sent it.

Underlining

The history of underlining is similar to that of missing lines. It involved changes in their hardware. Underlining was not fully operational until the summer of 1973.

Justification and Character Spacing.

Justification is the process of making the margins of the text even on both the left and right side. Printing systems justify lines by varying the size of the white space between letters or words and occasionally by varying the width of the letters themselves.

Any type face may be monospaced as on most typewriters and line printers; that is, the width taken up by each character and its white space is the same ("w" takes the same space as "i") or it may be differentially spaced as in this report and most books. With occasional exceptions as noted below, monospace faces can justify only by adding whole character spaces between words.

Originally, five of the seven type faces DDSI offered were monospaced: Times Roman and News Gothic were differentially spaced.

As noted, the ARC Output Processor knows the size of DDSI's characters and spaces and performs the calculations necessary for justification. It would be tedious to recount the details, but correct, full justification by means of spacing between characters and words was anticipated in the fall of 1972 [15] and was not implemented until the summer of 1973. The solutions partly involved hardware changes in their character generator.

Spot Size

A spot of light creates the characters on the CRT by outlining them in the case of the monospaced stick fonts, and by painting the characters in small strokes in the case of the graphic arts fonts, Times Roman and News Gothic. DDSI had originally anticipated forming characters of all font sizes and several levels of darkness (boldness) by varying the intensity of the painting process. Early attempts at graphic arts faces came out very muddy (see the header pages of the previous report [3]), and it eventually proved necessary to

III NLS Subsystems

12 Output Processor

reduce the intensity of the spot on small type sizes, [20] We later gave up the attempt to specify more than the usual three levels of print intensity (light, medium, and boldface).

A Joint Problem

During 1973 we added a directive called Dotsplit. If you put Dotsplit between two pieces of text, the Output Processor sends one piece of text to the left margin, the other to the right margin, and inserts the right number of dots between them. The directive is very useful, for example, for creating tables of contents.

As ARC first implemented the directive, each dot went to DDSI as a separate line segment. A line segment takes a second or two to pass through their CPU. The first time we sent them a table of contents, the DDSI operator reports bystanders thought their equipment had broken, and tiny light leaks in the camera box, normally trivial because of the speed with which the film passes through the box, became a serious problem. At their request we recoded Dotsplit so the whole line of dots is one line segment.

Halftones and Drawings

DDSI can produce graphic output and can, for example, mimic a Calcomp Plotter [21], but ARC cannot yet call for anything but ASCII characters.

DDSI is able to merge text with halftone (shaded) illustrations via their MISUR system, which digitizes the source halftones. We have never used MISUR because of the cost, about \$50 per halftone [21].

We have set up directives to shape white spaces sized to correct dimensions for a given illustration on the page. DDSI has attempted to superimpose these illustrations on their film output via a flasher of the sort normally used to impose forms on microfilm output, but the quality proved poor. At present we direct proper white spaces in the output and provide illustrations to the printer who merges them with the camera-ready copy in the manner usual in offset printing.

If halftones were required in a document to be distributed in microfilm form, we would have to devise another arrangement.

Printing

Normally the output of DDSI's equipment is 35mm microfilm. For proof purposes they make copyflow pages and mail them to ARC. For printing purposes they supply us with camera-ready copy (KP-5's) [21]. In either case they retain the microfilm in their specialized storage facilities. On some occasions in the past they have subcontracted the printing job and supplied us with printed copies. That option is open in the future.

Costs

DDSI's price for providing page images on microfilm is between two and three dollars per output page; copyflow proofs are ten cents more, camera-ready copy costs a little over half a dollar per page. There are additional charges for picking up tapes from ISI and a minimum per job. In general this is not an expensive output medium.

III NLS Subsystems

12 Output Processor

An arrangement has been made for DDSI to bill Network Users directly, but it has not yet been used [22].

Accomplishments

At the time of writing, three documents have been printed through COM that have been disseminated widely, the Output Processor User's Guide, which demonstrates the full range of type faces and sizes, and two papers: Coordinated Information Services for a Discipline or Mission-Oriented Community [24] and The Augmented Knowledge Workshop [23].

A feeling that it is difficult for documents COM'd through our system to meet deadlines is chiefly responsible for other standard ARC publications bypassing COM. Users see first the turn-around delay of four to ten days for sample proof copies. In general the delays result from DDSI's production schedule and from the travel time of copy proof from Los Angeles. However, extra delays have occurred when features, often small but important, like underlining, did not work as advertised and work was delayed while DDSI (or ARC) reimplemented them.

One unusual application is business cards for ARC staff. To our knowledge they are the only business cards available as an online file [25].

At the time of writing, test runs of two user manuals were taking place. They promise to be the largest documents produced at ARC through COM, the first created on other systems to be printed via NLS, and, in one case, the first output to microfiche. [26][27]

FUTURE DEVELOPMENTS

Computer composition and the allied field of "word processing" are generally in a period of rapid technical change. It is not difficult for us to imagine improvements in our Output Processor system [28]:

1. A system that would be able to look ahead in composing each page, and thus compose footnotes, glossaries, etc.; a system that would make two passes through the document and thus be able to make a paged index and references, etc.

A two-pass system is more expensive in CPU time and only occasionally useful; it should be an alternative.

2. Directives invisible to the users, but present in the file in, for example, the manner of markers.

Parts of highly formatted files, like the ARPA Network Resource Notebook, are unreadable online because of the profusion of directives. We resort to the cumbersome procedures of a separate file to read online and another to print from, with attendant maintenance problems.

3. Proofs available rapidly near the user's terminal.
4. A display system whereon the page appears as it would appear printed.

Such a display would be useful even if it did not show real fonts, but only layout.

III NLS Subsystems

12 Output Processor

5. As a step beyond the formatting programs, an interactive subsystem that would allow the user to instruct the system about layout via simple English words and graphic devices rather than by writing in bits of code (the directives).

12e1e

Such a system should be able to guide the user by questioning her intelligently about her wants.

12e1f

6. A forms system that allows the user to instruct NL's in the layout of an external form and to subsequently call for given files or other units of data to be printed into the proper boxes in instances of that form [29].

12e1f

We believe that such improvements will come most usefully from the development work of a community of organizations using NLS for the purpose of developing, producing, and controlling documentation. The present use of NLS to produce manuals for other systems, plus inquiries from possible users who need to publish, indicate that a potential community exists [30].

12e2

ACKNOWLEDGMENTS

12f

Credit should be given to Walter L. Bass and Bruce Parsley, who did the fundamental programming for the Output Processor.

12f1

REFERENCES

12g

[1] (12b2b1) SRI-ARC. Tree Meta Report -- Preliminary Report, Formal Description. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 23-JAN-73. (14046,)

12g1

[2] (12b2b1) SRI-ARC. Tree Meta Report -- Preliminary Draft. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 23-JAN-73. (14045,)

12g2

[3] ((12d1a) (12d2a5d1) SRI-ARC. Online Team Environment / Network Information Center and Computer Augmented Team Interaction. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 6-MAR-73. (13041,)

12g3

[4] (12c2a) Michael D. Kudlick. Overview of NLS for ICC. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 13-DEC-72. (12202,)

12g4

[5] (12c2c) Stephen R. Wilbur. Problems with SENDPRINT. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 28-NOV-73. (20594,)

12g5

[6] (12c2e) No Author. NLS-8 Command Summary. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 20-FEB-73 (14537,)

12g6

[7] (12c3a) No Author. Output Processor Users' Guide. Index. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 23-AUG-73. (12215,)

12g7

III NLS Subsystems

12 Output Processor

- [8] (12c3a) No Author. Output Processor Users' Guide: Introduction. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 23-AUG-73. (12209,) 1298
- [9] (12c4c) Walter L. Bass. Catalog Production Processor System Guide. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 23-AUG-73. (12209,) 1299
- [10] (12d1a) NIC. ARPA Network Resource Notebook. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 1-MAR-72. (6740,) 12910
- [11] (12d1a) Jeanne M. Beck. TNLS Users' Guide. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 28-NOV-73. (19200,> 12911
- [12] (12d1a) NIC. Current Catalog of the NIC Collection. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 1-MAR-72. (5145,) 12912
- [13] (12d1a) NIC. Arpanet Directory. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. JAN-74. (22979,) 12913
- [14] (12d1a) NIC. Arpanet Directory. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. JUN-74. (19275,) 12914
- [15] (12d2a1) (12d2a5c4) Walter Bass. Specifications for the Interconnection of an ARC and a DDSI. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 11-AUG-72. (11382,) 12915
- [16] (12d2a3b) Dirk H. van Nouhuys. JPL Continues to Nibble at COM. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 30-JAN-74 (12878,) 12916
- [17] (12d2a3b) Dirk H. van Nouhuys. Status of COM: Meeting with Jerry Shaw and Tony Sunley of Alphnumeric. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 6-JUL-73. <17691,> 12917
- [18] (12d2a4b1) N. Dean Meyer. Talk with DDSI -- Jan 4, 1973. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 4-JAN-72. (13677,) 12918
- [19] (12d2a5a1) Dirk H. van Nouhuys. Printing by COM (Computer Output to Microfilm). Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 19-DEC-72. (13422,) 12919
- [20] (12d2a5d1) Dirk H. van Nouhuys. Development of Computer Output to Microfilm -- Current Status. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 28-NOV-72. (13047,) 12920
- [21] (12d2b1) (12d2b2) (12d2c1) Dirk H. van Nouhuys. Output Processor User's Guide and DDSI Services. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 11-DEC-73. (20847,9e) 12921

III NLS Subsystems

12 Output Processor

- [22] (12d2d2) Dirk H. van Nouhuys. DDSI, Billing NON-arc Users, Net Ambitions Productions Status. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 23-MAY-73. (16787,) 12g22
- [23] (12d2e1) Douglas C. Engelbart, Richard W. Watson, James C. Norton. The Augmented Knowledge Workshop. In AFIPS Proceedings, Vol. 42, 1973 National Computer Conference, pp. 9-21, 1973. (14724,) 12g23
- [24] (12d2e1) Douglas C. Engelbart. Coordinated Information Services For A Discipline- Or Mission-Oriented Community. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. Paper presented at: Second Annual Computer Communications Conference, San Jose, California, 24 January 1973. 12-DEC-72. (12445,) 12g24
- [25] (12d2e3) N. Dean Meyer. Business Cards for COM. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 2-JUL-73. (17597,) 12g25
- [26] (12d2e4) Dirk H. van Nouhuys. Publishing a JOVIAL Manual through COM, Comments and Questions. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 28-FEB-74. (22140,) 12g26
- [27] (12d2e4) Elizabeth A. Riddle. NSW Microfiche Format. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 11-JUL-74. (23596,) 12g27
- [28] (12e1) N. Dean Meyer. The Ultimate Format Designer. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 30-AUG-72. (11649,) 12g28
- [29] (12e1f) Elizabeth K. Michael, Harvey G. Lehtman. Staged Forms System. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 6-FEB-74. (21808,) 12g29
- [30] (12e2) Douglas C. Engelbart. SRI and a DDPCS Community. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 6-MAR-74. (23702,) 12g30

Recorded Dialog: the NLS Journal, Identification, and Number Systems (by James E White)

OUR CONCEPTION OF RECORDED DIALOG

Recorded Dialog

One of the prime objectives of the augmentation system developed at ARC is to aid collaborating knowledge workers by providing flexible computer tools and methodology for communicating with one another. We collectively refer to such tools and a methodology as a Dialog Support System (DSS). Its primary task is to provide mechanisms for transmitting online messages and documents between users. However, for large projects or those about which some larger community of users must remain informed, the dialog soon becomes unmanageable without additional computer aids. ARC's DSS therefore 1) permanently records (copies to read-only storage), 2) numbers (assigns a unique accession, or catalog number), 3) and catalogs (records author, title, number, and location) each piece of dialog--for later consultation, for reference by later documents, and for examination by interested bystanders.

The Journal

ARC's DSS is implemented as a set of computer processes called the Journal, consisting of a foreground subsystem that interacts with the user and provides primitives for entering a message or document in the Journal (with title, author and other information), reserving catalog numbers, and so forth; and a background process that further processes submission requests and delivers mail to the addressees indicated by the author. The Journal is supported by several additional systems: an Identification System responsible for maintaining information about users--their location, group memberships, phone numbers, and so forth--and a Number System responsible for keeping track of which catalog numbers have been assigned, and to whom, and which are available for future assignment.

Since its implementation in April of 1971, the Journal has been heavily used (now containing over 10,000 messages and documents), initially by the ARC staff, then by a larger user community with network access to ARC's computer facility, and most recently by commercial and government users of a second computer facility operated for ARC. The Journal has evolved as a result of our experience and in response to the increased demands placed upon it by its growing user base. This section describes that experience and evolution.

OUR INITIAL IMPLEMENTATION

The ARC/NLS Environment

ARC's DSS resides on a heavily loaded Digital Equipment Corporation (DEC) PDP-10 running Bolt, Beranek, and Newman's (BBN) TENEX operating system. TENEX provides a time-sharing environment in which 10 to 20 users independently interact with any of a variety of applications packages called "subsystems." ARC's PDP-10 is devoted almost exclusively to providing access to a single subsystem, NLS [1], a comprehensive system of tools for manipulating structured text.

III NLS Subsystems

13 Recorded Dialog

NLS provides a very general set of primitives for manipulating and viewing tree-structured text files. Commands are provided for manipulating the tree's structure, e.g., for adding nodes called "statements" to the tree, for deleting single statements or whole branches of the tree, for moving or copying a subset of the tree from one location to another, and so forth.

In order to maintain flexibility in the first implementation and to facilitate maintenance of the system, NLS text files were consistently used in implementing the Journal, Identification, and Number Systems' principal databases, as well as for catalogs, indices, and a variety of internal, inter-process communication files.

Structure

The Journal

The Journal System is a set of procedures that runs in both foreground and background modes to maintain a database of recorded documents, and to distribute them to specified addressees.

Larger Journal documents are stored as separate files in a set of system directories. Short documents, called "messages," given special treatment in the interests of economical storage, are stored in a set of (currently about 20) files, several hundred to a file. Whenever a document remains unreferenced for a month, it is archived to magnetic tape by TENEX, and its online storage released for other use. Although over 10,000 items have been journalized on the PDP-10 since April of 1971, most have long ago been archived and therefore do not occupy online storage, except when brought back for reexamination.

The Journal maintains a system catalog of all recorded documents, implemented as a set of (currently five) online files. The catalog contains information used by NLS to locate a Journal item given its catalog number, as well as information used by stand-alone programs to produce nonsystem catalogs and indices (by author, titleword, and number).

Journal mail addressed to a particular user is delivered in one or both of two delivery modes, online and hard copy. The delivery parameters are selected by the addressee and maintained by the Identification System. A document's author need know nothing about the delivery modes of its addressees.

Online Delivery

Regular users of NLS normally receive online delivery of all their Journal mail. Each item is placed by the Journal in a special NLS file called the user's "initial" file (so named because the file's name is the user's ident, which is usually his initials). For convenience, this file is automatically loaded for the user when he enters NLS. The text of short messages is delivered to the user in its entirety. For longer items, only a citation giving the document's author, title, and date, and a convenient, machine-readable pointer (called a "link") to the text of the document are delivered.

Hard Copy Delivery

Hard copy line printer output is sent by U.S. mail to users who never or only infrequently use NLS or who, for one reason or another, want it in place of, or in addition to, online delivery. A substantial amount of clerical support is required to support hard copy delivery.

III NLS Subsystems

13 Recorded Dialog

The Journal maintains information about ongoing distribution operations in a single NLS file, used also as a vehicle for communication between the submission and distribution components of the background system.

The Identification System

The Identification System is a set of procedures that maintains a large database, implemented as a single, very large NLS file, containing information about individuals, groups of individuals, and organizations (each of which is assigned a unique name called an "ident"). Various information fields are maintained for each ident, and procedures are provided for manipulating each field.

The Identification System includes an NLS subsystem that permits users to interrogate and modify the database themselves, subject to the appropriate access controls.

Because of the database size, and because updating the database involves creation of a new version of the file (requiring about 30 seconds or more of real time on a loaded system), all of the changes for a particular ident are collected from the user before the file is updated.

The Number System

The Number System is a set of procedures that manage a database, implemented essentially as a single NLS file, containing information about the assignment of catalog numbers to Journal documents. The database contains:

- 1) a number of blocks of numbers available for assignment
- 2) a list of assigned numbers (either recently used, assigned but as yet unused, or in the process of being used) and for each the date and time of assignment and the ids of the users to whom they were assigned.

It is often useful to know in advance what number will be assigned by the system to a particular item. This is necessary, for example, to create a set of documents that internally reference one another. A catalog number may therefore be reserved for later submission, or "preassigned."

The RFC number system, a separate special-purpose number system patterned after the master system (and thus able to use most of the same primitives), was implemented at the request of an informal group of network protocol developers. An item may have an RFC number in addition to the master catalog number.

EXPERIENCE AND PROBLEMS

A number of problems with the initial Journal implementation have been encountered and attacked. Some of the major problems are described below.

Excessive real-time required for submission:

In the initial implementation, the entire submission process, with the exception of delivery, was performed in the foreground and therefore kept the user from other work for what often, given the system load, proved to be an inordinate amount of time. In an attempt to alleviate this problem, the submission mechanism was restructured, and all manipulation of catalog, distribution, and storage files deferred to the background process.

III NLS Subsystems

13 Recorded Dialog

A special system directory was established for queuing submission requests for the background process that now goes through two distinct phases. First, all queued submissions are processed: numbers are assigned where necessary, the document is stored in the appropriate message or separate file in the appropriate system directory, the document is cataloged, and a distribution request is queued. And second, whenever distribution requests have accumulated are processed, one addressee at a time.

To further reduce the amount of processing that must take place in the foreground, a form of submission is permitted in which the task of assigning a catalog number is deferred to the background process. Deferred submission is the default, and most submissions are therefore of this type. Since deferred submission does not require write access to any system files, a user can submit an item in this mode at any time, regardless of the state of the Journal or Number System files.

Background delivery degraded system performance:

The Journal background process has proven to be very expensive to run, and often has had a detrimental effect upon the responsiveness of the system as viewed by its interactive users. We have experimentally varied the frequency with which the background process runs (and thus with which mail is delivered) from once per day initially, to its current frequency of once every hour.

The background process now periodically checks the load average (the TENEX monitor's measure of system demand) and suspends processing if it is above some predetermined cut-off value. Processing is resumed only when the load average drops sufficiently. The check is performed at a point in the process when the system files are consistent and least vulnerable to a crash. Between these check points, the process runs at high priority.

The benefits of this strategy are threefold: the background process does not add appreciably to the system load when it's already high; it can exploit slack times throughout the day; and since the probability of a crash increases with system load, the Journal and Number System files are usually in a relatively invulnerable state when a crash occurs.

Databases vulnerable to system failures:

A very serious problem of the initial Journal implementation was the vulnerability of the various system files to hardware (especially disk) problems, monitor crashes, and exhausted disk storage. The processing of hard copy output, besides being time consuming, was similarly vulnerable to both software and hardware failures.

The danger of losing system files because of lack of disk storage has been greatly reduced by also checking for available disk space at the same time the load average is checked. Processing is terminated until the next hour if space is too low. This strategy prevents losing a system file due to exhausted disk space during a file update.

A number of problems associated with the processing of hard copy output have been largely eliminated. A variety of monitor bugs have been fixed or avoided. The bulk of the processing is done during the evening or early morning hours. Because of the volume of hard copy output produced by the Journal, the print requests were first placed on magnetic tape and printed on an IBM 360 system elsewhere at SRI, and finally contracted outside of

III NLS Subsystems

13 Recorded Dialog

SRI Network delivery, described in the next section, has, on the other hand, drastically reduced the volume of hard copy produced, and thus recently permitted us to resume printing on our own system at OFFICE-1.

EXTENSIONS FOR A NETWORK ENVIRONMENT

The ARPANET Environment

In July of 1970, ARC's PDP-10 became part of the ARPANET, now an international network of large-scale computer facilities called "hosts" linked by 50 kb communication lines. Once the lowest level, inter-machine communication protocol was developed, the central task was to design and implement the software protocols required for general, inter-process communication, and other, more specialized exchanges. This task was undertaken by an informal group of geographically separated systems programmers called the Network Working Group (NWG).

In early 1969, ARC had offered to serve as the Network Information Center. As soon as hardware connections were made and protocol development reached a stage sufficient to permit simple, teletype-like use of a remote time-sharing system, ARC began to provide dialog support for the NWG via the Journal.

Journal Changes to Support the Network

At first, the Network user used the Journal in nearly the same manner as a local user. Like local users, he had to login to the ARC system and use NLS to compose and journalize a document. But unlike most local users, he received hard copy, rather than online delivery of his Journal mail. When ARPANET protocols developed to the point of permitting the transmission of text files and mail to users at remote hosts via the Network itself, the Journal was modified to utilize this new capability.

Network Delivery

The File Transfer Protocol (FTP) [2] devised by the NWG permits the transmission of text to a named "mailbox" at a remote host. For purposes of receiving mail, therefore, each Network user has a network address consisting of a host name and a mailbox name. To exploit this new Network capability, we added a third, network delivery mode to the existing online and hard copy modes, storing a network address in the ident file for each Network user. A Network user can thus take delivery of all Journal mail addressed to him, in his own system, simply by storing the appropriate delivery parameters in the Identification System.

Rather than deliver extremely long documents in their entirety, via the Network, we made the same size distinction for network delivery as for online delivery, sending only citations for long documents. We modified the FTP software supplied by BBN to recognize a distinctive pathname (that the Journal provides with the delivered citation) that, when used to retrieve Journal documents, invokes a conversion of the tree-structured document to sequential form before transmission through the Network. A Network user can thus retrieve the full text of any Journal document sent to him.

III NLS Subsystems

13 Recorded Dialog

Network Submission

13c2c

The fact that the Network user had to explicitly connect to and login at ARC's PDP-10 to enter a document into the Journal, and that he had to compose the document using NLS, complicated life for some users, forcing them to learn the details of NLS, in which some had only one, specialized interest.

13c2c1

To alleviate this problem, we implemented a facility that permits users to journalize documents composed via their local editor without explicitly connecting to the ARC system or logging in, and without any knowledge of the NLS command language. We did this by further modifying BBN's FTP software to recognize a special mailbox name of the form "authors/addressees" and to interpret it, in the context of a mail delivery, as a Journal submission. The ident lists of "authors" and "addressees" are verified by NLS, running beneath the FTP program in an inferior fork. If the ident lists are found correct, the "mail" is immediately journalized. Thus the remote user can journalize a document using the normal, Network mail facility provided by his system.

13c2c2

EXPERIENCE AND PROBLEMS

13c3

The Journal's Network submission and delivery facilities have been in operation since mid-1973. The latter has suffered from a few, relatively minor problems. Network addresses, for example, are not well understood by some users who, in attempting to modify them themselves, have frequently modified them incorrectly. In such cases, delivery of the user's mail is prevented until the error is discovered and corrected by ARC personnel. Because of this, almost all identification changes are now done by ARC staff. Many users are unwilling to explicitly retrieve the text of long documents for which they are sent only a citation, even though the retrieval process is straightforward, even automatable.

13c3a

The submission facility suffers from more severe problems, one of which is that the ident verification and journalization processes are very time-consuming and must be completed before the user's request is acknowledged and he is "set free." A more satisfactory strategy would be to queue the request and acknowledge it immediately, releasing the user for other work, and then to perform the expensive processes in background mode, with a Network message sent to the author in case of failure.

13c3b

A second problem is that the conversion that the Journal must make between the sequential text file presented by the user and the tree-structured NLS file required by the Journal is often unsatisfactory to the user. We believe this to be a very difficult problem to solve, one perhaps best handled by permitting the inclusion of sequential files in the Journal database, thereby eliminating the need for conversion.

13c3c

A final problem is the inadequacy of the mail subset of the FTP, which makes it difficult or impossible for the user to transmit any of the optional parameters supported by the Journal, and which forces the user interface to remain somewhat artificial. ARC has proposed a separate mail protocol [3], but no protocol development is being carried out in that area at present.

13c3d

III NLS Subsystems

13 Recorded Dialog

EXTENSION TO A DUAL-SITE SYSTEM

The SRI-ARC Utility Environment

In January of 1974, ARC began operation of a second, "utility" PDP-10 system we call OFFICE-1 to provide NLS support in a stable environment to what has proved to be an ever-growing clientele. The facility is operated for ARC by Tymshare, Inc. from Cupertino, California. Like ARC's own PDP-10, OFFICE-1 is connected to the ARPANET, through which most of its users gain access to it. The Utility's software configuration is essentially identical to ARC's, providing the full range of NLS service to its users. One such service is, of course, the DSS.

In providing Journal service from the Utility, we decided to include that second system within the domain of what is conceptually a single Journal spanning both the ARC and Utility machines. That is, rather than simply replicate the software, thereby creating a second, independent system, we decided to couple the two DSS systems, making all items journalized from either system available at both and addressable to users resident on either machine. Thus, for example, we employ a single Ident File, but maintain it in duplicate.

Structural Changes

In implementing a dual-host Journal, we were somewhat pressed for time and therefore decided to design and implement an interim system and later replace it with a more efficient and carefully thought-out implementation.

The interim dual-host Journal we decided upon involves duplicate Journal, Identification, and Number Systems, cognizant of each other at only a few points in the code. The two systems communicate with one another through the ARPANET via FTP. We implemented a special, assembly-language module to perform the FTP operations on NLS's behalf, since the corresponding FTP software provided by BBN is neither designed to be called by another program (since it's implemented as an interactive subsystem) nor structured in such a way that the relevant subroutines can be easily extracted. The portion of BBN's FTP software that was retained has been modified to deal more satisfactorily with NLS files, which have blank spots in their address space.

Two Journal Systems

Each submission request, regardless of its source, is fully processed by the Journal System on each machine. Each system's Journal catalog and document files, though in a sense maintained independently, are always identical (neglecting the obvious time lag). To avoid duplicate delivery of each Journal item, as would naturally occur as a consequence of duplicating the submission request, we partitioned the ids, assigning responsibility for delivering mail to any particular user to (in most cases) just one of the two systems--the one on which the user does most of his work.

Submission requests are duplicated in the following manner: The background process on each system, before processing recent submissions, moves any files in the other host's special communication directory (OUTJOURNAL) to a local submission queue directory (TEJOURNAL), thus adding them to the list of local submissions to be processed. Then, in processing that list, a copy of each submission request, except those obtained from the

III NLS Subsystems

13 Recorded Dialog

second host, is queued for the other system in the local communication directory (OUTJOURNAL again).

Two Identification Systems

To simplify the task of uniting the two Identification Systems, we bypassed the problem entirely by permitting additions and modifications from only one machine. The other machine is periodically sent an updated copy of the entire database.

Two Number Systems

The two Number Systems function independently, each assigning catalog numbers from a separate block. Numbers preassigned on one machine must be used on that machine, and the RFC Number system is available on only one machine.

EXPERIENCE AND PROBLEMS

Aside from the obvious inefficiency of duplicating each submission on the remote machine even though the item may be of only local interest, there have been no serious problems with our interim implementation.

An occasional asynchrony problem arises as a result of the time delay between an addition or modification to the ident file and receipt of the modified version of the database at the second machine. For instance, an ident could be added to the Identification System, a Journal item sent to him from that machine (which already knows of his existence), and the item could reach the remote system via FTP before that system becomes aware of his addition to the system, causing an error in the remote system's Journal delivery function.

The most common problem with the dual-host system is Network transmission errors during file transfers. Such failures cause the item being transmitted to be delayed until an operator finds the file in an unusual state on the source machine. He must then check the destination system to verify that the file has not in fact arrived, which is the usual case, and then requeue it for transmission. Since occasional Network failures are inevitable, we are attempting to enhance the performance of the dual-host system by automating the detection and requeuing process.

The redundancy of information within the dual-host system is occasionally useful for reconstructing data lost due to a malfunction of the file system. A backup of the file system recently experienced by the Utility cost no more than reconstruction time; no Journal files were lost.

PRIVATE DIALOG

Coming to Grips With the Problem

From the outset, one of the design goals for the Journal has been to provide an atmosphere in which memos, formal design documents, proposals, and other items, once published, would thereafter be readily accessible to anyone who cared to consult them. Author and subject indices are periodically produced and anyone, whether an active participant in the dialog or not, can therefore browse through the list of items authored by a particular individual or

III NLS Subsystems

13 Recorded Dialog

written on a particular subject, skimming or reading in full any items that look useful or appealing to him.

13e1a

This model of dialog was appropriate for the system's initial user community, ARC itself, where subgroups working on highly interrelated tasks must keep abreast of one another's activity. As the Journal's user community grew to encompass researchers throughout the ARPANET, the model remained for the most part appropriate. Again the participants were engaged in separate but interrelated subtasks of a single, large project (i.e., ARPANET protocol design and implementation), and each working group had legitimate (and often vital) interest in the work of the others. But with the extension of the Journal to a dual-host system, a new class of users became involved. Many Utility users, though anxious to use the Journal as a dialog support aid, were not at all anxious to have all of their dialog (including, perhaps, personal correspondence, new product information, and so forth) accessible to the general public. Thus ARC was compelled to address itself to the problems of nonpublic, or private dialog, and to provide support for it through the Journal.

13e1b

Changes to the Journal

13e2

What follows is a brief discussion of the more fundamental implementation problems that we encountered in tackling this problem; the reader is referred to [4] for a more detailed statement of the Journal changes made.

13e2a

Three tests must be applied in establishing a user's right to view a recorded document:

13e2b

- 1) Who is requesting access to the document?
- 2) Has he explicitly been granted access to the document?
- 3) Is he a member of any group (perhaps by way of one of more levels of indirection) that has been granted access to the document?

13e2b1

13e2b2

13e2b3

Who is the Requestor?

13e2c

The Journal has always tolerated imposters, simply accepting the user's word for the ident he declares at login to be his. It has done so because it could afford to, and because it was difficult to do otherwise.

13e2c1

Access to a user's personal files is controlled by the monitor, and all system files (i.e., Journal documents) were accessible to everyone. The only thing that hinged on the ident claimed by the user was the authorship of items he journalized during the session.

Since the Journal designates users by ident, rather than by directory name, and since elements of the two name spaces cannot, in general, be placed in one-to-one correspondence (several users, each with an ident, often sharing a single directory), the monitor's login identity check was of little use as it stood.

Rather than significantly perturb the TENEX login procedure, we adopted the following strategy:

13e2c2

- 1) For those users who have personal directories, we constructed a system database giving ident as a function of directory. TENEX was modified to infer the user's ident from his stated directory name (which, of course, had to be accompanied by the appropriate

III NLS Subsystems

13 Recorded Dialog

password) at login, using the database, and to store it in a read-only, job-global cell for subsequent interrogation by NLS.

2) For those users who share a directory, we placed opposite the directory name in the database the idents of the users who use the directory. When TENEX encounters such a user at login, it interrogates him for his ident, accepting only one that appears in the list.

Thus, those users who are assigned a personal directory, and who login only under that directory, are completely protected by the System (i.e., they cannot be impersonated), while those who work in a community directory are less fully protected, since they can be impersonated by any other member of the directory community. We are encouraging user organizations to set up separate directories for each user.

13e2r3

Has the Requestor been Granted Access to the Document?

13e2d

We have defined two classes of Journal items: private and public. Whenever a document is entered into the Journal, its author can select the class most appropriate, with public being the default. Private documents are defined to be readable only by the clerk, an author, or a distributee. That list of idents, including in general those of both individuals and groups, is stored as text in the first statement of the file that ultimately holds the document in read-only storage. Whenever a user attempts to load the file, the list is consulted, and if the requestor's ident appears in it, his request for the document is honored.

13e2d1

Has He been Granted Access by Implication?

13e2e

Since authors and distributees may be groups of people (or other groups), as well as individuals, the access list for a private document in general contains group, as well as individual, idents. A user who requests access to a private document may therefore have legitimate access to it by virtue of his membership in a group, without his individual ident appearing explicitly in the access list. Because group idents are used heavily in this way, we were compelled to provide efficient means for verifying an ident's implicit appearance in an access list.

13e2e

To this end, the Identification System was modified to maintain back links, as well as forward links between each group ident and the idents of its members. That is, not only is a membership list maintained for each group ident, but in addition, now a group list is maintained for each individual or group ident, specifying the list of groups in which the ident is a member.

13e2e2

The loggedin user's group list is loaded by NLS once per session, and by a simple search of that list, most instances of legitimate access attempts to private documents can be identified. For those cases in which the user's claim to a document is more complicated (e.g., requestor A is a member of group B that is a member of group C, that appears in the access list), the Identification System is consulted and its database examined more thoroughly.

13e2e3

EXPERIENCE AND PROBLEMS

13e3

The private dialog feature of the Journal has been in advertised use for only a few months, and hence any in-depth attempt to evaluate its performance or use would be premature. The areas in which effects are most likely to be expected are those involving intimate collaboration

III NLS Subsystems

13 Recorded Dialog

between users. It's long been common practice, for example, for cooperating users to impersonate one another to get at a file that, though necessarily residing in one particular directory, is in reality a joint file. In implementing private dialog, we've necessarily restricted such practices, and the result will probably be the design and implementation of more formal methods for accomplishing such shared tasks.

OUR THINKING ABOUT A GENERAL, MULTISITE SYSTEM

Motivation

Recognizing the immediate need to provide dialog support for Utility users, and recognizing also that the implementation of an efficient dual-host dialog support system would require significantly more than simple modification of the existing, single-host system, we elected to make the short-term modifications described earlier and then to begin design work on a general, multihost system to be distributed on an arbitrary number of ARPANET host systems.

The implementation of such a system would involve a complete rewriting of the present Journal, Number, and Identification Systems. Furthermore, we expect that the new DSS will in many ways be a different system, one in which many of the basic concepts of the previous system find a place, but also one in which new concepts appear.

Design Goals

In designing a MultiHost Journal System (MHJS), we had a number of goals in mind, the first necessarily being modularity:

Modularity:

We envision a system composed of modules, each providing some specialized service to the others, or to the end user, and which together comprise a coherent system.

Each module implements a set of primitives whose syntax and basic function are to be standardized, but whose internal workings would be left unspecified by the design (within certain broad constraints), being dependent upon the implementation machine, and the particular role that the module is to play within the System as a whole.

Reconfigurability:

The MHJS must be reconfigurable. Although the design suggests in broad terms the manner in which the System is to be constructed from its component modules, the design does no more than specify a family of MHJSs from which a particular configuration can be selected [in the same way that a computer system manufacturer provides a set of hardware modules (disk drives, CPUs, etc.) from which the customer configures his particular system].

The design specifies a small set of module types, each of which is replicated in appropriate numbers for a particular system configuration.

The MHJS must be reconfigured, for example, to accommodate the addition of new hosts to the system, or it might be reconfigured to place an instance of a frequently used module closer to a population center, or for any of a variety of other reasons.

III NLS Subsystems

13 Recorded Dialog

Optimum Data Base Distribution:

131203

It is, of course, more expensive to manipulate remote databases than local ones; sometimes it is impossible (e.g., when the remote host is down). The MHJS, therefore, must attempt to reduce the frequency with which remote databases must be dealt with by replicating portions of them in centers of user population and message traffic.

Uniform and Consistently Applied Access Controls:

131204

The MHJS must recognize the existence of private information of every type (documents, catalogs, idents, etc.) and provide the access controls necessary to protect it, providing for private dialog of a much more flexible nature than that described in the preceding section.

With these goals in mind, then, we began designing a MultiHost Journal System. Some of the more important concepts we came up with are described below; the reader is referred to [5] for a more complete discussion.

131205

SOME IMPORTANT CONCEPTS

131206

Isolating the Recording, Cataloging, and Distribution Functions

131207

The original Journal implemented a single user primitive we called "Submit" which records, catalogs, and distributes a document. We considered that primitive fundamental to dialog support, and the vision of it colored our thinking about the Journal's internal structure. We've since learned that the subprimitives from which Submit is constructed are also of interest to the user.

131208

For example, we've found it useful to be able to distribute a previously submitted document to additional users, an operation that we've implemented and call "secondary distribution" (even the name reflects our bias toward "Submit"). We now recognize, further, the need to be able to distribute a document without recording it at all, a facility that the present Journal still does not offer. And we recognize the cataloging subfunction of "Submit" to be a more generally useful tool, applicable, for example, to personal as well as system databases.

Access Controls

131209

We decided from the outset of the design to implement flexible access controls throughout the MHJS, applying them not only to documents, but to data elements of all types--catalogs, idents, and so forth. Controlling access to a data element consists of specifying, when the data element is created, the list of individual or group idents granted access to it, and then limiting access to members of that list.

131210

This is the same kind of access control now implemented in the present Journal, as we've already described, and is by far the most satisfactory type we know. In the MHJS, we've taken the additional (2nd natural) step of assigning passwords to idents, and requiring their use, as a means of verifying the user's identity.

131211

Catalog Number Assignment

131212

The present Journal assigns every recorded document a unique identifier, called a catalog number, by which the document can be referenced or retrieved. Since the MHJS is

III NLS Subsystems

13 Recorded Dialog

conceptually a single Journal, we must somehow maintain uniqueness in catalog number assignment, while yet hopefully making the assignment process reasonably efficient and reasonably insensitive to host failures. These requirements preclude the simplest implementation, i.e., assignment of numbers by a single module at a single host.

The approach we think most satisfactory is to station several instances of a module we've called the Number Vendor at strategic points about the system. Each additional Number Vendor, assuming it resides on a different host, increases the probability of a user's being able to obtain a catalog number when he wants it, as well as reducing the overhead (by placing the source closer to him).

At any time, each Number Vendor owns a subset of the universe of catalog numbers from which it can satisfy user requests. A Number Vendor may assign only catalog numbers that it itself has been assigned by another Number Vendor, except for one special root Number Vendor assigned initial possession of the entire name space.

Number Vendors might be stationed throughout the MHJS, each with responsibility for servicing a segment of the user population, and each replenishing its number supply, when it nears bottom, from the Root Vendor. This strategy permits a form of number assignment that is both efficient and insensitive to the host failures that periodically make the Root Number Vendor inaccessible.

Publishing a Document

In our design of a MHJS, we've made central a concept that is given only lip service in the present Journal, that of subcollections. A subcollection is a subset of all recorded documents, each of whose members shares some common attribute, e.g., author, subject, and so forth. A single document may be assigned to zero or more subcollections, either explicitly by the author, or by the system. Although hard copy subcollection catalogs can be generated, the Journal maintains no online subcollection catalogs, thus severely limiting the utility of the concept in its present implementation.

A major concern of the MHJS is to provide specialized marketplaces in which documents can be exchanged. Such a marketplace is called a "forum," and one speaks of "publishing" a document in a forum. In the MHJS we've thus placed great stress on the concept of allying a recorded document with other documents related to it (i.e., placing it in a subcollection), relegating the concept of simply recording a document to a less central role.

Users with interest in a particular forum can formally declare that interest, and, subject to appropriate access controls and accounting disciplines, become "subscribers" of it, thereafter automatically receiving an announcement of each new document published. The prime responsibility of the Publisher, the module that implements a forum, is therefore to catalog each document as it is contributed, and send a copy of the catalog entry (giving the document's author, title, date of publication, etc.) to each of its subscribers. We've thus given the old concept of subcollections an active, rather than passive character, with the system notifying interested users as new documents are made available.

III NLS Subsystems

13 Recorded Dialog

Maintaining Networks of Documents

1333e

For reasons of efficiency and reliability, it is necessary to permit an arbitrary number of physical copies of a document to exist simultaneously within the MHJS. Each additional copy, assuming it is created on a different host, increases the probability of a user's being able to retrieve the document when he wants it. A retrieval request can be satisfied most quickly, of course, if a copy of the requested document already exists on the user's own host. The system might therefore create a copy of the document at each major population center, anticipating a rash of retrieval requests, and then delete the copies a month later, once the period of peak demand has passed.

1333e1

Access to a document and all its copies is uniformly controlled on the basis of access lists assigned by the author. A user, for example, cannot read a document unless the author granted him read access to it. The copying of documents, however, is a system function designed to promote efficiency and is therefore unhindered by access controls.

1333e2

Each recorded document within the MHJS is therefore implemented as a network of copies whose topology is a dynamic characteristic of the system and changes with such things as the frequency with which it is referenced. The system keeps track of the various copies of a document, and can thus direct the curious user to the nearest one.

1333e3

Distributing Information About Users and Modules

1333e4

A need that pervades the MHJS, even more so than in the present Journal, is that of swift access to information about users of the system. In the present system the database is called the Ident File and describes the users and user groups known to the system. To implement the access controls that the MHJS seeks to maintain throughout, both human users and system modules are assigned identents. Group identents are very heavily used, being extremely convenient for implementing access lists for the various databases within the system.

1333e5

For reasons of efficiency and reliability, it is highly desirable to maintain copies of subsets of the Ident File at various locations within the system, each under the control of a module called a Registrar. An ident can be known to an arbitrary number of Registrars, and that particular set of Registrars is called the ident's "domain." Information about the ident can be obtained from any Registrar in its domain. Modifications to an ident are relayed by the Registrar that receives the modification request to all Registrars affected.

1333e6

The Registrar turns out to be the workhorse of the MHJS, and its importance cannot be underestimated. In designing the MHJS we discovered that:

1333e7

- 1) Virtually every system module must deal with incidental databases that are lists of user/program names (e.g., access lists), and each must provide mechanisms for retrieving and modifying them.
- 2) System modules can be relieved of a significant burden by providing a specialized module (the Registrar) whose function is to provide the primitives required to manipulate these databases.

III NLS Subsystems

13 Recorded Dialog

3) Furthermore, the lists then become accessible from any one of an arbitrarily large set of Registrars (the group ident's domain), since the Registrar already implements the required broadcast facility.

4) Since the existence of a document's read access list (for example) implies the existence of the document itself, whether or not a document exists can be determined by consulting the nearest Registrar.

5) Race conditions associated with the creation of a document (e.g., two users attempting to create a document with the same catalog number simultaneously at two different points in the system), for example, can be arbitrated by the use of locking mechanisms implemented by the Registrars.

CONCLUSION

Having made heavy and continuous use of the Journal for over three years now, ARC has found it to be a powerful dialog support tool for knowledge workers

During the course of its use, the Journal has been substantially modified to increase its efficiency, extend its geographical reach, and provide the new features we've discovered to be important. Initially an experimental system supporting a fairly small number of geographically concentrated researchers, it now supports a large, geographically distributed user community linked by the ARPANET. Initially a software system implemented on a single computer, it now operates on a pair of FDP-10 systems linked by the Network, and design work has been done for a general, multihost system. Initially exclusively a forum for public dialog, it now supports private communication as well.

The Journal will further evolve and new features will be implemented and experimented with as we continue to gain experience in the dialog support field.

ACKNOWLEDGMENTS

Many past and present members of the Augmentation Research Center have contributed to the design, implementation, and evolution of ARC's Dialog Support System. The contributions of the following individuals warrant special acknowledgement: William S. Duvall, Douglas Engelbart, David Evans, J. David Hopper, Charles Irby, and Jeanne North.

REFERENCES

- [1] (13b1a) SRI-ARC. Online Team Environment / Network Information Center and Computer Augmented Team Interaction. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 6-MAR-73. (13041.)
- [2] (13c2b1) Nancy Neigus. File Transfer Protocol. BBN. Boston, Massachusetts. 12-JUL-73. (17759.)
- [3] (13c3d) James E. White. NWG/RFC 524 #1 A Proposed Mail Protocol. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 31-MAY-73. (17140.)

III NLS Subsystems

13 Recorded Dialog

- [4] (13e2a) James E. White. A Description of the New NLS Privacy Features. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 7-MAY-74. (22911,) 13 4
- [5] (13f2b) James E. White. Description of a Multi-Host Journal System Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. MHJSPAPER.NLS;33,. (23144,) 13 5
- [6] D. C. Engelbart, R. W. Watson, J. C. Norton, The Augmented Knowledge Workshop. In AFIPS Proceedings, Vol. 42, 1973 National Computer Conference, pp 9-21, 1973. (14724,) 13 6

User Program System and Library (by N Dean Meyer)

INTRODUCTION

As described in our previous final report [1], the L10 programming language and the User Programs subsystem in NLS allow the user to write programs that:

- act as filters through which the user may view or edit a file,
- automatically edit a series of statements based on content analysis,
- define special-purpose sorting algorithms,
- produce special sequences of statements from a file, and which
- constitute special-purpose additions to the NLS command language.

Our emphasis in this contract period has been on delivering these extended tools to the user. Our efforts have been focused on two areas:

- 1) We have provided a library of user programs, written by our staff and our users, that satisfy common needs.
- 2) We have made L10 easier to learn.

Our work in this area is in keeping with our efforts to apply the Knowledge Workshop techniques to the needs of groups outside of ARC, and to study the process of integrating NLS into ongoing work situations.

The User Programs Library allows us to experiment with a wide variety of additions to the NLS environment, to freely evolve potentially useful tools, and to examine their usefulness. It also provides an inexpensive way to add capabilities that are not general enough or used enough to warrant inclusion in the NLS command language.

As users learn to program, NLS should adapt itself very closely to the unique needs of each group. ARC, in turn, will be given a tangible medium for understanding the special problems of our application groups.

USER PROGRAM LIBRARY

User Programs

At the time of writing, there are 37 programs in the User Programs Library. They include:

- content analyzer filter programs (e.g., one that displays only those statements with Output Processor directives in them),
- content analyzer editing program (e.g., one that deletes spaces at the beginning of each statement),
- sort algorithm program (e.g., one that ignores statement names),

III NLS Subsystems

14 User Programs

executable program, which to the user looks like a special purpose command (e.g., one that executes a substitute command on a set of files). 145144

Many were written to fulfill specific needs. Some can be grouped into process aids. 145145

For example, message handling might be facilitated by using the executable program that copies one's MESSAGE.TXT file into NLS, the content analyzer program that edits journal and message citations to a form designed for one line views (so that the two types of mail can be integrated), and then the sort algorithm that orders the branches by date. 145150

These few aids make it easy to handle one's mail entirely in NLS, exploiting all the powers of NLS for viewing, classifying, deleting, redistributing, and working with the messages.

The documentation production process is another example. The Library includes programs that produce a bibliography of the journal references in the file, produce a Table of Contents at the front of the file, insert directives throughout the file to format according to any of a number of predesigned styles, create a title page, show only those statements with Output Processor directives in them (to ease format adjustments), and (having processed the file for output) delete all directives (leaving the file clean for online viewing). 145155

All programs included in the User Programs Library meet certain standards. They must include adequate error checking, they must be well commented and documented, their source code uses NLS file structure to make their flow clear, they only use unconditional program control transfers where required, and the executable programs must follow the command syntax conventions throughout NLS. Programs that were offered but did not meet these conditions were made to do so. 145160

The programs in the User Program Library are a rich source of examples for the novice programmer. They range from simple to very sophisticated programs and cover a wide range of activities. 145165

Access from NLS 145170

The command that loads a program into the user's programs buffer space has been modified to check the directory <user-progs> [where all the programs in the Library (both source and object code) are stored] if it does not find the program in the connected directory. The user need only know the name of the program. 145175

The command also looks at the extension of the object code file, which simplifies use of programs in the Library. 145180

The extension "REL" simply makes the command load the program. These are executable programs. 145185

"CA" makes the command load the program and institute it as the current Content Analyzer filter program. 145190

"SK" makes the command load the program and institute it as the current Sort Key extractor program. 145195

"SG" makes the command load the program and institute it as the current user Sequence Generator program. 145200

III NLS Subsystems

14 User Programs

Documentation

Every program in the Library has in its source code a branch that explains how to use that program, what it does, how much buffer space it requires, and who wrote it.

All but the author's ident is also stored under each entry in a Table of Contents to the available User Programs (user-progs,-contents,1:w). This Table of Contents is formatted so that a one-line one-level view will give a quick listing of all available programs (user-progs,-contents,1:x); this list is included below.

In combination with the very basic User Programs Users' Guide (user-progs,-userguide,1:w), the non programmer should find it easy to use the programs in the User Programs Library.

USER L10 PROGRAMMING

Documentation

To facilitate the learning of L10, a new L10 Users' Guide has been written [2]. This is the first L10 guide to be tutorial (as opposed to simply a reference guide). It begins with simple Content Analyzer filter patterns, and develops the user through L10 Content Analyzer filter programs, editing programs, and finally executable programs. It provides examples, and does not assume much in the way of programming expertise. As well as an explanation of the many features of L10, it includes tricks of the trade and system information so necessary to programming in the complex NLS context.

Although this document is large (100 pages), it is arranged so that the user can work to whatever stage of skill he wishes. The document has been modified as we got feedback from those who have used it. It is expected that the L10 Users' Guide will continue to grow in both content and care of presentation; it should continue to take on aspects of a programmers' handbook.

A listing of all procedures in the NLS system is another aid to L10 programming that is available. For each procedure, this document provides a list of the formal parameters, a brief explanation of the procedure, and a link to the source code. Since so much of L10 programming depends on knowing what system procedures are available, this file has proven extremely important to both novice and expert programmers.

In addition to helping the programmer find a procedure that satisfies his need, it allows the novice to find examples of L10 in the system code itself. If the programmer finds a command that at some point must do what he wishes to do, he can follow system code through and see how the problem is solved there. This is a good way to learn both L10 and programming style. The list of procedures helps one follow the code and find the source listing of each procedure the system calls.

Interface to Core Procedures

To further ease the work of the novice programmer we have begun writing a set of procedures which match the NLS commands. Each command will have a corresponding procedure which requires the same parameters in the same order as the command. These procedures will

III NLS Subsystems

14 User Programs

include very careful error diagnostic messages. This should allow the novice to call in his program any NLS command he knows.

14-2a

LIST OF USER PROGRAMS

14-3

Program	Function	Type
Addname	Adds name to nameless stmts from first word in stmt	C
Address	Asks for ident, inserts the address at the bug	E
Addtext	Adds text to front/back of statement in plx/brnch/ grp/st	E
Append	Sequentially appends stmts in group, text between	E
Appendlist	Like Append, but leaves substructure	E
Changed	Marks statements changed since a given date	C
Delcol	Deletes bugged col, assumes next col lined up	E
Deldir	Deletes Output Processor directives	C
Delname	Deletes statement names	C
Delsp	Deletes leading spaces from statements	C
Format	Adds print directives to a file	E
Index	Creates a word index for st/br/plex/group	E
Inmes	Inputs all of message.txt file into NLS file	E
Inseqh	Does a sophisticated Input Sequential file	E
Inrun	Inserts TENEX Runoff file into NLS file	E
Jform1	Reformats Journal references	C
Jform2	Reformats Journal references	C
Jform3	Reformats Journal references	C
Letter	Puts file in letter form, adds dear & sincerely	E
Lowercase	Recovers from an erroneous XSET UPPER CASE	C
Makeref	Scans for Journal links and makes ref branch	E
Notabs	Replace tab keys by spaces in plex	E
Printcml	Runs and prints CML programs/grammars.	-
Sendmes	Sends messages from NLS	E
Showdir	Shows only stmts with Output Processor directives	C
Sortmes	Sorts key extractor: by date at beg of statement	SK

III NLS Subsystems

14 User Programs

Sortnmskp	Sorts key extractor: as usual but ignores statement names	SK
Sortnocase	Sorts key extractor: alphabetic, disregards case	SK
Sortnum	Sorts key extractor: sorts by first number in stmt	SK
Sortrev	Sort key extractor: exactly the reverse of usual	SK
Sriform	Puts in OP dirs and spaces to SRI standard format	C
Sublist	Does substitutions on stmt of files given column	E
Tblpts	Adds periods to end of stmt out to given column	C
Toc	Generates Table of Contents with stmt num refs	E
Trace	NLS call return tracing system for microanalysis	E
Truncate	Truncate: st/br/plx/grp to one line-assume 3/lev ind	E
Wordcount	Counts visibles in st/branch/group/plex	E

REFERENCES

- [1] (14a1) SRI-ARC. Online Team Environment / Network Information Center and Computer Augmented Team Interaction. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 6-MAR-73. (13041.)
- [2] (14c1a) SRI-ARC. L10 Users' Guide. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 11-SEP-73. (18969.)

Query/Help Software and Data Bases
(by Harvey G Lehtman, Kirk Kelley, Dirk H van Nouhuys,
and Jeanne M Beck)

**QUERY I -- ON-LINE PORTRAYAL OF THE NIC RESOURCE NOTEBOOK FOR
THE ICC: CAPABILITIES AND LIMITATIONS**

In October 1972, ARPA participated in the first International Computer Communications Conference held in Washington, [1] D.C. Many sites on the ARPANET gave demonstrations of their systems showing online distributed computing techniques to those attending. As part of its function as the Network Information Center, ARC provided on and offline information about the identity and resources of nodes on the net. The database containing this information was housed in several NLS files and was known as the Resource Notebook; these files were created and maintained in NLS with hard copy summaries and editions produced with the Output Processor. While they could also be studied online using NLS, a simple special purpose user interface was produced for users unfamiliar with the NLS system. This interface was known as Query.

Query I, which operated only in typewriter mode, had two principal commands: "Bring," which loaded the database for a particular site, and "Show," which permitted the display of particular information about a site. (Other commands listed the sites for which descriptions were available.)

If a node was to be shown (e.g., by giving the command "Show Personnel") all information at the node (in the NLS statement named) would be printed out. In addition, the first line of information one level under the node being shown was printed. These first lines of the lower level statements constitute a menu of immediately selectable nodes. The names in the lower menu could then be used as the operands in further Show commands.

An example of a typical Query menu for a host file from the Resource Notebook follows:

-bring rand-rc

(RAND-RCC) THE RAND CORPORATION

Rand Computation Center

Choose one by typing: s[how] personnel <CR>

(FUNCTION)

(ADDRESS)

(PERSONNEL)

(RESOURCES)

(SYSTEM-USE)

(INTERESTS)

(DOCUMENTATION)

Preceding page blank

III NLS Subsystems

15 Query/HELP

Two limitations in Query I are apparent:

- 1) The system operated in typewriter mode only. We wished to incorporate some of the display techniques available in DNLS (e.g., a cursor pointing device to make selections).
- 2) The database builders were limited to a strict structure: the power of NLS links to connect nodes (to avoid duplication of information) was not available.

Considerations such as these, combined with the desire to provide online documentation for NLS-8, led to the design for Query II/Help described below.

QUERY II DESIGN

Design meetings, implementation of partial test versions of the system, evaluation, and redesign took place on a new version of Query. (The existing Query I was cleaned up and an LIO program written to aid in the maintenance and verification of the Resource Notebook.)

The design for Query II resulted from a need for the incorporation of general reference-data management and retrieval tools in NLS. As such, the design attempted to satisfy several, perhaps conflicting, needs.

A general system required a fairly complex user interface to permit general full-text searching of data. Eventually, inversions would be desirable. Searches by name in NLS are reasonably fast, but content searches over large uninverted multi-file NLS databases can be slow. The use of the Data Computer for storage and retrieval of information was also considered [2][3]. A full Boolean expression retrieval language was designed. Commands for the creation of new sets of information and output to files or printers were desirable.

A general Forms system [4] was also being designed, and it was desirable to incorporate its data management and retrieval requirements into the Query II design.

We hoped the system (or at least a simple subset) would be capable of providing online documentation of the user's current state and of the rest of NLS. Information about the user's state from the CML interpreter could be used to provide entry keys into a structured database describing the system.

The part of the new Query system with highest priority was the Help system. Because of resource limitations, the designs for a general Forms system and reference-data management systems were completed, but not implemented. The core code of the current Help system could be used, with modifications, to permit access of more than one file (with an attendant complication of search algorithms). Other straight forward additions would be implementation of a more complete user interface permitting bugging of selections in DNLS and the use of Boolean search expressions over general databases.

A Help System, however, had to be oriented toward the needs of novice users. The command set had to be minimal. Some in the group felt that even the use of the command word "Show" as in Query I was excessively confusing. The Help mode had to be a single command. Its wider powers need be apparent only to sophisticated users. The Help System user was to be led through the database by its structure and presentation of menued nodes that could be selected by number.

III NLS Subsystems

15 Query/HELP

THE HELP SYSTEM AND OTHER CML-NLS-8 HELP FACILITIES

The Help System, an intended subset of the general Query-II, was to be but a part of an array of online user assistance tools: typing a question mark at any point in the command syntax gives the user of NLS a list of command alternatives at that point; typing a Control-S at any point gives the complete syntax of the command (with all of its remaining variations). These facilities are implemented by code that threads through the command state information maintained by the CML interpreter and the CML NLS grammar tree (see sections 8,9). If the user types a Control-Q, semantic information related to the user's exact status in the command specification is obtained from the Help database and displayed.

The CML interpreter maintains a command stack which contains, among other information, pointers to command words specified in the current command. This stack is followed and these command words extracted into a list which serves as the first parameter keys into the Help database.

All information at this "principal node" is portrayed, along with linked information: a discussion of the structure of the Help database is found below. In addition, information offerings are menued -subsidiary information offerings are listed with menu numbers. The user may follow any learning path desired, by menu number or node name selection. He may return to information that had been seen before. Return to NLS is accomplished by typing a Command Delete. Entry to the database may also be accomplished by issuing the Help command directly from any NLS subsystem with an optional node path specification. The paths in the database may be followed in the manner described above.

Note that portrayal of the database in the Help command is controlled by the Help System and the database builders. In the rest of NLS, portrayal is controlled by the user with Viewspecs. In Help, embedded Viewspecs are inserted by the database builders. (The file in the database is an NLS file and may be studied using the normal NLS facilities.)

Examples of these help tools are illustrated below in Figures 1 through 7.

III NLS Subsystems

15 Query/HELP

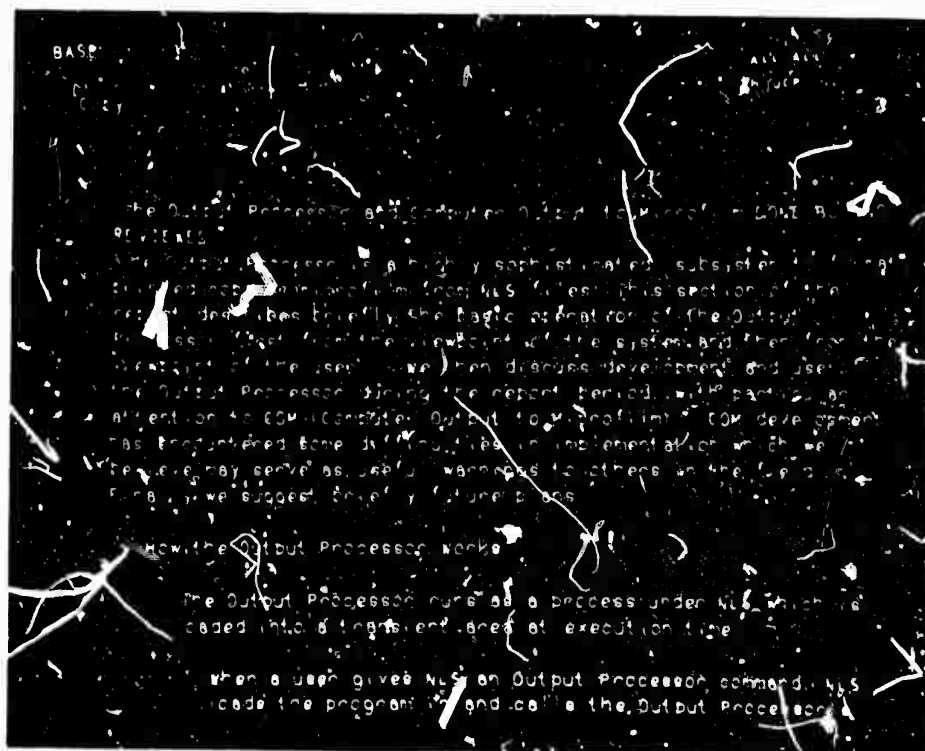


Figure 1. A user issuing the NLS "Copy" command. (Prompting for the next command word is done below the command feedback line--i.e., by the "C" on the line beneath the word "Copy".)

1543

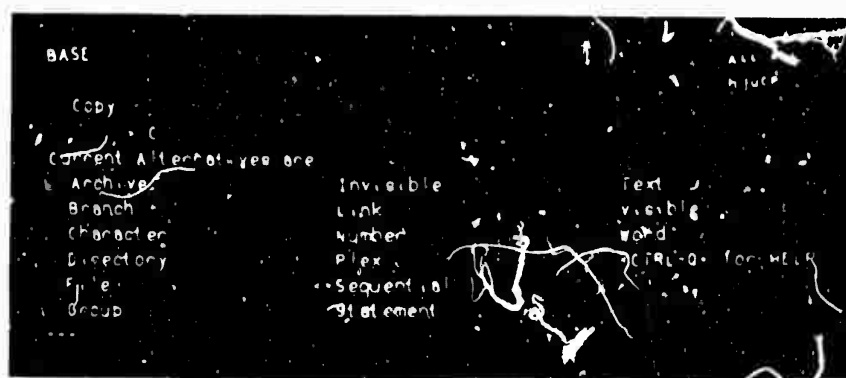


Figure 2. Typing a question mark leads to the display of the next level of command choices.

1544

III. NLS Subsystems

15. Query/HELP

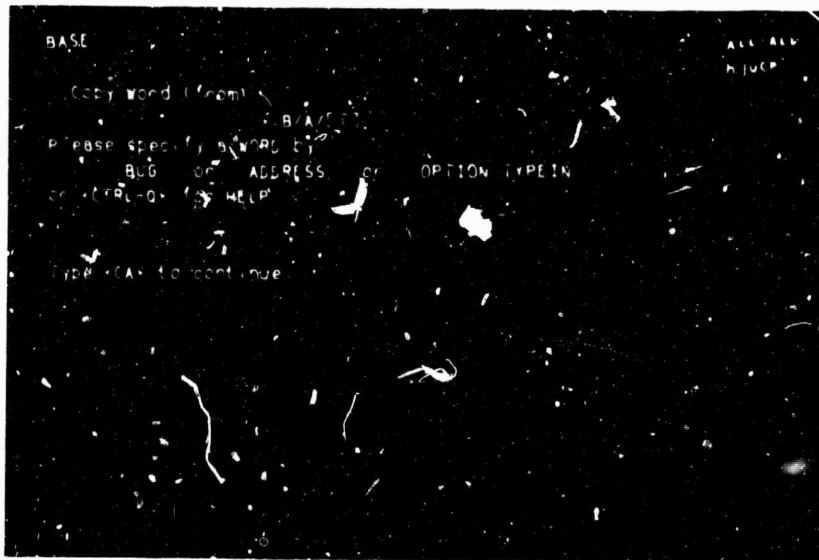


Figure 3. After selecting the "Copy Word" command, the user is prompted for an entity selection, as indicated by the "B/A/[T]". A question mark produces the explanation shown.

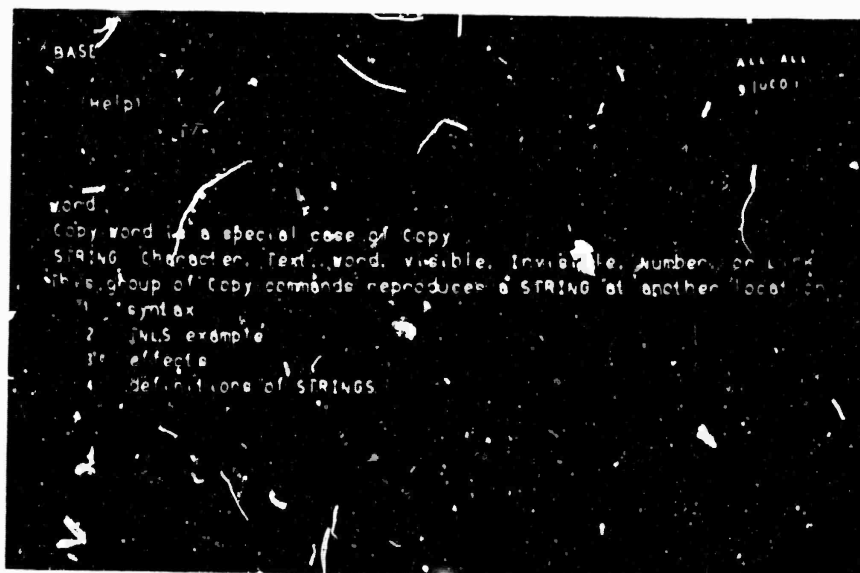


Figure 4. More detailed information is obtained through the "Help" command entered by typing Control-Q. A simple explanation is followed by a menu of nodes which may be selected.

III NLS Subsystems

15 Query/HELP

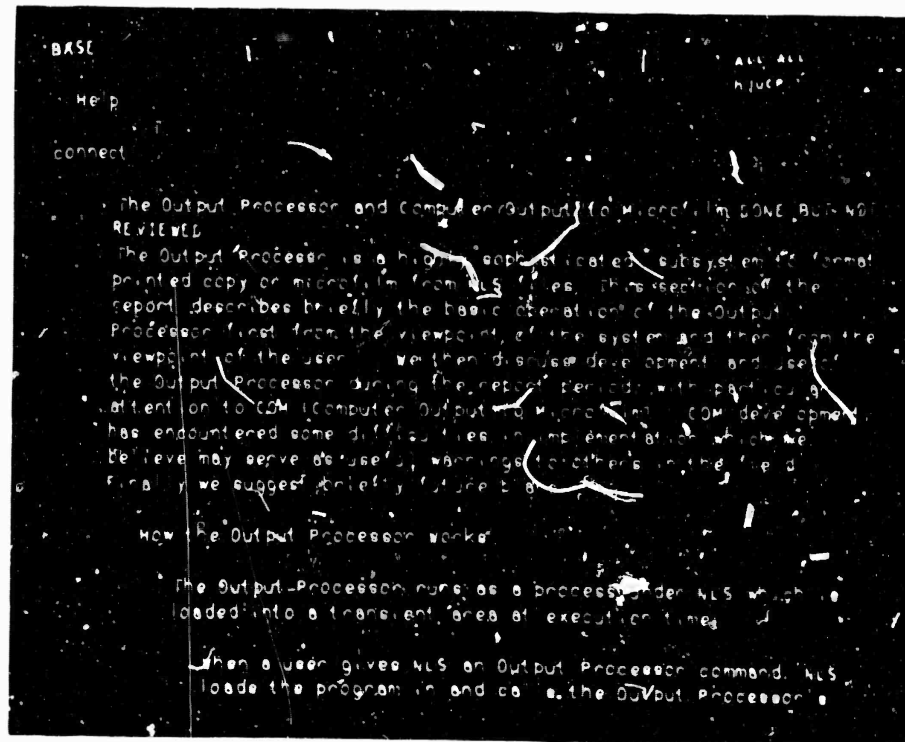


Figure 5. The Help command may also be evoked directly outside of the context of other ongoing NLS commands. Here, information is sought on "CONNECT" on entry to the command.

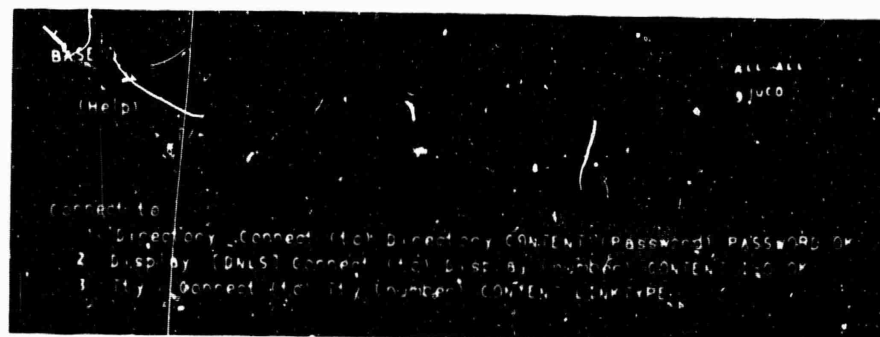


Figure 6. A menu is provided.

III NLS Subsystems

15 Query/HELP

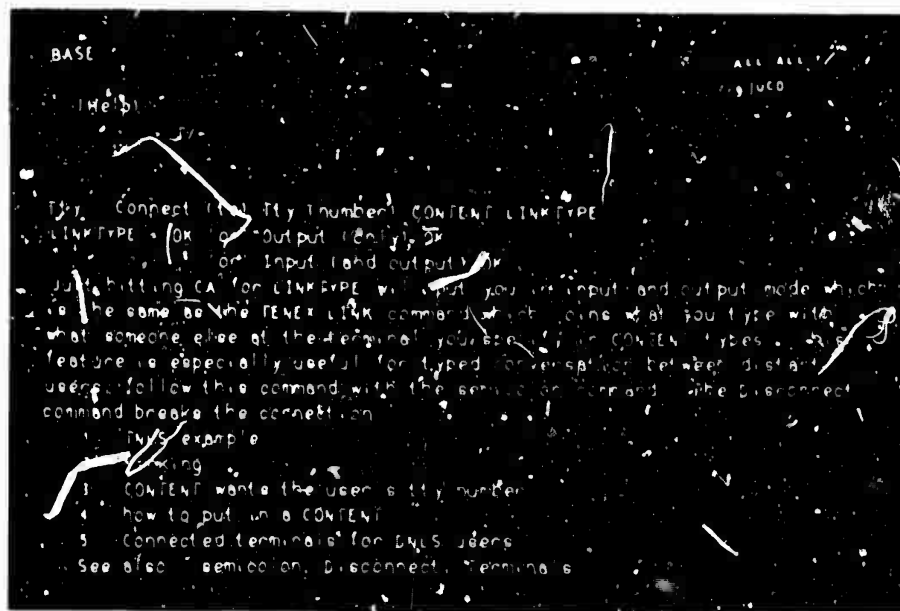


Figure 7. Paths may be followed. A command delete (CD) returns the user to the current subsystem.

HELP DATA BASE STRUCTURE AND CREATION

Overall Structure

Currently the Help database occupies the equivalent of about 200 pages (50,000 words) and is written in a modular hierarchy where each module is an NLS statement called a "node" and contains either around eight lines of text or an indirect address called a "link," which points to a node containing text.

The database is divided into three major sections:

- 1) NLS concepts
- 2) Syntax of commands
- 3) Lexicon: glossary, thesaurus, index.

The concepts are arranged logically. The commands are arranged in a branching structure reflecting the tree structure of the commands -- e.g., the node that defines "output" has as substatements nodes defining the various second words possible in the Output command (Printer, Quickprint, File, etc.). The lexicon serves to route searches from approximately correct words to the proper node -- e.g., a lexicon entry for "links" routes the search to "link," or the lexicon item for "show" offers the user a choice among the several types of showing in NLS.

The NLS Concepts

The NLS concepts are arranged in a combination hierarchy-network. This is to allow the user several ways of getting Help with NLS. Some of the possible ways are:

III NLS Subsystems

15 Query/HELP

- 1) Reading sequentially from the beginning as a tutorial
- 2) Entering at any point and finding the answer to a specific question
- 3) Browsing through the database; a combination of the above.

As the command language first functioned, there was no indication to the user that the database was hierarchical. We discovered that this approach made many users feel unanchored. It was particularly disconcerting to start down a path and suddenly find oneself looping back to a "higher" point already visited. Beginnings, major subdivisions, and tables of contents were all missed.

Later we made available to the user a command to show the next higher node. Only fairly sophisticated users employ this command. Such users can picture themselves in a hierarchy and yet easily jump off to learn about an unfamiliar concept that may be in a completely different structural and conceptual branch.

The Syntax of Commands

This section of the database is divided according to NLS subsystems into sections that contain lists of the available commands in alphabetical order. The Editor is the largest subsystem and contains so many commands besides text editing commands that we decided to further divide it into one more level of categories. Each command is described in a concise but not cryptic notation specially developed for the Help subsystem. For example, command words appear with initial capital letters while special variables that can mean more than one thing are in all caps. DESTINATION tells the user to point to something; CONTENT means you can point to something that already exists or type something new; OK tells the user to hit a confirmation key. Examples:

Delete Character (at) DESTINATION OK

Replace Word (at) DESTINATION (by) CONTENT OK

The Lexicon

The Lexicon is a linear list of words in alphabetical order with or line pointers to synonyms, alternate spellings, duplicate locations of the same word, and definitions of special terms. Currently this list is searched before the rest of the database. In this way we can trap a term that is used to mean different things in different parts of the hierarchy. We try to avoid using ambiguous terms wherever possible, but it is unavoidable in many cases. For example, asking for the word "number" by itself gets you several choices:

- 1) Journal number
- 2) STRING number
- 3) Statement number
- 4) SID: Statement IDentification.

One of these four is picked to specify more precisely the kind of number the user wants to know about

III NLS Subsystems

15 Query/HELP

Format for Nodes

The optimum format for a single node given the current accessing system differs very much from conventional hard copy formats. The first line of a node is written to appear appropriately as a single line in a menu. The body of a node contains a short description of its subject. At the end of most nodes is a list of suggested reference terms. The total length of a node is limited to from six to eight lines so as not to burden the user with too much printing at one time. After a node has printed, the first lines of each node in the next level of that node's substructure are automatically menued as choices in a numbered list.

For example, the node and associated menu for the term "NLS", as they would be displayed to a user, are:

NLS: Online System

NLS is ARC's central tool in the Augmented Knowledge Workshop. It has a Command Language divided into subsystems for specific tasks in Information Space. When you enter NLS, you begin in the Editor, which contains many capabilities including information modification commands. You are now in the NLS Help subsystem. Show also: ARC, subsystems.

1. Systems: entering and leaving
2. Viewing information:
3. Information space:
4. Modifying and creating information:
5. Command language:
6. Editor subsystem:
7. Help subsystem:
8. Sendmail subsystem:
9. Useroptions subsystem:
10. Other subsystems: Calculator, Identification, Programs and TENEX.

Maintenance

Labeling: Statement Names

A form of NLS address called statement names is important to the maintenance of Help. Names are strings at the beginning of statements set off by certain characters. The database builder can choose what characters serve to set off names. Different name delimiters are used for different effects. Names ending in colon are used for conceptual definitions. Names ending in carriage return are used for command syntax to allow only the name to be seen or to allow nothing but the body to be seen when desired.

Various name maintaining programs were written. These can be used to have the system show only unnamed or show only named nodes. One program automatically goes through a file and makes sure every statement containing a colon in the first line also has its first word designated to be a name.

III NLS Subsystems

15 Query/HELP

Attempts were made wherever possible to allow meaningful multiple word term queries by placing named statements in appropriate hierarchical positions. For instance, "syntax delete file," "editor commands," and "statement name" are all valid queries.

15d1a1

Indirect Addressing: Links

15d3b

Wherever information is desired in more than one place, it is written once and then a pointer to it is placed in all of the other locations where direct access to that information is desired. These pointers are addresses delimited by special characters. The search path resulting from a link is identical to that followed by the Help System when a similar series of node names is specified directly by a user. They are written to search in the same order and to use the same words as the accessing system does when searching for words asked for by the user. A program was written that automatically locates and follows every link of the proper format and makes a list of those that don't work.

15d3c1

To prevent visibility of the links to the user, they are entered in the database in a special format and are stripped away before portrayal. For example:

15d3d2

Sendmail Commands

```
##<sendmail!commands>##
```

Documentation Algorithm

15d4

A Help manual is maintained and updated online. It contains conventions listed above as well as other conventions and the step-by-step process necessary for writing, updating, and developing the Help database as the primary up-to-date source of documentation for NLS and the Knowledge Workshop.

15d3c1

DEVELOPMENT HISTORY: SUCCESSES AND MISSTEPS

15e

Introduction

15e1

The Help System began modestly with the notion that NLS should be able to answer more complex questions than systems that yield only lists of possible commands or brief scenarios. The ambitious system that evolved has a complicated data structure in terms of NLS, attempts to give a reasonable explanation of the user's situation at any point, or of any term a user might reasonably expect to be relevant, and replaces the heavy System Reference Guides common in the industry. It has acquired this complexity through evolution rather than planned growth. Other developers are building systems like this and more will come. It seems worthwhile to record here for future works, some of our difficulties in making design decisions and some of the unforeseen consequences of those decisions.

15e2a

Level Versus Hierarchical Structure

15e2

We began with a resolution to offer prose definitions of NLS terms and under each definition offer a menu of related subjects. The user might then choose a menu item and see that

III NLS Subsystems

15 Query/HELP

definition with its associated menu. The menu items would be a form of NLS link [5]. The question was, how should the statements be named in the link?

NLS statements may be addressed by names. Names are a string of characters set off by delimiters. The names are stored in a hash table and searches for statements by name are fast. NLS statements may also be identified by numbers attached to each statement sequentially at creation.

NLS files in normal use are organized hierarchically, in the form of an outline. The simplest form of Help file would name each statement with the subject of inquiry: one name to one statement. This plan was seriously considered; it would have greatly simplified development but made a less powerful system. It was rejected mainly because of the problem of duplicate names. How could we distinguish between two statements with the same name? If the Help file had not had structure, we would only have been able to write one definition of "delete" that would have had to apply equally to "Delete Branch," a file shaping command, and to "Delete Edge," which alters the arrangement of the display screen. Because of restrictions in the NLS definition of a statement name, an alternate solution would have made use of hyphenated names to distinguish between, for instance, "delete-edge" and "delete-file," but it soon became apparent that we would end up with a large number of strangely constructed names.

In the end we chose to arrange the file hierarchically; with a hierarchical file we could distinguish between two statements with the same name by their position in the structure. It was then possible to specify statements in links by a path arrangement: that is, the search for the link <delete! edge> continues until a statement named "delete" is found and then search begins in the branch headed by "delete" for a statement named "edge."

One result has been a very highly branched file that has, at least twice, been thoroughly reorganized [6][7][8][9] with considerable expenditure of person-hours.

Our decision to differentiate among names by using a complex hierarchy was undoubtedly influenced [11] by our study of the ZOG system [10].

Other arrangements are possible; for example, the database could have been distributed over several files. We rejected that alternative mainly because of software complications and time consumed in opening files in response to a user's questions.

The Relation Between On- and Offline Documentation

The development of the Help System occurred at the same time as did a generation of widespread changes in the command language and allied features, spoken of as the transition from NLS-7 to NLS-8. A thick userguide [12] existed for NLS-7, which cost in the order of a person year to create. Since the NLS system is so easily revised, we had more than our share of the problems of maintaining a hardcopy manual, as many people who used NLS through the ARPANET were geographically scattered. It was clear that an effort of the same scope was necessary to make new documentation for NLS-8 that would serve as much more

III NLS Subsystems

15 Query/HELP

than an occasional reference guide. We had originally planned to write an NLS-8 Userguide and draw from it the content of the Help database. It proved more reasonable to write the Help database and derive documentation from it.

Initially we expected to prepare a hardcopy reference guide which would document NLS-8 completely. The online Help database was to offer a simple, incomplete overview of the system. As the Help file structure became more complex, it became apparent that it could contain a complete accounting of NLS. It also became apparent that completing the database would consume the available resources.

More important, checking the database has proved so time consuming, both because it covers so much and because of the difficulty of working out its complex structure, that the reference guide has never been written -- the reference place for NLS information is the Help database, not a hardcopy document.

It is not at all clear whether this was a desirable outcome. Many users are uncomfortable because they lack "something I can read and get an idea of the system." Printing various primers, introductions, and special purpose guides has only ameliorated this nervousness. An interested user cannot always log in, and is not accustomed to learning from reading computer output. Exposition via a medium like Help is not nearly so evolved as book exposition. From its introduction to users at ARC, there has been much feedback, through the Journal, concerning both initial implementation bugs and general reactions to the system itself; opinions were initially mixed with strong feelings pro and con. [13][14][15][16][17][18]

The role of the Help database as basic reference source juxtaposed with ease of revision here led to strains between a newspaper sort of function and an archived function. For example, any user may write a user program in CML or L10 that may be valuable to other users. It also may not work perfectly, especially when applied in new contexts or when systems programmers alter some fundamental procedure. ARC has undertaken to certify certain user programs as bug-free and to maintain them against deterioration caused by system changes. These programs plainly should be documented in Help. But, because of limited resources, many very useful programs are not supported. Should they be described in Help? The present compromise is that Help users are referred to other online sources for documentation on unsupported programs with a warning that ARC does not guarantee that the programs will work.

If we write the usual sort of user guide, these programs probably would not be documented at all, and if they were, the document would easily become out of date.

The Help Command Language

Many controversies in the development of Help arose from conflicts between the special needs of a Help command language and the traditional style of NLS commands. NLS commands characteristically begin with a verb that names the action of the command followed by a noun that names the object of the verb, then one or more arguments concluded by a character that

III NLS Subsystems

15 Query/HELP

begins implementation of the command. See --5d, above)). A variety of recognition modes are available and it is possible to limit echoing of command words and printing of noise words. A substantial majority of users, particularly among the system developers using DNLS, invoke commands by typing a single letter, or 2 or 3 letters in the case of little-used commands. Then they expect to see the full command word, one or more explanatory noise words, and then a prompt for the next step.

A general opinion at ARC holds that command phrases that express the English meaning of the command and include full echoing following little typing are good human engineering. But important exceptions exist. Viewspeak is entered by unmnemonic single characters and certain single characters (<CTRL-E> and <CTRL-B>) replace a complete command and cause repetition of the previous command respectively. It is possible to implement the command <CTRL-B> by pressing two mouse buttons up and down with the right hand, a gesture unrelated to any English characters.

For reasons having to do with user views of the command language, NLS was divided into subsystems, each with an independent list of commands. A separate command moves users from one subsystem to another.

We first made the Help base accessible to the user in two ways. The simpler is <CTRL-Q> which initiates a search in Help for the node that explains the command words the user has just specified. The more complex method was to call a subsystem named Help that included mainly the command "show" borrowed from the NIC query system (see 15a2). For most users that meant typing "s", seeing the word "show" echo, then typing in a term of interest or a menu number, and then a <CA>.

Although there have been some suggestions that another character or multi-character sequence (e.g. "??") replace <CTRL-Q>, that part of the Help command language has always been satisfactory. On the other hand we soon realized that the subsystem entry and "Show" command were awkward to learn and needlessly complex. A discussion followed in which the participants agreed that the command to search for a term or menu item should be

- 1) easy to learn,
- 2) easy to use,
- 3) in conformance with the general style of NLS.

But opinions on the order of those priorities and the way to carry them out varied.

This issue was closely bound to the issue of whether a view of Help limited to a node and a menu was sufficient. Designers concluded ([19][20][21]) that a one-line outline view and a view including the full text of the node should also be available. This discussion affected the evolution of the command language but limitation of resources prevented actually coding the two other views.

III NLS Subsystems

15 Query/HELP

In terms of the reference of the nodes in Help to one another, the database is a network, not a tree. An issue bound closely to the previous two was whether the user should be allowed to see the tree structure if he wants or be forced to see it. The influence of the Zog system [10], which the Software designers studied closely, pressed us to give the user commands that use the database's tree structure. Some users accustomed to information hierarchies became nervous when an item they had seen previously as a node appeared as a menu item later, down the tree as they see it; others found this outcome harmonious with browsing.

Bugging in DNLS is another related issue. Most discussants agreed that it should be possible to begin a search of Help by bugging a word in a node displayed. Not to do so would be a clear departure from NLS style. It was not so clear whether bugging a word in a menu item should provoke a search for that word or display the full text of the menu item. Again, the current command allows a place in its syntax for bugging but the operation has not actually been coded because of lack of resources.

Proposed command styles ranged from a "show item" command followed by a field to enter viewspecs (to specify menu vs outline vs full view) very much like the TNLS print statement command, to a command that could be executed entirely by pressing mouse buttons [23]. Those who suggested command modes relatively austere in feedback and with non-mnemonic command elements were no doubt influenced by the acquaintance of ARC with TECO [22], which has one-letter commands and the Whole Universe Catalog accessing system [11] which uses only mouse buttons.

A compromise was adopted as described in [24]. It resembles the "repeat mode" and "insert mode" of NLS discussed above and in (8d,) rather than the usual NLS pattern of a series of discrete, isolated commands. We see no evidence that this difference slows down people learning NLS with the help of Help. The command has proven reasonably simple to learn and use. It allows a user to search in terms of tree structure but does not depend on it.

Help as an Instructional Tool

As the system design evolved, we had increasing hopes for the use of Help as the primary teaching tool for naive NLS users. Those hopes have not been fulfilled. In practice, truly naive users are generally unable to learn from Help because of the technical vocabulary and because of the general awkwardness and discomfort with using a computer via even a relatively simple command language.

Help, however, has proven itself as an effective training tool for less naive users and has been used effectively by Office-1 Utility customers.

In the future, we hope to provide simpler answers to users identified as naive through their User Profile. For the present, we have tailored initial offline training and documentation to bring these users to the point at which they can successfully employ Help to explore and learn about other parts of NLS on their own.

III NLS Subsystems

15 Query/HELP

ADDITIONS AND MODIFICATIONS

The following additions to the Help System are anticipated in the existing designs:

- 1) Permit "bugging" of data in DNLS. This requires the creation of an additional data structure that contains the formatted information displayed to the user. The mechanism in NLS that permits interpretation of bug selection points into this structure.
- 2) Expansion to multifile databases and, in particular, the Resource Notebook and other tools in the NSW.
- 3) Implementation of the full Query Language.
- 4) Implementation of commands to gather information into other files.

ACKNOWLEDGMENTS

Jacques Vallee, now employed at the Institute for the Future, is responsible for the initial design and coding of the Query I language. Other people who participated in the design discussions for Help include Michael D. Kudlick, Richard W. Watson, Elizabeth J. Feinler, and N. Dean Meyer. Programmers involved include Diane Kaye and Elizabeth K. Michael.

REFERENCES

- [1] (15a1) Richard W. Watson, Robert E. Kahn. Notes from ICCG Planning Meeting 7 July. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 12-JUL-72. (11025.)
- [2] (15b2a) Walter Bass. First Thoughts on an Interactive Data Description Language. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 12-DEC-72. (13279.)
- [3] (15b2a) Harvey G. Lehtman. Notes on Visit of Hal Murray of CCA to ARC: Possible Interconnections between NLS and the Datacomputer. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 13-MAR-74. (22397.)
- [4] (15b2b) Elizabeth K. Michael Harvey G. Lehtman. Staged Forms System. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 6-FEB-73. (21808.)
- [5] (15e2a) Dirk H. van Nouhuys. An Introduction to the Current Status of the Help Data Base. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 13-SEPT-73. (19062.)
- [6] (15e2c) Michael D. Kudlick. Help Data Base Design. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 12-OCT-73. (19634.)

III NLS Subsystems

15 Query/HELP

- [7] (15e2e) Kirk Kelley. Structural Position of Lexicon in Help Data Base. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 15-OCT-73. (19675,) 15-10
- [8] (15e2e) Jeanne M. Beck. Updating Syntax, Function and Example Branch of Help. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 3-DEC-73. (20681,) 15-11
- [9] (15e2e) Jeanne M. Beck. Help Software Need. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 11-APR-74. (22745,) 15-12
- [10] (15e2f) (15e4h) A Newel, H. A. Simon, R. Hayes, and L. Gregg. Report on a Workshop in New Technology in Cognitive Research. Psychology and Computer Science Departments, Carnegie-Mellon University, Pittsburg, Pennsylvania, 7-JUNE-73. 15-13
- [11] (15e2f) (15e4j) Kirk Kelley. The Whole Universe Catalog. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. JUN-75. (24276,) 15-14
- [12] (15e3a) Jeanne M. Beck. TNLS Users' Guide. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 28-NOV-73. (19200,) 15-15
- [13] (15e3d) Charles H. Irby. I need Journal help. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 4-FEB-74. (21779,) 15-16
- [14] (15e3d) Susan R. Lee. Sometimes it's hard to get out of help. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 23-APR-74. (22799,) 15-17
- [15] (15e3d) Robert N. Lieberman. annoying loop in help menu system. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 23-APR-74. (22810,) 15-18
- [16] (15e3d) Robert N. Lieberman. help system : wrong info for menu item. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 24-APR-74. (22824,) 15-19
- [17] (15e3d) Beauregard A. Hardeman. FEB 24 - MAR 2 1974: A Week In Review. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 9-APR-74. (22674,) 15-20
- [18] (15e3d) Dirk H. van Nouhuys. Some Good Words for Help. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 2-DEC-74. (24643,) 15-21
- [19] (15e4g) Dirk H. van Nouhuys, Kirk Kelley. Four Help Show command Alternatives Necessary for NLS to be Self-Teaching. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 14-MAY-74. (22992,) 15-22

III NLS Subsystems

15 Query/HELP

- [20] (15e4g) Dirk H. van Nouhuys. Minutes of Documentation Meeting of 10-14-74: Status of Documentation , Plans for Introductory Hardcopy for Help, Plans for Something for Learners to Read. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 18-OCT-74. (2427,)
- [21] (15e4g) Dirk H. van Nouhuys, Jeanne M. Beck. Specifications for Help Command Language Functions. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 16-JUL-74. (22128,)
- [22] (15e4j) Phyllis Hauser. [Class session at BBN]. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 21-JUN-73. (17403,)
- [23] (15e4j) Dirk H. van Nouhuys. Against Command Words in HELP. For more Views.. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 20-MAR-74. (22441,)
- [24] (15e4k) Richard W. Watson. Suggested Changes to Help System Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 23-JUL-74. (22667,)

Chapter IV: WORKSHOP FOUNDATION

The Group Allocation System of ARC'S Time Sharing Resources (by Paul R Reeh)

INTRODUCTION

Early in 1973, ARC's computer services became so oversaturated that a user control system had to be implemented to allocate in a practical and systematic fashion the available computer resources to the many diverse users of ARC's computer services.

To accomplish this goal we have outlined a number of tasks, and have recognized a number of problem areas.

This system has now been operational for over a year on ARC's time sharing system. It has also been implemented on the NLS Utility that is being run by TYMSHARE Corporation, as well as on some other TENEX systems on the ARPANET. So far its implementation has been successful as it has allowed ARC to allocate its computer resources in a balanced fashion while still maintaining a flexible working environment.

The basic concept of the group allocation system is simple. The entire user population is partitioned into a set of functional user groups that are each allocated a quota of access slots to the available computer resources. This reduces the overall scheduling problem to a set of smaller scheduling problems and allows an effective control of the time sharing performance of the system by limiting the number of users who can be logged in simultaneously.

All unused access slots are pooled and made available to the entire user population under a lower priority system called the "off-quota" priority system. Under that system, any authorized user can utilize, on a first-come/first-served basis, an unused access slot as long as it is not reclaimed by a priority user who is entitled to an access slot on his group's quota. An "off-quota" user must release his slot when a priority user wants to log in -- usually, the one who has been online the longest. Otherwise, all off-quota users can utilize the system as long and as much as they need it.

Thus, by design, the group allocation system fulfills the following needs:

- 1) It guarantees adequate responsiveness of the time sharing system by appropriate control over the total number of users logged in at any time.
- 2) It guarantees management control over the allocation of the available computer resources to the various user groups and it provides a framework for making contractual arrangements for marketing purposes.

Preceding page blank

IV Workshop Foundation

16 Group Allocation

3) It reduces the overall scheduling problem to a set of smaller scheduling problems that are more tractable and that lend themselves much better to informal arrangements within groups. This preserves some of the flexibility needed for personal needs and variations in work requirements.

4) It offers the possibility of using all unused slots under a lower priority system and therefore avoids the potential waste of a more rigid allocation system.

THE GROUP ALLOCATION SYSTEM

The entire user population is partitioned into a small number of functional user groups in such a fashion that each member belongs to one group and one group only. All members of such a group should ideally have similar functions, e.g., be programmers, or staff members, or members of one same project. They should have approximately similar needs for computer resources and be granted similar access rights to these resources. All the members of a given group should also work closely together in order to be able to understand each other's working requirements and be sensitive to each other's actual constraints.

Either by negotiation or by managerial decision--or by subscription in a multient situation--each group is allocated a quota of the total number of available "access slots" to the computer resources assigned to users. These quotas can vary with the day of the week and the time of the day and can be renegotiated or changed when the circumstances demand it. However, the premium should be on stability, as it is much easier for both individuals and groups to plan ahead when their working environment is stable and predictable. Stability of group quotas fosters ease in planning and greatly reduces the uncertainty level of everyone concerned. A great deal of waiting and guessing can thus be eliminated and the many frustrating negative reinforcements that are inherent in unpredictable or unstable services can be avoided.

Provided he can log in under his group's quota and barring operational difficulties, a member of a given group is guaranteed unconditional usage of the available computer resources. Such a user is called a "quota" user. A quota user has total priority over all nonquota users. He can work online as long as he wants and use as much of the system as he needs as long as the group quota does not change and the informal group arrangements allow him to do so.

When all quotas are not filled and some empty slots remain available, the off-quota pool of unused slots is made available on a first-come/first-served basis to all users who request computer access, regardless of their group. They are informed at login time that they are on an off-quota status. This means that they can use the available slots as long as no regular quota users are reclaiming them. When a quota user is reclaiming one of these slots, the off-quota user who has been online the longest gets a message asking him to log out, within say five minutes, and if he does not comply he gets logged out automatically after that time is over.

It is important for every user to have easy access to the system for brief periods of time in order to be able to get a message, or send one, or print out a file, or for any other short task that needs to be done when a user has his working material online. The Express Login feature has been designed to allow him to log in for five minutes without having to log in under his quota or on

IV Workshop Foundation

16 Group Allocation

the off-quota priority. The command is ELOG instead of LOG to log in under that system. Only two users can be in ELOG simultaneously and therefore a user might have to wait a while before being admitted under ELOG. A queueing system has been considered but so far has not been implemented.

Change of priority status: A user's priority may change. This occurs when a quota user logs out. He is then replaced on the quota priority list by the user of his group whose name is found first on the off-quota list. Thus the priority status of the latter is changed and his name is dropped from the off-quota priority list.

Demonstration priorities: Another exception is the "visitor" priority. Under such a priority, any authorized personnel should always be allowed to log in for demonstration purposes whenever the need arises. This would allow easy access to a terminal without having to make special arrangements. This type of priority should have precedence over all off-quota priorities and, when the system is fully busy with quota users it should have precedence over a predetermined quota user, such as a staff user.

Partitioning of CPU time: In the NLS environment, DNLS users are using on the average 4% of the system when they are online and the TNLS users 2%. Thus there is no real need for partitioning the CPU time as long as users are limited by the scheduler's queueing system from using an inordinate amount of CPU time when the system is loaded. It is even very likely that the overall service would deteriorate if such a CPU time partitioning would be instituted as the average group populations are not large enough to absorb temporary high-CPU-time needs. The overall system would move even further into the unsteady state that is already hindering the performance under present conditions.

Nonpriority users: Some users might be authorized to use only off-quota slots. If they do not belong to any group, they can only log in in the off-quota mode and therefore the group allocation system gives us the possibility to have nonpriority users on the system.

Group status: To plan his work effectively schedule and to make it compatible with his group's schedule, a user needs specific information about the status of the system, about his group's utilization schedule and statistics, and about his current priority status. A command called GROUPST gives all this information to the users.

SUMMARY

In the group allocation system, all the computer services that are assigned to users are always available to the members of all groups.

When the system is not fully utilized by quota users, the slack is available on a first-come/first-served basis to whoever wants it or needs it. Thus when the total demand is less than the total number of slots available there are practically no access restrictions and the users can utilize the system as they please for as long as they want.

IV Workshop Foundation

16 Group Allocation

When the total demand exceeds the total number of slots available then the group allocation system decides the allocation priorities and actually manages the off-quota pool in the manner discussed above. 10010

Thus, the group allocation system is a partitioning system only for priority purposes, and not a partitioning system of the available time sharing resources per se. 10010

ACKNOWLEDGEMENTS 10010

The Group Allocation System was implemented by Donald C. Wallace and William R. Ferguson. 10010

NLS File System

(by Charles H Irby and Harvey G Lehtman)

INTRODUCTION

NLS operates on a hierarchical, random file system with several unique features evolved over the years that make possible the efficient online interaction used by the ARC community. Information stored within separate structure and data blocks aids in rapid movement within and between NLS files; a "partial copy" locking mechanism provides security against attempted modification of a file by more than one user at the same time and provides a high degree of backup security against system failure or user error. A technical description of the file system as well as discussions of motivating factors leading to the implementation are also discussed. The design of the file system provides room for further extensions, some of which are also examined.

Discussion of the hierarchical structure of NLS files at a user level, as well as a description of the user commands that permit movement through the files, may be found in [1].

GENERAL CONSIDERATIONS LEADING TO THE CURRENT DESIGN

The format and structure of NLS files were determined by certain design considerations:

It is desirable to have virtually no limit on the size of a file. This means it is not practical to have an entire file in core when viewing or editing it.

The time required for most operations on a file should be independent of the file length. That is, small operations on a large file should take roughly the same time as the same operations on a small file. The user and the system should not be penalized for large files.

In executing a single editing function, there may be a large number of structural operations.

A random file structure satisfies these considerations. Each file is divided into logical blocks that may be accessed in random order.

An early version of the file system was implemented on the XDS-940. Minor changes in the logical structure of the file system were made in the conversion of the system from the XDS-940 to the PDP-10 for two reasons:

- 1) The current ARC programming language, L10, is more powerful than the several languages it replaces, MOL and the SPLs. L10 permits special purpose constructions anywhere in its code. It is a higher level language and provides greater compiler optimization.
- 2) An effort has been made to further modularize the functions within the system to ease development by a team of programmers.

IV Workshop Foundation

17 NLS File System

RELIABILITY AND THE NLS FILE SYSTEM

The reliability and security of file data both against system crashes and in face of the possibility of attempted simultaneous modification by more than one user were central goals in the design of the NLS file system. An attempt was made to minimize the amount of work which would be lost due to both hardware and operating system difficulties.

Unlike the sequential file systems of some editors which require copying large sections of a file whenever an edit is made, NLS modifies copies of pages in which structural or data changes are made: all data in the original file is secure; and a minimum of unaffected data is copied. Still other editors maintain recent changes in a dynamic buffer which may not be incorporated into the file in the event of a system crash; in NLS, barring a major hardware collapse, all changes other than those specified by the command being processed are present in the copied pages. Again, the original file is untouched.

Other techniques to assure high reliability have been used such as organizing the code and sequence of operations in a way to minimize time windows of high vulnerability.

An important problem in an online team environment such as that at ARC involves group collaboration on the same data files. The current file system permits multiple readers and a single writer to a file. The person obtaining write access to a file locks it in a manner described below; no other user is then permitted to write on the file, though they may read the original material. Readers without write access do not see the changes of the user currently editing the file until the file is explicitly "updated," causing the incorporation of edits and the unlocking of the file. Thus there can be no conflict between the edits of more than one writer.

Details on the partial copy locking mechanism which implements these features of the NLS file system are discussed below in section (17f7)

EXTENSIONS TO THE CURRENT FILE SYSTEM

There has been discussion within ARC concerning possible extensions to the current NLS file system. An example is the design of a file system that has property lists at each node. Instead of the current system in which there is a one-to-one correspondence between ring elements and (textual) data elements associated with a particular node in the file. In the property list scheme there could be a list structure of data nodes pointed to by one or more structure elements. There would be no restriction on the types of data node: for instance, graphic or numerical information may be possible as well combinations of data types within a single node. Such a scheme would prove useful in a cataloging system.

The possibility of implementing the NLS file system on the CCA Datacomputer in the Datalanguage has also been discussed.

IV Workshop Foundation

17 NLS File System

SHORT TECHNICAL OVERVIEW

This section gives a brief overview of the implementation of NLS files. For more detail see section (17g).

Block Header and Types of Blocks

An NLS file is made up of a file header block; and up to a fixed number (currently 465) of 512-word (= equals one TENEX page) structure blocks (up to 95); and data blocks (up to 370).

There are several types of blocks, each with its own structure:

File header block--always page 0; contains general information about the file.

Structure (ring) blocks--contain ring elements that implement the NLS structure: there currently may be a maximum of 95 of these blocks, each containing 102 five-word ring elements. They may appear in file pages 6 through 100.

Data blocks--contain the data (currently text) of NLS statements: each data block has Statement Data Blocks (SDBs) that have five-word headers followed by text strings. There currently may be a maximum of 370 data blocks. They may appear in file pages 101 through 471.

Miscellaneous blocks--not used in the current implementation.

File Header Block

In each file there is a header block that contains general information about that particular file. The header block remains in memory while the file is in use.

The file header is read into core by the procedure (nls, ioexec, rdhdr). This procedure checks for the validity of certain keywords. If the file is locked and has a partial copy, the header is read in from the partial copy. If the partial copy header block is invalid in the key spots, the file is unlocked and the header read in from the original file. If that is bad, the file may be initialized.

RDHDR sets the value of file head[fileno] where fileno is the NLS file number of the file (an index into the file status table that provides, among other things, a correlation between JFNs for the original and partial copy and the single NLS file number).

Procedures in (nls, filmnp.) are responsible for reading, manipulating, creating, garbage collecting, and storing into ring blocks and ring elements within those blocks, and data blocks and statement data blocks within them.

IV Workshop Foundation

17 NLS File System

Structure Blocks -- Ring Elements

Conceptually an NLS file is a tree. Each node has a pointer to its first subnode and a pointer to its successor. If it has no subnode, the sub-pointer points to the node itself. If the node has no successor, the successor pointer points to the node's parent. Each node is currently represented by a ring element. These ring elements point in turn to the associated data block.

Structure blocks contain five-word ring elements with a free list connecting those not in use.

Data Block -- Statement Data Blocks

Data blocks are composed of variable sized blocks called Statement Data Blocks (SDBs) that contain the text of NLS statements. Each SDB has a five-word header with node related information followed by the text made up of 7-bit ASCII characters packed five to a word. New SDBs are allocated in the free space at the end of a data block. SDBs no longer in use (because of editing changes) are marked for garbage collection when the free space is exhausted.

String Identifiers (STIDS) and Text Pointers

A string identifier (STID) is a data structure used within NLS to identify strings (possibly within NLS statements).

If the string is in an NLS statement, the STID contains a file identifier field (STFILE) and a ring element identifier (STPSID).

The presence of a file identifier within the STID permit all editing functions to be carried out between files.

Procedures in (nls, filmnp,) and (nls, strmp,) permit traversal through the ring structure of a file given an STID. See, for example, (nls, filmnp, getsuc), which gets the STID of the successor of a statement; see also (nls, filmnp, getsdb), which returns the STPSDB for the statement whose STID is provided as an argument. (An STPSDB has, like an STID, a file number field and a pointer to a data block, a STPSDB).

Text pointers are two-word data structures used with the string analysis and construction features of L10. They consist of an STID and a character count.

Locking Mechanism -- Partial Copies

The NLS file system under TENEX provides a locking mechanism that protects against inadvertent overwrite when several people are working on the same file. Once a user starts modifying a file, it is "locked" by him against changes by other users until he deems his changes consistent and complete and issues one of the commands: Update File, Update File Compact, or Delete Modifications, which unlock the file. A user can leave a file locked indefinitely--this protection is not limited to one console session.

IV Workshop Foundation

17 NLS File System

When a file is locked (is being modified), the user who has modification rights sees all of the changes that he is making. However, others who read the file will see it in its original, unaltered state. If they try to modify it, they will be told that it is locked by a particular user. Thus the users can negotiate for modification rights to the file.

This feature is implemented through the use of flags in the status table in the File Header and through the partial copy mechanism.

All modifications to a file are contained in a partial copy file. These include modified ring elements and data blocks.

Any file page that is to be and that is not in the partial copy (discovered through a write pseudo-interrupt) is copied into the partial copy. All editing takes place there. The TENEX user-settable word in the FDB (TENEX file data block) for the original file contains locking information.

The NLS Update file command merely replaces those structure and data pages in the original file that have been superseded by those in the partial copy, unlocks the file, and deletes the partial copy. For Update file old, this is done in the original file; for Update to new version, the pages are mapped to a new file from the original or partial copy where necessary. The Update file compact command garbage collects unused space; the update file command does not.

Core Management of File Space

When space is needed for more data, the following steps are taken, in order, until enough is found to satisfy the request (See (nls, filmnp, nwrngb), (nls, filmnp, newsdb), and related routines):

- 1) Core-resident pages are checked for sufficient free space.
- 2) Other pages are checked for free space. If one has sufficient space, it is brought in.
- 3) If garbage collection on any page in the file will yield a page with sufficient free space, then the page that will give the most free space is brought into core and garbage-collected; otherwise a new page is created.

DETAILED TECHNICAL DISCUSSION

Note on Fields in NLS Records and Other L10 Language Features

Several parts of this section are taken directly from record declarations in the code of the NLS system written in the L10 programming language.

Record declarations in the L10 language serve as templates on data structures declared in the system. Byte pointer instructions are dropped out by the compiler permitting access to

IV Workshop Foundation

17 NLS File System

specified parts of the array. Multiword records are filled from the lowest to the highest address of the array. Within words, bits are allocated from the first bit on the right. If several fields fail to fill a 36-bit word and the next field definition would go over the remaining bits in the word, the field is allocated in the next word available.

Example:

Bit 0 is the leftmost bit in the word; bit 35 the rightmost. Suppose there is a record declaration of the form:

```
(newrecord) RECORD % A two word record %
field1[10], %bits 26 through 35 (rightmost) of first word%
field2[25], %bits 1 through 25 of first word %
field3[15], %bits 21 through 35 of second word (field would not fit in remainder of first
word%
DECLARE array[2];
```

There may be code within a program of the form:

```
variable = array.field2;
array.field3 = 20;
```

In L10, false is zero and true is nonzero.

See the L10 manual for further information.

Block Header and Types of Blocks

An NLS file is made up of a file header block and up to a fixed number (currently 465) of 512-word (= one TENEX page) structure blocks (up to 95) and data blocks (up to 370).

Each block has a two-word header telling the type and giving the file page number and an index into a core status table. The record declaration from (nls, utility,) follows:

```
(fileblockheader) RECORD %fbhdl = 2 is length%
fbnull[36], %unused%
fbind[9], %status table index%
fbpnum[9], %page number in file of this block%
fbtype[5], %type of this block (types declared in (nls, const.))
hdtyp = 0 = header
sdbtyp = 1 = data
rngtyp = 2 = ring
jnktyp = 3 = misc (such as keyword, viewchange etc.)%
```

There are several types of blocks, each with its own structure.

File header block--always page 0; contains general information about the file.

IV Workshop Foundation

17 NLS File System

Structure (ring) blocks--contain ring elements that implement the NLS structure: there currently may be a maximum of 95 of these blocks, each containing 102 five-word ring elements. They may appear in file pages 6 through 100. 17-260

Data blocks--contain the data (currently text) of NLS statements: each data block has statement data blocks (SDBs) that have five-word headers followed by text strings. There currently may be a maximum of 370 data blocks. They may appear in file pages 101 through 471. 17-263

Miscellaneous blocks--not used in the current implementation. 17-264

File Header Block 17-265

In each file, there is a header block that contains general information about that particular file. The header block remains in memory while the file is in use. 17-266

FILE HEADER CONTENTS (taken from (nls, data,)): 17-267

DECLARE EXTERNAL

%...file header...%

% DONT CHANGE THE ITEMS IN THE HEADER %
filled[5].

% these extra words may be taken for additions to header%

fcredt, % file creation date--TENEX gtad jsys internal format %
nlsvwd = 1,

% nls version word; changed when NLS file structure changes %

sidcnt, %count for generating SID's%

% An S (statement identifier) should not be confused with PSIDs (see below). The SID is uniquely generated for each statement in a file and is not reused if a statement is deleted; it is unchanged if a statement is moved. It may be used by a user for accessing particular statements in a file without worrying about changes because of additions or deletions (as is the case with statement numbers). The sidcnt field in the header is increased by one as statements are created. The value is stored in the RSID field of the ring element (see description below). %

finit,

% initials of user who made the last write (by updating or outputting the file)--see DATA BLOCK description below for explanation of initials %

funo, % user number (file owner) %

lwtim, % last write time--TENEX internal JSYS gtad format %

namdl1, % left name delimiter default character %

namdl2, % right name delimiter default character %

rngl,

% upper bound on ring (structure) file blocks used %

dtbl, % upper bound on data file blocks used %

rfbs[6].

% start of random file block status tables (see description below) %

rngst[95], % ring block status table %

dtbst[370], % data block status table %

mkrtxn = 20, % marker table maximum length %

mkrtbl, % marker table current length %

IV Workshop Foundation

17 NLS File System

mkrtb[20], % marker table %
 % Markers provide an alternative form of NLS addressing; see NLS Users Guide for description %
 filhde; %end of the file header%

Notes on File Header

The file header is read into core by the procedure (nls, ioexec, rdhdr). This procedure checks for the validity of certain keywords. If the file is locked and has a partial copy (the file that includes current modifications--see below), the header is read in from the partial copy. If the partial copy header block is invalid in the key spots, the file is unlocked and the header read in from the original file. If that is bad, the file may be initialized. RDHDR sets the value of filehead[fileno] where fileno is the NLS file number of the file (an index into the file status table which provides, among other things, a correlation between JFNs for the original and partial copy and the single NLS file number; see description of the file status table below.)

(nls, ioexec, setfil) initializes a file header.

It should be noted that fields within a file header are accessed by full word indexing rather than by record pointers for speed. Thus we have the following typical code (from (nls, utility, esc)) that reads the default name delimiters from an NLS file header:

```
.
.
.
ELSE IF rplsid.stpsid = origin THEN
BEGIN %use standard delimiters for that file%
fhdlc = filehead[rplsid.stfile] - $filhed;
dlleft = [fhdlc + $namdl1];
dlrght = [fhdlc + $namdl2];
END
.
.
.
```

Also, code from (nls, ioexec, rdhdr) that gets the address of the word in core that contains the nls version word for the file whose header has been read in order to check its validity:

```
.
.
&vwd = (header - filhdr(fileno)) - $filhed + $nlsvwd;
filehead[fileno] = header;
.
.
```

The file header is initialized by (nls, ioexec, rdhdr) which fills up contiguous words declared in (nls, data,) and then moves the contents of those words to page zero of the file.

IV Workshop Foundation

17 NLS File System

Procedures in (nls, filmp) are responsible for reading, manipulating, creating, garbage collecting, and storing into ring blocks and ring elements within those blocks, and data blocks and statement data blocks within them.

Random File Block Status Table Entries in File Header

The random file block status tables appear in the file header. There is one word per ring block or data block page. Each entry contains the following: record declaration and comments from (nls, utility.).

```
(rfstr) RECORD % Random file block status record (The entry will be equal to 0 if the
page (i.e., block) in the file is unallocated. Otherwise, the entry will be an instance of the
following record.)%
rfexis[1], %true (i.e., nonzero) if the block exists in the file%
rfpart[1], %true if block comes from partial copy%
%Whether page has been modified by a user.
(rfpart will be true in that case.)%
rfnull[2], %unused%
rfused[10], %used word count for the block%
%Current used word count (may be used to calculate post-garbage collection free space
count.)%
rffree[10], %free pointer for the block%
%Free space count (for data block)
Pragarbage collection free space count.
Free list pointer (for ring block)%
rfcore[9], %0 then not in core, else page index%
```

Notes on Random File Block Status Tables

The table RFBS in the file header is broken into two sections, each of which contains a collection of records of the above type. The first section includes RNGM entries from RFBS[RNGBAS] up to and including RFBS[RNGBAS+RNGM-1] and contains information about the ring blocks in the file. (RNGBAS is currently 6 and is the first page in a file that may be a ring block; RNGM is currently 95 and is the maximum number of ring blocks permitted.)

The second section includes DTBM entries from RFBS[DTBBAS] up to and including RFBS[DTBBAS+DTBM-1] and contains information about the data blocks in the file. (DTBBAS is currently 101 and is the first page in a file that may be a data block; DTBM is currently 370 and is the maximum number of block blocks permitted.) The entry RFBS[RNGBAS+i] may also be referenced as RNGST[i], likewise RFBS[DTBBAS+i] may be referenced as DTBST[i]. The index in RFBS of a block is the actual page number of the block in the file.

A pointer to an SDB (PSDB) consists of a nine-bit data block number in the range [0,DTBM) and a nine-bit displacement from the start of the block. The variable DTBL is maintained in each file header as the current upper bound on allocated data blocks for that

IV Workshop Foundation

17 NLS File System

file. This is used to limit the search for a location for a new SDB. The variable DBLST contains the index of the block from which an SDB was last allocated or freed. 17-151

A pointer to ring element (PSID) consists of a nine-bit ring block number in the range [0,RNGM) and a nine-bit displacement from the start of the block. The variable RNGL is maintained in each file header as the current upper bound on allocated ring blocks for that file. This is used to limit the search for a location for a new ring block. The variable RNGST contains the index of the block from which a ring was last allocated or freed. 17-152

Structure Blocks -- Ring Elements 17-153

These blocks contain five-word ring elements with a free list connecting those not in use. 17-154

```
(ring) RECORD %ringl is length% % from (nls, utility) % 17-155
  rsub[18], %psid of sub of this statement% 17-156
  % A pointer to the first substatement of this statement %
  rsuc[18], %psid of suc of this statement% 17-157
  % A pointer to the successor of this statement (to the parent if no successor) %
  rsdb[18], %psdb of sdb for this statement% 17-158
  % Pointer to the data block that contains text for this statement. %
  rinst1[7], %DEX interpolation string-- scratch space% 17-159
  % Information in scratch fields may be reset and used by other subsystems such as DEX.
  No other assumption concerning their contents should be made. %
  rinst2[7], %DEX interpolation string-- scratch space% 17-160
  rdu.mmy[1], %DEX dummy flag-- scratch space% 17-161
  repet[3], %DEX repetition-- scratch space% 17-162
  rhf[1], %head flag, true (= 1) if this is head of plex% 17-163
  rtf[1], %tail flag, true if tail of plex% 17-164
  rnamef[1], %name flag, true if statement has a name% 17-165
  rnull[2], %unused% 17-166
  rnameh[30], %name hash for this statement% 17-167
  % hash algorithm may be found in (nls, utility, hash) %
  rsid[30], %statement identifier% 17-168
  % See SIDCNT description in file header above. %
  %although only need four words, use five so that have room to grow% 17-169
```

PSIDs and PSDBs are pointers to other ring or data blocks in a file. They have two nine-bit fields: one (stblk) is a block index; the other (stwc) is a word displacement within that block. Procedures in (nls, filmnp,) permit the traversal of a file's structure. 17-170

Given an STID (see below), one may use the primitive procedures in (nls, filmnp,)--e.g., (nls, filmnp, getsuc)--or the more elaborate procedures in that file--e.g., (nls, filmnp, getnxt)--to move around to related ring elements and retrieve or change (display or edit) relevant data. 17-171

There are two "fixed" values for PSIDs for special statements: 17-172

The PSID of the origin statement is always 2.

IV Workshop Foundation

17 NLS File System

The entire STID (and hence PSID) of the end of a file is endfil (= -1), which does not correspond to any real statement in the file, but which is returned by the "get" procedures in flmnp and strmnp to indicate the end has been reached or an error has been found.

Some other conventions implemented in the file structure make possible special features in NLS:

The successor of a statement with no real successor is its "parent."

The substatement of a statement with no sub is itself.

The origin is at a unique level; thus statement 1 is the sub of the origin.

Data Block -- Statement Data Blocks

Data blocks are composed of variable-sized blocks called Statement Data Blocks that contain the text of NLS statements. Each SDB has a five-word header with the following information followed by the text made up of seven-bit ASCII characters packed five to a word. New SDBs are allocated in the free space at the end of a data block. SDBs no longer in use (because of editing changes) are marked for garbage collection when the free space is exhausted.

```
(sdbhead) RECORD %sdbhdi is length% % from (nls, utility,) %
sgarb[1], %true (non-zero) if this sdb is garbage, i.e., no longer used%
slength[9], %number of words in this sdb%
schars[11], %number of characters in this statement%
slnmdl[7], %left name delimiter for statement%
srnmdl[7], %right name delimiter for statement%
spsid[18], %psid of the statement for this sdb%
%Pointer to ring element.%
sname[11], %position of character after name%
% This is i for a statement with no name. Thus if the text of the statement were:
(author) The person who
and the name delimiters were '(' and ')', the value of this field would be 9. %
stime[36], %date and time when this SDB created%
% This is stored in the TENEX internal format; see the TENEX JSYS manual, gtd jsys
%
sinit[21], %initials of user who created this SDB%
% This is stored in 5-bit characters to permit NLS user ids of four characters and still
maintain compatibility with files created when only three-character ids were available.
This kluge still requires translation of old-style ids, but at least both old and new style
fit in the same space. [See (nls, flmnp, getint) and (nls, flmnp, trnsint)%
%sgarb and slength must be in the first word of the header for newsdb% %although only
need four words, use five so that they will have room to grow%
```

String Identifiers and Text Pointers

A string identifier (STID) is a data structure used within NLS to identify strings (possibly within NLS statements).

IV Workshop Foundation

17 NLS File System

If the string is in an NLS statement, the STID contains a file identifier field (STFILE) and a ring element identifier (STPSID). (See PSID description above under ring elements.)

The presence of a file identifier within the STID permit all editing functions to be carried out between files.

Procedures in (nls, filmnp,) and (nls, strmnp,) permit traversal through the ring structure of a file given an STID. See, for example, (nls, filmnp, getsuc), which gets the STID of the successor of a statement; see also (nls, filmnp, getsdb), which returns the STPSDB for the statement whose STID is provided as an argument. (An STPSDB has, correspondingly to an STID, a file number field and a pointer to a data block, a STPSDB).

Text pointers are used with the string analysis and construction features of L10. They consist of an STID and a character count.

Other Relevant Arrays

The following arrays are used in system core and file management. They are described here to facilitate the study of the NLS file-handling code.

Filehead

An array of pointers (each contained in a single word) to the file headers of files currently in use is FILEHEAD. At present, up to 25 files (and their partial copies, if any) may be open simultaneously.

CORPST and CRPGAD

The array CORPST provides the correspondence between the 100 (octal) pages in core reserved for file pages and user program buffer and the pages in files that are currently loaded into core. (This is really a maximum of 100 octal since the user program buffer may be enlarged into this area; the maximum is given by $RFPMAX - RFPMIN + 1$.)

(corpgr) RECORD %one word. core page status record. gives status for a given core page for random files.%

ctful[1], %true if the page is in use%

ctfile[4], %file to which the page belongs; an NLS file number%

ctpnum[9], %page number within the file%

ctfroz[3]; %number of reasons why frozen (locked into core because of some current NLS system need-- editing is in progress on a statement, a statement is being displayed, etc) %

The array CORPST is the core page status table and is made up of instances of the above record. (RFPMAX (current user program buffer size) gives the number of core pages that may contain file pages. The core pages are located at positions indicated by the array CRPGAD (core page address). CORPST is indexed by numbers in the range (RFPMIN, RFPMAX). The elements in this array are actual addresses. The starting location of page

IV Workshop Foundation

17 NLS File System

k is given by `crpgad[k]`. `RFPMIN` is initialized to be 5; four pages are initially allocated for a user program buffer. See (`nls`, `usrpgm`, `gpbsz`) for the procedure that changes these limits.

FILST

An NLS file number provides an index into the `FILST`, the file status table. This 100-word array is made up of 25 four-word entries and contains the following information for files of interest that have NLS file numbers at any time (these may or may not at that time be open; they do, however, have JFNs.) The information comes from the record declaration in (`nls`, `utility`):

```
(filstr) RECORD %File status table record.  entry length = filstl = 4, max no. entries =
filmax = 25%
flexis[1], %true: entry represents an existing file%
flhead[9], %crpgad index of the file header%
flbrws[1], %true: file in browse mode%
fllock[1], %This file was locked by another user when loaded%
flpread[1], %PC read only--write open failed (openpc)--see discussion of partial copies
below%
flaccm[8], %file access mask%
% Used to tell whether or not the file may be written on by the current user.  Used
primarily for files such as those in the Journal that are read-only to most users. %
flrno[12], %directory number for the original file%
flpart[18], %JFN for the partial copy%
flbpart[18], %JFN for the browse partial copy%
florig[18], %JFN for the original file%
flastr[18], %address of the file name string%
flpest[18], %address of partial copy name string%
flbpest[18], %address of browse partial copy name string%
```

REFERENCES

- [1] (17b2) Douglas C. Engelbart and Staff of ARC. Computer-Augmented Management-System Research and Development of Augmentation Facility [Final Report]. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. APR-70. (5139.)

Software Engineering (by Harvey G Lehtman and Kenneth E Victor)

FRAMEWORK: THE SOFTWARE DEVELOPMENT PROBLEM

Many observers have become concerned recently about the inefficiency in software production. Overruns in cost and delivery time are common, quality is low, and maintenance cost is often greater than the cost of the original development.

A study of the software production needs of the Air Force cited in Boehm [2] estimated the 1972 Air Force software costs to have been between \$1 and \$1.5 billion per year while hardware costs were in the range of \$0.3 to \$0.4 billion. The ratio of software to hardware costs is predicted to increase in the next few years until it reaches a level of about 9 to 1 in the 1980s as hardware costs continue to go down and the costs of personnel increase.

Typical of the recently mounting interest in software production was a symposium on "The High Cost of Software" held at the Naval Postgraduate School in September 1973 [1]. Those attending the conference agreed that direct and indirect software costs are unnecessarily high and are growing rapidly. They discussed the following general problem areas in controlling the cost and increasing the quality of large-scale software production in the Federal Government:

- 1) Programmer management and organization: Tools for coordinating and controlling resources and tasks in the design, development, testing, documentation, release, and maintenance of software are not widely used; unambiguous means of specifying requirements for software and effective bookkeeping facilities for controlling an inventory of programs are needed.
- 2) Design: Unambiguous languages for software designs are desirable; effective recorded dialog is essential to minimize duplication or contradiction of previous design decisions.
- 3) Implementation and debugging: Currently, offline preparation of code, batch compilation, offline debugging techniques, and the use of poorly designed languages and programming methods are widespread, neglecting the major advances in productivity made possible by the use of online tools, and structured programming methods.
- 4) Documentation: Effective documentation methods must be developed and used since undocumented programs are virtually impossible for other programmers to understand and maintain.
- 5) Analysis: Metering and analytic tools for discovering bottlenecks within produced codes are essential to ensure software quality.
- 6) Quality control: Better methods for expressing or checking the correctness of a program with respect to its intentions or requirements would be beneficial.

IV Workshop Foundation

18 Software Engineering

7) Maintenance: Modifications, including bug fixes and upgrading of capabilities, are often difficult to carry out due in part to difficulties in coordination, documentation, and analysis.

Many of these problems have been successfully approached and solved, and the solutions used, in research environments (e.g., online code preparation and debugging and the use of structured programming techniques). However, the resultant tools and techniques are not currently in wide use in large software production groups.

Throughout its history, ARC has taken steps to augment software engineers-- those people concerned with software production, including programmers, software production managers, and technical writers. The developments described below are part of our ongoing activity to create a full and balanced set of tools, techniques, methods, and principles to aid those users.

SOLUTIONS: ARC EXPERIENCE WITH SOFTWARE ENGINEERING TOOLS AND TECHNIQUES

Programmer Management and Organization

To assist in the coordination and control of programming tasks, we have adopted a programming team approach. Each team is responsible for a specific programming task and consists of one or more programmers with one of them being the task pusher--the person with the responsibility for completing the task. A programmer is usually on more than one team at any time, and may be the pusher of one team, and a member of another team. Teams frequently interact with each other, and one team may subcontract with another team for subtasks.

The source code for the large NLS is divided into 50 files. Work on the system is currently carried out by ten programmers who often have reason to edit the same file. The Partial Copy locking mechanism controls editing access to files and prevents our tripping over changes made by each other. (This mechanism is described in Section 17f7.)

A status file is to record changes to the system and to note when new running systems are made available to general users. Backup systems are also noted in this file. (Development takes place in an experimental system which is used by programmers until it is ready to be put out to the general APC community. These evolutionary steps in NLS developments take place approximately on a weekly basis.)

Programmers make frequent use of the signature facility of NLS to discover the originator or most recent modifier of code.

Any change to the content of NLS statements is automatically associated with the identifying initials of the user making the change and the date and time of the edit. A special filter may be used to turn these signatures on or off. Additionally, elements in content pattern filters permit selective display of statements edited by particular users, or statements edited before or since particular dates.

IV Workshop Foundation

18 Software Engineering

Design

: 8 b 2

The ARC Journal and other Dialog Support tools (discussed in section 13) have assisted in controlling, recording, and cataloging documents for the design process from the conception of system needs and possibilities through initial requirements specification to final release. Continuing review of the implementation and use of new subsystems is also recorded in the Journal.

: 8 b 2 a

Implementation and Debugging

: 8 b 3

Languages

: 8 b 3 a

Introduction

: 8 b 3 a 1

ARC currently makes use of several languages created at the Center: the Tree-Meta compiler-compiler system, which is used to generate compilers and has been used to bootstrap compilers onto different computers; the L10 programming language, which is used to write NLS programs; and ML, which is used to describe user interactions with NLS.

In addition, Tree-Meta has been used to develop an interpreter for the Output Processor directive language.

Tree Meta

: 8 b 3 a 2

Tree Meta is a metacompiler system for context-free languages developed at ARC. The parsing statements of the metalanguage resemble Backus-Naur Form with embedded tree-building directives. Unparsing rules include extensive tree-scanning and code-generation constructions. All compilers produced by the system are single-pass compilers that produce loadable binary files.

A metacompiler, in the most general sense of the term, is a program that reads a metalanguage program as input and translates that program into a set of instructions. If the input program is a complete description of a formal language, the result of the translation is a compiler for the language.

Tree Meta is built to deal with a specific set of languages and an even more specific set of users. There is no attempt to design universal languages, or machine-independent languages, or to achieve any of the other goals of many compiler-compiler systems.

A version of Tree Meta was discussed in an appendix to the Rome Report of April 1968 [4]. Since that time, the syntax has been expanded and the system made more flexible. A new Tree-Meta report [3] includes a formal description of the Tree Meta language.

IV Workshop Foundation

18 Software Engineering

L10

18b3a3

NLS on the PDP-10 is written in the L10 programming language, an ALGOL-like language that has some high-level special-purpose features for string analysis and manipulation, and for interacting with NLS users.

Also among the constructs in L10 are flexible CASE statements and several types of looping facilities which make possible the creation of code essentially free of GOTO elements and hence more immediately understood and controlled. A SIGNAL creation and trapping facility permits clean error recovery. Coupled with the special debugging tools described below, the language offers extraordinary power for producing clearly structured code.

The formal description of the L10 language may be found in [9]. A less formal discussion oriented toward novice user programmers may be found in [10].

CML

18b3a4

CML is a language developed at ARC for the specification of user interactions with NLS. For a more detailed description of CML, see section 9.

System Catalog and Tools for Moving within the Source Code

18b3b

We use the NLS Jump and Jump Link commands in conjunction with level clipping and line truncation to move around within source files to examine various pieces of code. An automatically generated system catalog, SYSGD, is a useful aid in moving through code to find particular procedures. Its use is also illustrated.

18b3c

The first series of illustrations (Figures 1 through 4) shows online examination of a typical L10 source code file. Note the use of level truncation and commenting to zero in on the desired procedure.

18b3d

The second set of photographs (Figures 1A through 1D) demonstrates another use of NLS structure in code to clarify the fairly complex structure of some nested IF statements.

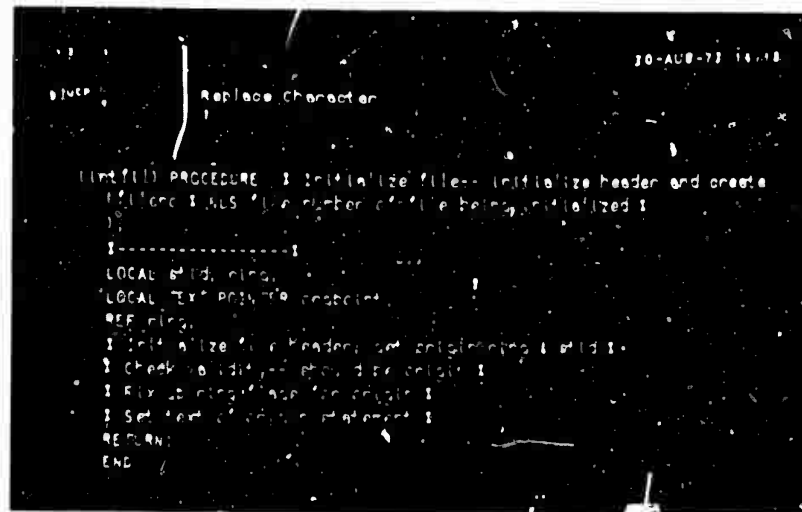
18b3e

Figure 1. Typical source code file--truncated view. The file, IOEXEC, handles NLS's interface with the TENEX file system. This view shows top-level statements which are L10 comments; hidden at lower levels is code which implements the functions described. PAGE 10

Figure 2. The same file after jumping to a particular function branch and opening one more level. Visible are the first lines of procedures with simple descriptive comments extractable by a program which produces the system catalog, SYSGD.

IV Workshop Foundation

18 Software Engineering



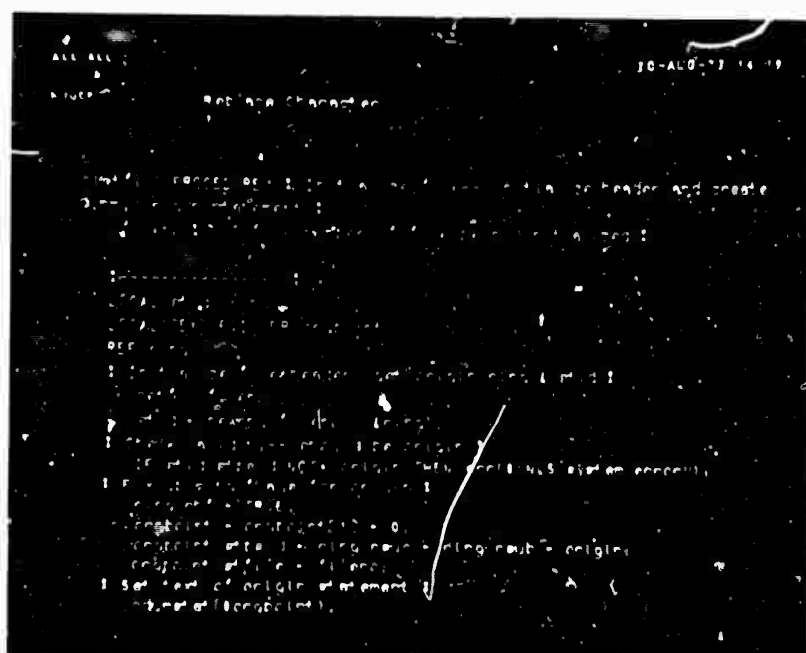
```

10-AUG-73 14:18
Replace Character

INTFIL PROCEDURE 1 Initialize file-- Initialize header and create
1 file and 1 NLS file number of file being initialized 1
1-----1
LOCAL MTD, nmg,
LOCAL TEXT POINTER, endpoint,
REF nmg,
1 Initialize file headers and originating 1 MTD 1
1 Check to identify should be origin 1
1 Fix up origin base for origin 1
1 Set text of origin statement 1
RETURN
END
  
```

Figure 3. The procedure INTFIL with one more level displayed. Hidden beneath the comments is the code described.

18b1b6



```

ALL ALL
10-AUG-73 14:19
Replace Character

1 Initialize file-- Initialize header and create
1 file and 1 NLS file number of file being initialized 1
1-----1
LOCAL MTD, nmg,
LOCAL TEXT POINTER, endpoint,
REF nmg,
1 Initialize file headers and originating 1 MTD 1
1 Check to identify should be origin 1
1 Fix up origin base for origin 1
1 Set text of origin statement 1
RETURN
END
  
```

Figure 4. The beginning of the INTFIL procedure with all levels visible.

18b1b7

Figure 1A. A truncated view of a nested IF statement.

Figure 1B. The preceding code with one more level shown.

page 159

IV Workshop Foundation

18 Software Engineering

```

ALL
30-AUG-73 15:04

hjump Insert Character

IF f1 = 'x' AND f1 = 'orig' THEN
  IF f1 = 'x' OR f1 = 'orig' AND f1 = 'name' THEN
    BEGIN
      IF f1 = 'x' THEN Insert new version with write access!
      BEGIN
        IF f1 = 'x' THEN
          END
        ELSE
          Insert new version with write access!
          BEGIN
            IF NOT writeable f1 = 'x' THEN
              IF NOT writeable f1 = 'orig' AND f1 = 'name' THEN
                IF NOT writeable f1 = 'orig' AND f1 = 'name' THEN
                  THEN
                    newfile = f1 = 'orig'
                    IF NOT indrdr f1 = 'x' THEN end(1)
                    IF f1 = 'x' empty THEN
                      END
                    I get partial copy header address 1
                    fhd = f1 = 'x' f1 = 'x'

```

Figure 1C. Yet another level visible.

18b3b10

```

ALL
30-AUG-73 15:04

hjump Insert Character

IF f1 = 'x' AND f1 = 'orig' THEN
  IF f1 = 'x' OR f1 = 'orig' AND f1 = 'name' THEN
    BEGIN
      IF f1 = 'x' THEN Insert new version with write access!
      BEGIN
        IF f1 = 'x' THEN
          BEGIN
            IF NOT writeable f1 = 'x' THEN
              IF NOT writeable f1 = 'orig' AND f1 = 'name' THEN
                end(1)
              IF NOT writeable f1 = 'orig' AND f1 = 'name' THEN
                THEN
                  BEGIN
                    newfile = f1 = 'orig'
                    IF NOT indrdr f1 = 'x' THEN end(1)
                    IF f1 = 'x' empty THEN
                      END
                    I get partial copy header address 1
                    fhd = f1 = 'x' f1 = 'x'

```

Figure 1D. All levels visible.

18b3b11

In addition, we have a user program that will generate a cross reference file, SYSGD (See Figures 5 through 7, following.) This file can be used in conjunction with the new PLS command, Jump Name External, to locate, online, procedures held in separate source files.

18b3b12

IV Workshop Foundation

18 Software Engineering

The command "Jump to Name External" may be used to jump to a named statement not within the current file. When this command is executed, a default system catalog file (specified in the user profile, described in section [yyy]), is searched for a statement of that name. An NLS link points to the file in which the target resides. SYSGD is typical of such system catalogs. While reading code online, a programmer may issue the `Jump to Name External` command, point at a word (perhaps a procedure call) on his screen, and have the appropriate target displayed.

```

30-AUG-73 14:27
NLS>
Replace character

<NLS>SYSGD.NLS;22; 26-JUL-73 08:56 NDM ;
(a1) (nls,utility.)
(a2) (nls,utility.)
(a3) (nls,utility.)
(a3ddrec) (nls,colort,a3ddrec) 7C
(a4) (nls,utility.)
(abort) (nls,auxcod,abort) 3D
(abs(g) (nls,const.)
(accctyp) (nls,loexec.)
(accum1) (nls,nddt.)
(accvele) (nls,calo,accvele) 5E
(adedoa) (nls,utility.)
(adedf) (nls,utility.)
(adedh) (nls,utility.)
(adedrx) (nls,utility.)
(adedry) (nls,utility.)
(adedseq) (nls,utility.)

```

Figure 5. A truncated view of the SYSGD file.

```

30-AUG-73 14:27
NLS>
Jump to Link

(newring) (info,filamp,newring) 2C3
(find room for a new ring element and allocate it. Called with file
number of file where wait the new element. Returns

```

Figure 6. A typical SYSGD citation. The comment was extracted automatically from the cited procedure.

IV Workshop Foundation

18 Software Engineering

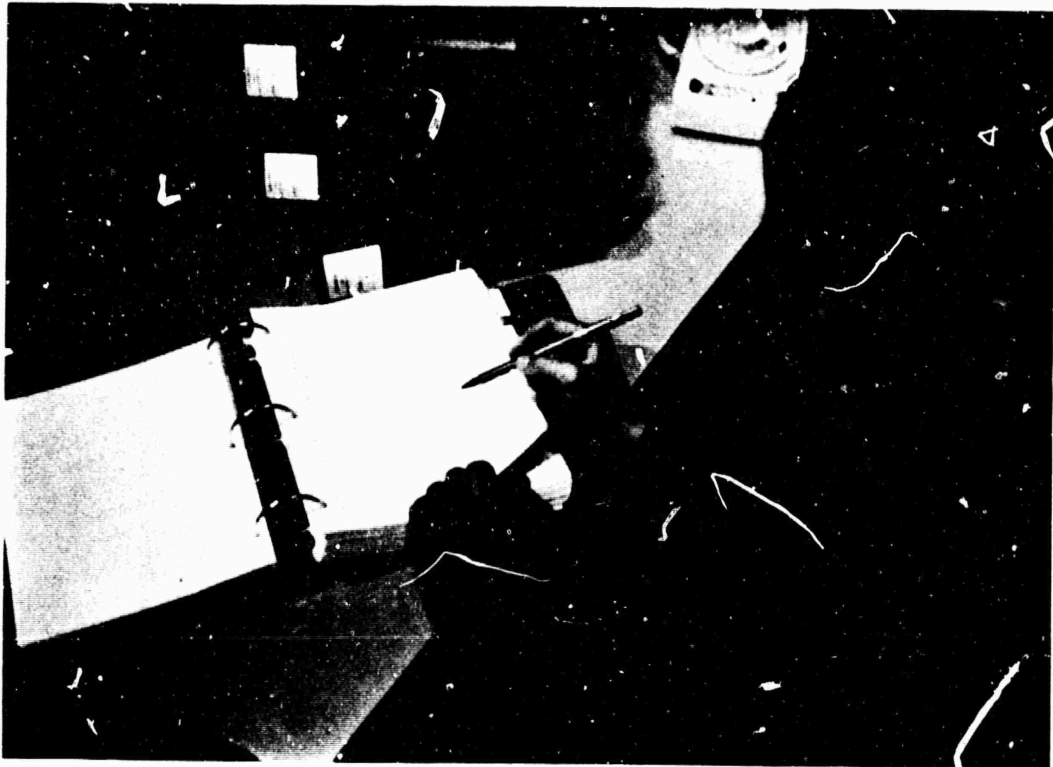


Figure 7. Examining SYSGD offline.

18b1015

Tools for Creating and Testing New System Features

18b1016

A number of tools within the Knowledge Workshop aid in the creation and testing of new NLS features. In addition to the standard NLS editor, several commands and subsystems are specifically aimed at the needs of programmers.

18b1017

Copies of procedures may be made, edited, compiled and instituted into the running system or substituted for existing parts of the running system without disturbing other users. (This may be thought of as being incremental compilation at a procedural level.)

Coordination of modifications to NLS by the NLS programmers is eased through this user programming system. Only after new or changed procedures have been tested are they placed into the actual system code. A tool for recording changed system files is associated with a compiler driver. The names of source code files to be compiled are placed into a branch in this special system file, TASKS. When desired by any of the NLS programmers, the accumulated file list may be driven by another system to compile the files. Records of

IV Workshop Foundation

18 Software Engineering

successful and unsuccessful compilations are kept in this file along with records of system loads. This Utility System is also used for printing out source code listings.

This tool thus permits compilations and loads of a system to be done at more convenient times and provides a record of recently made system changes along with the identifiers of the programmers who made them.

Source Level Debugging

By making minor changes to the TENEX Dynamic Debugging Technique System, DDT, and to the ARC L10 programming language compiler, and by providing a fairly simple debugging submode accessible through NLS, NLS-DDT, ARC software engineers have provided themselves with a primitive but effective source level debugging and (procedural level) incremental compilation system.

Documentation of the commands in the system may be found in [5].

The NLS-DDT system provides an easier way to examine individual cells and L10 data structures, such as records, fields, strings, and call stack frames, than is available in the current TENEX DDT.

Procedures that are compiled in the User Program submode may replace procedures in a running system during a debugging session without the necessity of either patching in machine language code, as in the TENEX DDT, or loading an entirely new system, a slow process for a large, multifile program such as NLS. Symbol definition is resolved with the rest of the running code. Such procedures may also be inserted into the program.

The break-pointing features of TENEX DDT are provided as well as a conditional break-pointing capability.

The command language is less obscure than that of TENEX-DDT and is more consistent with other commands in the NLS environment.

Documentation

Several programmers continually modify the 150,000 computer words of NLS code. In such a large system it is essential that code be clearly documented to permit anyone to fix bugs and make additions to the system as flexibly and easily as possible. A well-documented source code, viewed using the linking and level-clipping features of NLS, provides an immediate overview of the system and an important tool to the augmented software engineer.

Thus, in the development of a software engineering system design discipline, standards and methods for documentation must exist.

Standards for documentation and coding were proposed in [6]. These rules, which describe indentation and commenting suggestions for all common features in the languages used at ARC, have been in effect for some time now. Used in conjunction with the level

IV Workshop Foundation

18 Software Engineering

clipping and line-truncation features of NLS as well as various special user programs for generating system cross-references, they have proven to be valuable for learning and maintaining the system. They also provide handles for the use of automatic documentation extraction programs.

:86101

Illustrations of typical commented L10 procedures are seen in Figures 8 and 9, following.

:86102

IV Workshop Foundation

18 Software Engineering

```

ALL ALL
30-AUG-73 16:21

NLSR      Replace Character

      PROCEDURE 1 Initialize file header and core page status
      table. The header of the file whose number is passed is initialized
      by this routine. All status blocks are set to empty, the marker
      table is set to empty, and the CORPST (core-page-status-table)
      entries pertaining to this file (other than the header) are set to
      indicate that no pages are loaded.
      End 1 NLS file number of file being initialized.

      -----
      LOCAL (index, header, bindex, 'b', 'i', block;
      REF 'b, 'i;
      Initialize file header by updating dummy file header
      and copying that to the real file header:
      nhead = 1;
      nend1 = 1;
      nend2 = 1;
      i file owner (user number) 1
      'SYS g/infi
      funo = n216
      finit = 1; fin = 0; dont = 0;
  
```

Figure 8. The beginning of a typical L10 procedure: SETFIL.

185457

```

BASE
ALL ALL
9/10/73

COMMAND 1 insert 1
      insert
      USER 1
      file = null
      ent = LINK 1;
      * to follow *
      next = USE 1; VISIBLE
      char = USE 1; ent
      ent = text 1
      * to follow *
      next = USE 1; ent
      char = USE 1; ent
      ent = TEXT 1;
      * to follow *
      next = USE 1; CHARACTER
      char = USE 1; ent
      ent = structure
      * to follow *
      next = USE 1; STATEMENT
      ent = LEVADU
  
```

Figure 9. Part of the CML code for the NLS "insert" command from the file <NLS>SYNTAX.

185457

IV Workshop Foundation

18 Software Engineering

Analysis

The designers of a continually evolving system must be able to measure the effectiveness of modifications introduced into the whole system. They must be able to quantitatively and qualitatively measure the effect of a change on the command use of individual users and on the whole system response. Analyses of these measurements indicate the need for modification in training techniques and for further changes.

NLS can measure its own activity in various ways. Each of these measurement techniques was added to NLS at different times and in response to different questions the system programmers were asking about system activity.

For a detailed discussion of the measurement and analysis tools available at ARC, see Section 20, System Measurement Tools.

POTENTIAL TRANSFER OF KNOWLEDGE TO THE GENERAL SOFTWARE DEVELOPMENT COMMUNITY

In a discussion of the need for a National Software Works [7], Balzer, Cheatham, Crocker, and Warshall speculate on the reasons for the gap between the research and production communities:

"First, organizations tend to optimize over the short run, and hence, innovative tool development has been sacrificed to the need to build systems as quickly as possible.

"Second, users have been limited to using only those tools that were both available on the machine they were using and for which adequate information existed."

The tools and techniques that we have developed, and that we use in our own software work, are by no means so special purpose that they are inapplicable to large-scale software producers. ARC has been involved in the general problem of technology transfer with the entire collection of tools in its Knowledge Workshop (5) [8], and the collaborative development and transfer of its software engineering tools to a community distributed over the ARPA Network has already begun to a limited extent.

REFERENCES

- [1] (18a2) Jack Goldberg (Editor). Proceedings of the Tri-Service Symposium on the High Cost of Software. Held in Monterey, California, 17 to 19 September, 1973. Stanford Research Institute, Menlo Park, California 94025. (XDOC -- 24502,)
- [2] (18a1a) Barry W. Boehm. The High Cost of Software. Keynote Address in Proceedings of the Tri-Service Symposium on the High Cost of Software. Jack Goldberg (Editor). Held in Monterey, California, 17 to 19 September, 1973. Stanford Research Institute, Menlo Park, California, 94025. (XDOC -- 24502,)

IV Workshop Foundation

18 Software Engineering

- [3] (18b3a2c) Diane S. Kaye. Experimental NLS Bug-reporting Mechanism. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 27-JUN-72. (10869,) : 834
- [4] (18b3a2c) Douglas C. Engelbart, William K. English, J. F. Rulifson. Development of a Multidisplay, Time-Shared Computer Facility and Computer-Augmented Management-System Research. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. APR-68. (9697,) : 835
- [5] (18b3d1a) Charles F. Dornbush. NDDT Symbolic Debugger User's Guide. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 7-JAN-73. (13716,) : 836
- [6] (18b4b1) Harvey G. Lehtman, Kenneth E. (Ken) Victor. Proposed NLS Code Format and Documentation Standards. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 13-APR-73. (15934,) : 837
- [7] (18c1) Robert Balzer (USC-ISI), T. E. Cheatham (HARV-10), Stephen Crocker (ARPA-IPT), Stephen Warshall (MCA). Design of a National Software Works. USC/Information Sciences Institute, Marina del Rey, California. ISI-RR-73-16. NOV-73. (19208,) : 838
- [8] (18c2) Douglas C. Engelbart, Richard W. Watson, James C. Norton. The Augmented Knowledge Workshop. In AFIPS Proceedings, Vol 42, 1973 National Computer Conference, pp. 9-21, 1973. (14724,) : 839
- [9] (18b3a3b1) SRI-ARC. L10 Users' Guide. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 6-NOV-74. (24426,) : 840
- [10] (18b3a3b1) William H. Paxton. L10 Documentation. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 5-DEC-70. (7052,) : 841

TENEX Development

(by William R Ferguson Donald C Wallace, and Kenneth E Victor)

INTRODUCTION

The work that has gone on at ARC on NLS and other subsystems is all supported on a PDP-10, running the TENEX operating system. This operating system was originally written at BBN, and we receive updates of their version as they make changes. Twice during the past funding period we have received a major update, called a new release. Some of our TENEX work has been to incorporate these new releases into our monitor. In addition, we have added a number of important features to TENEX, both to increase the efficiency for our particular environment, and to add needed features that allow our major subsystem (NLS) work to continue.

One of ARC's most important contributions to TENEX was the development of BSYS (Backup System). Largely the creation of Don Wallace, BSYS was turned over to BBN, who released it in a set of separate procedures as a file system Utility for all standard TENEX sites.

BSYS was designed to provide better dump and archive procedures, and in general give TENEX a better and more sophisticated tape database backup system.

Some of its most helpful features include a global verify of the system; a reasonably global trim function after a system dump; a mapping function which considerably aids problem spotting and fixing, both in hardware and software; and an octal dump, which further aids operators in discovering and clearing up trouble spots.

The TENEX ARCHIVE function as it exists today came out of the work done in developing BSYS.

Further, we have brought up two new EXECs during this period. The EXEC is the user interface program to TENEX. When the user first types a control-C to gain the attention of the system, she is in fact talking to the EXEC. In our environment, we have adopted a philosophy about the EXEC similar to that of TENEX, i.e. we incorporate new BBN releases as they become available, and we maintain an ongoing development of features that are necessary or useful to our particular group.

The final major area of work in the TENEX arena has been the development of a Group Allocation system (16). We (like many other TENEXs) found that we were allowing so many users on the system that the computer became overloaded, with a resultant decrease in service to everyone. To solve this problem, we developed a system that allowed a rational algorithm for users accessing the system. Only a certain number of users are allowed to be logged in at any given time, and this number is further divided into groups, with the membership of each group decided by management. Thus this system allows management to accurately determine who has access to the machine at any given time. We have found this very helpful in bringing the total

IV Workshop Foundation

19 Tenex Development

load on the system down to a level at which each user receives sufficient response speed to have a productive working environment.

TENEX MONITOR MAINTENANCE AND DEVELOPMENT

Process of Bringing Up TENEX 1.29

One of the major jobs in TENEX maintenance is incorporation of our modifications to the monitor into a new BBN release. One of the first activities during the past contract period was to get TENEX version 1.29 running.

During the period we were running the previous monitor, version 1.28, we found it necessary to build many new features into TENEX. When 1.29 was released we found ourselves faced with the formidable task of isolating all our changes and including them in a new monitor. The lessons we learned during the process were very important ones that have changed the style of all our subsequent TENEX work.

A bit of background is essential to understand the problems that faced us in 1.29. The TENEX monitor is divided into about 45 different packages. For instance, the routines for scheduling are in SCHED.MAC, many of the JSYSes (calls on the TENEX operating system) in JSYS.MAC, and the teletype routines in TTYSRV.MAC. This makes modifications of TENEX relatively simple, as most of the packages are somewhat autonomous, accomplishing one logical function. Most changes can be accomplished by modifying one or two of these modules.

What we had done in TENEX 1.28 was to simply put all our changes directly into each module. At the time, this seemed the most logical way to do things. But when 1.29 came along we found it very difficult to isolate the changes we had put into 1.28 versus the changes that BBN had put into 1.28 in making 1.29.

Because of the problems we saw in the way we had done 1.28, we decided to include our modifications to 1.29 in a different manner. First, we decided that we would get BBN to include as many of our changes as possible in their version of TENEX. Then, the next release of TENEX (version 1.30) would already have accomplished much of the work that we would need to do. Second, we decided to change the organization of our changes into a more modular design. Specifically, we defined several new modules for the TENEX monitor. They were SRICOD, MONSRI, and TTYARC, containing respectively a) all the JSYSes written and supported at ARC, b) all our modifications to the core page management routine (PAGEM) and the scheduler (SCHED), and c) all our changes to the teletype routines (TTYSRV). Thus when we received a new monitor, there would be the easier process of including our modules, rather than making major edits to all the BBN modules.

To accomplish the first part of our objective (getting some of our modifications into BBN TENEX), Ken Victor and Don Wallace began a dialog with the BBN personnel. First a list of the modifications to be included was agreed upon. Second, a detailed description of the implementation of these changes was discussed. And finally, Messrs. Victor and Wallace

IV Workshop Foundation

19 Tenex Development

traveled to BBN in Boston to work on the actual implementation. All parties involved felt that this three-step process was quite useful and efficient. BBN received some features that they also needed without the necessity of much programming time, and we were saved the problem of always having to incorporate these changes when a new BBN release was made.

After final discussion and implementation, BBN had accepted the following additions from us:

- 1) Not allowing users to log into the system if the file system is in a disorganized state. Only system programmers can log in to fix the problem.
- 2) Several JSYSes that allow setting and reading some user-specific information cells.
- 3) Routines that determine whether a user has write access to a file before he actually begins editing that file, and thereby saves the user futile efforts to edit that file.
- 4) And, a group of routines that allow setting, testing, and resetting a system-wide global flag.

The second part of the process was also fairly involved. It was necessary to go through all the code for 1.28 and attempt to isolate all our changes. These changes were then consolidated into the three modules mentioned above, MONSRI, SRICOD, and TTYARC. However, even with this goal of separation and consolidation in mind, we were still forced to make a number of edits to some of the other modules. But the extent of the changes within the BBN modules was greatly decreased.

Though the above process was time-consuming, as it essentially required a line-by-line analysis of the monitor (approximately 50,000 lines of code), it was valuable when later versions of the monitor were released. We also decided that in the future we would attempt to do all our modifications to TENEX in one of the above manners, either getting BBN to support them, or including them in our separate packages.

Important Changes in TENEX 1.29 Worthy of Further Discussion

After the initial process of incorporating all our 1.28 changes into 1.29 and debugging 1.29, we embarked upon two major modifications to TENEX. One of these was a major rewrite of the BBN scheduler to make it faster and more efficient in our environment. The second was a rewrite of the BBN disk driver to allow us to use two disk controllers.

The scheduler changes were very important to our goal of supplying reasonably fast computer service to a growing number of users. We found that on our system we were spending approximately 25 to 30% of the total CPU time in scheduler operations. This was because the BBN scheduler had been designed for a much different subsystem-user environment than the one at ARC. In general the BBN scheduler had been written as an all-purpose scheduler. It was directed at handling a mix of subsystems that range from large to small working sets (a working set is the number of pages a user needs in core to run his program), and vary from very interactive to basically batch type programs. This was not acceptable at our site.

IV Workshop Foundation

19 Tenex Development

At ARC, we run only one subsystem extensively, and that is NLS. NLS has a very large working set, and is highly interactive. Due to these characteristics, the BBN scheduler was simply very inefficient for the type of usage at our site. To alleviate this problem, Don Andrews began looking closely at the way the BBN scheduler worked.

Don made several fundamental changes in the design of the scheduler. The two areas in which changes were made were the wait-list manager and the balance-set manager. The wait-list is a list of processes that are currently inactive as they are dependent upon the completion of something else. Reasons for a process being on the wait-list are such things as awaiting TTY input, awaiting the passage of some time interval, and awaiting a state change in some other process. It was found that the wait-list was being polled too frequently, with the result that many processes were being examined, only to find that the conditions for running had still not been satisfied. Thus the scheduler was spending a great deal of time simply deciding that an inactive process was still inactive.

The other area of inefficiency was in the balance-set manager. When conditions had changed such that a process was runnable, the scheduler used a very elaborate algorithm to decide the priority of that process relative to all other runnable processes. This algorithm was taking almost as much time to determine which process to run as was available to run the process.

The fixes for these problems were to 1) reduce the frequency of scanning the wait list, and 2) reduce the complexity of the balance-set manager. The combination of these changes resulted in reducing our system's scheduler overhead from 25 to 30% down to a more livable 12 to 15%.

(NOTE: Our particular scheduler changes were not directly incorporated into the BBN monitor. However, Don Allen has independently made many of the same changes to the BBN scheduler. This scheduler, which will be released in version 1.32, includes these two major changes, and has a similar increase in efficiency.)

The other change included in the scheduler was the addition of a large number of meters, which could be queried to ascertain system characteristics. These meters are sampled and analyzed by the subsystem SUPERWATCH. This subsystem is discussed in the Section 20. The inclusion of these meters and use of SUPERWATCH have allowed us to accurately monitor our system's performance, and make necessary adjustments to improve efficiency.

The other area in which we made a major modification to version 1.29 was in the disk pack driver. The BBN module has capability for handling one controller, which in turn can service up to eight disk drives. To increase the speed of disk operations on our system, we chose to add a second controller, with a resultant configuration of a total of six disk drives, three on one controller and three on the second.

After hardware installation, it was necessary for us to rewrite the driver so that it could deal with more than one controller. We did this in as general a manner as possible, so that if we later expanded to three or more controllers (up to a limit of eight) all we need to do is change one parameter.

IV Workshop Foundation

19 Tenex Development

We have found that the addition of the second controller and corollary monitor changes have greatly increased the speed of disk access. Using only one controller, it took an average of 120 milliseconds to get one page off the disk. With the two controller system, this time has been reduced to 40 milliseconds. The increase in speed is due to the fact that one controller can be preparing for a disk operation while the other controller is performing an operation. The speed-up has been very noticeable during periods when our drum was down, and we were forced to swap off the disks. Using two controllers has made such a disk-only system change from virtually useless to an adequate back-up configuration.

Process of Bringing Up TENEX 1.31

After completing the scheduler and disk driver modifications in 1.29, the TENEX front was fairly quiet, until TENEX 1.31 was released. The reason that we will not discuss version 1.30 is that it was never released: the distribution of that version was delayed until BBN decided to simply ignore distribution of that version, and await completion of other projects. When the monitor was finally released, it was named TENEX 1.31.

Bringing up version 1.31 was less involved than for 1.29. Due to the work that had been put into organizing our changes, putting our modifications into 1.31 was much less laborious. The primary focus of work in getting 1.31 running was the following. Previously, TENEX had been written in a mixture of languages. Some modules were written in MACRO and others were written in FAIL. With the advent of 1.31, all modules were in MACRO. This was nicer in that it simplified assembling a new monitor.

However, this necessitated rewriting all of our changes that had been in FAIL. This was not too difficult, as the format of FAIL and MACRO are relatively similar. So, bringing up 1.31 was fairly straightforward.

Since running 1.31, we have continued our policy of attempting to get BBN to maintain our modifications, or find another way to accomplish the end result through changes to the EXEC or user programs.

The major emphasis in our discussion with BBN has been to get them to incorporate our Big Character Input routines. We have modified our TTYSRV (in our module TTYARC) to handle a special kind of input. When working in DNLS, some of the characters that the user types are sent to the computer with coordinate data. Thus, if the user types a control-D (Command Accept for NLS) not only does the computer receive a control-D, but also the X and Y coordinates of a pointer on the screen at the instant the Control-D was typed. This is used in our environment for the two-dimensional editing which DNLS allows. At the time of this report, we are engaged in specifying such an input scheme for the BBN monitor. We currently expect that this feature will be included in version 1.33, which will be released in January, 1975.

IV Workshop Foundation

19 Tenex Development

Summary of the ARC Modifications to TENEX

As the reader may have already inferred, over the years we have made many changes to TENEX, and the major work required at release time is to incorporate these changes into the new monitor. We would like to now summarize the differences between our version of TENEX and that run by BBN. The changes that we specifically made during this contract period have been explained in depth, and this summary will be a list of them in addition to all the changes made prior to this contract.

We currently run the following changes (listed in order of the module where they reside):

MAJOR

DSKPAK.MAC:

- 1) This is the change discussed above, which allowed us to run two disk controllers for an increase in speed.
- 2) We also modified the manner in which pages are assigned on the disk packs. In the BBN monitor, their assignment sequence requires that all of the swapping is done on one pack exclusively. This is very slow. We changed the algorithm such that swapping is spread over all the packs. This has helped to make the three-fold increase in swapping speed that our new disk pack driver delivers.

DISPLAY.NLS (Local Module): We have written about 20 JSYSES that do the manipulation of our DNLS display data structures. These JSYSES are used in conjunction with our 32K of external Ampex memory. When we move to a new display system, utilizing line-processors, these JSYSES will be eliminated. They have already been removed from the Office-1 monitor.

LINEPR.MAC: Rather than running a DEC printer, we purchased a Data Products Printer, as the latter gives us better a type font. To run this printer, we had to write a new driver, with different character codes.

SCHED.MAC:

- 1) As discussed, we changed the wait-list manager and balance-set manager to get an increase in speed and efficiency.
- 2) We also added meters, which are used in conjunction with SUPERWATCH, to accurately measure the functioning of the system.

SRICOD.ARC (Local module): These JSYSES include a number of features, such as the following which we use in our environment.

- 1) Capability of having the monitor write out all error messages in a file, in addition to the BBN standard of printing them on the logging teletype.

IV Workshop Foundation

19 Tenex Development

- 2) Ability for a user to read monitor cells without the necessity of entering monitor DDT.
- 3) Routines essential to allow a user to change his login password, without having to request this of systems personnel.
- 4) Routines to initialize and manipulate our 32K of external Ampex memory (used for NLS display areas).
- 5) Routines to start and read our Real-Time Clock: We use this to set system time, rather than requiring the operator to type-in the time at system start-up.

TTYSRV.MAC: This is the input capability discussed above which we use to input coordinate data along with a typed character. As mentioned, we are in negotiation with BBN to include this in standard TENEX.

MINOR

PAGEM.MAC: In order to handle our 32K of external Ampex memory, we had to slightly modify this module so that TENEX would not try to swap out the display databases.

SWPMON.MAC:

- 1) We have purchased a Real-Time Clock, and there are routines in this module to start and read it. We set the system time from this clock, rather than by operator type-in.
- 2) In order to properly program our local displays, we have changed the terminal initialization procedure. Depending upon line number, we either set up standard BBN terminal defaults, or special defaults for our displays.
- 3) We also set up different job initialization parameters, which include user ids.

EXEC MAINTENANCE AND DEVELOPMENT

Bringing up EXEC 1.50

Very much like TENEX, we have undertaken the following procedure with regard to the EXEC. About once a year, BBN will release a new EXEC, with all the changes and modifications that have been made in the previous year. Our job at this time is to incorporate all our EXEC changes into the new version. After this major task has been accomplished, we undertake an ongoing program of adding needed features to the EXEC. The following is the story of this process during the contract period.

EXEC 1.50 was released early in the contract period. Ken Victor was responsible for getting this version of the EXEC running. He had little trouble in bringing it up, as we had few changes in the EXEC at that time. However, during the year in which we were running on 1.50 he added many features that will be detailed below.

IV Workshop Foundation

19 Tenex Development

We discovered that our changes and modifications in the EXEC posed problems similar to those we had met in TENEX development. These were that, though making changes to a running version was relatively easy, the process of incorporating them into a new BBN version was exceedingly laborious. So the release of EXEC 1.51 posed a major obstacle.

Bringing up EXEC 1.51

When EXEC 1.51 was released, we found ourselves faced with the following dilemma. The new EXEC had some increased speed, efficiency, and new features that we wanted, but our 1.50 changes were scattered throughout the code of 1.50. (Like TENEX, the EXEC is broken into several modules; however, since the EXEC is smaller, there are only ten, rather than 45 modules.) We decided to localize our changes into a specific area.

We decided that the most appropriate way to include our changes was to modify the BBN EXEC in the following way. First, we tried to put all our changes into one package, called SRIXEC. Any extra routines that the user would call were included in this package. Second, we wrote a new command dispatch table, which included the extra commands that we had added to the EXEC. Third, and most important, we placed all our changes under assembly switch control. For those of you unfamiliar with this term, this means that by defining a flag, such as SRIARC=1, we tell the compiler that we wish to include our code. If this flag is set to zero, our code is not assembled in the EXEC. Thus we have total control over which of our modifications are activated, and which are left in a BBN context. The definition of all these switches is found in SRIDEF.

We found this methodology very useful, since over time we have decided to eliminate some of our changes. Before this organization existed, eliminating a modification was almost as difficult as adding it. Now it is simply a matter of setting the assembly switch to off.

Major Modifications to the EXEC

During this contract period we have made two substantial modifications to the EXEC. Otherwise, we have carried forward the changes previously made in version 1.50. The two new features are to define a number of different user types, all with different capabilities and available subsystems, and second to have the EXEC maintain each user's ident (needed for NLS) and his group number (necessary for the group allocation system).

The necessity of different user types was apparent from the very heterogeneous nature of our user community. We found ourselves faced with setting up user parameters for the following types of users. They are:

a) Network users who have access to the ARPANET, but are allowed to use only the NLS subsystem, i.e., they are denied access to such programs as TECO, MACRO, and LOADER.

b) Commercial users are similar to Network users, but are denied access to the ARPANET. Thus they are not allowed to use such programs as TELNET and FTP.

IV Workshop Foundation

19 Tenex Development

c) NIC users are allowed access to a very small subset of commands and subsystems, but are allowed to run only the NIC/QUERY system.

These user types are also allowed a subset of EXEC commands, and in particular are denied access to programming commands, such as START, ENTRY, and DDT, and also denied certain commands that are inappropriate, such as LIST.

We have found this mechanism useful from a managerial point of view, in that users are allowed only the capabilities for which they have contracted. It also increases the efficiency of the system as all the above user types share the common attribute of being limited primarily to NLS. This increases system responsiveness, as all available memory can be used to run NLS, without competition from other large subsystems, such as FORTRAN, MACRO, LISP, and LOADER.

The other substantial change was to have the EXEC maintain information about a user's ident and group number. As will be discussed below, we have installed a group allocation system, which regulates login access to our system. Since the system works on the basis of N members of group X allowed on at any given time, it was necessary to include this information in a database available to the EXEC. At the same time, we also added a feature so that the EXEC would have that user's ident available, as a convenience to the user.

The actual philosophy and mechanics of the allocation system will be discussed below in the section on Analysis. However, what the EXEC does is as follows: At login time, the EXEC queries the database of group number and ident maintained for each user by the operators. On the basis of a user's group number, and the number of other users in his group currently logged in, the EXEC determines whether or not this user can have access to the system. If the system is already overloaded, the EXEC denies the user access, and he must try again later.

However, if the EXEC determines that this user can log in, the EXEC then gets that user's ident from the same database. The EXEC saves the ident in a monitor table, and NLS can later query that table to find the user's ident. This saves the user the inconvenience of having to type his ident every time he enters NLS.

In the event that more than one person logs into the same directory, a list of idents is stored. The EXEC asks the user to provide his particular ident, and then checks this ident against the total list for that directory. If the ident is valid, the EXEC saves it away. And if the ident is not permissible for that directory, then the EXEC asks again.

Summary of ARC Modifications to the EXEC

As mentioned, many of the EXEC changes have been written in the past, and are carried forward with each new version. The following is a list of all the changes we currently maintain:

A. Changes implemented by BBN in their version of the EXEC:

IV Workshop Foundation

19 Tenex Development

1) SRI-ARC change in password of login directory command. This allows a user to change directly his own password, without the need for intervention by the operators. 190101

2) SRI-ARC DOWNTIME command: This copies the file <SYSTEM> SCHEDULED-DOWNTIME to the user's terminal, and informs the user of the up-down schedule of the system. 190101

3) SRI-ARC DISCUSE command, which tells the total number of disk pages that are free or in use. 190101

B. Changes we maintain at ARC to improve our working environment: 190101

1) SRI-ARC network user definitions, which include 190101

a) restriction of EXEC commands to network users (unless they are NETWHEELS).

b) default to <NETSYS> (rather than <SUBSYS>) for programs.

2) SRI-ARC commercial user definitions, these 190101

a) restrict EXEC commands to commercial users.

b) default to <COMSYS> (rather than <SUBSYS>) for programs.

4) SRI-ARC group allocation scheme, includes (mentioned above): 190101

a) Login restrictions

b) GROUPSTAT and ELOG commands

5) SRI-ARC autologout of inactive jobs stuff, which includes the REFUSE AUTOLOGOUT and RECEIVE AUTOLOGOUT commands (requires SRI-ARC JSYS ATLJB) 190101

6) SRI-ARC terminal types IMLACs and LINEPROCESSOR: 190101

a) IMLACs are terminal types 5 (no long vectors) & 6 (long vectors)

b) LINEPROCESSORS are terminal type 13

c) This also assumes TENEX has the needed terminal types

7) SRI-ARC FILSTAT command, which always prints connected directory. 190101

8) SRI-ARC COMMAND ACCEPT feature, which treats COMMAND ACCEPT (Control-D) as equivalent to EOL in EXEC. 190101

IV Workshop Foundation

19 Tenex Development

9) SRI-ARC ident stuff, includes:

- a) Setting ident at login time
- b) (requires SRI-ARC JSYS SJBST and RJBST)

10) Allow user to log in without typing LOG

11) Tell user about new Journal mail at login time

12) SRI-ARC MESSAGE command, that is shorthand for using READMAIL subsystem

13) SRI-ARC BYE command, that is equivalent to the BREAK (LINKS) command

!4) SRI-ARC SYSTAT command, which doesn't type the entire SYSTAT if the load is too high, unless the user is a network user, a commercial user, or a user who has certain special privileges necessary to system maintenance.

System Measurement Tools (by Donald I Andrews)

INTRODUCTION

We have developed two kinds of system measurement tools at ARC during the report period.

The first tool is a system of collecting and displaying information about our PDP-10 TENEX system--both about how it is performing and how it is being used. This system is called Superwatch.

The other tool is a general tracing facility for NLS debugging and analysis work. It consists of a Trace subsystem and a Cross-Reference subsystem.

These tools are independent, but can be used together to get information about NLS performance on our TENEX.

SUPERWATCH MEASUREMENT SYSTEM

Motivations for Superwatch.

The Superwatch system grew out of a need to identify problems within our TENEX. At the time we did not know whether these problems were software performance problems, hardware failures, or configuration problems.

BBN provided a subsystem called "watch", that printed very general system information and CPU usage by user. Also, an EXEC command could be used to get some statistics about system performance and subsystem usage. We found these two facilities quite inadequate: they did not provide us with the information we needed.

Also, considering our overall plans, we wanted to obtain a record of system performance that we could refer to later. This would allow us to compare system configurations, various versions of NLS, etc.

Design

As a first step in implementing Superwatch, we studied the structure of the TENEX scheduler and memory manager, and inserted metering codes in strategic places.

This usually took the form of incrementing a counter for every event we wished to meter. In many cases it consisted of timing sections of code and adding the time used to a variable, and incrementing a counter. The average time could be computed from these two variables. By recording these variables at intervals, the average time during a interval could be computed.

IV Workshop Foundation

20 System Measurement Tools

The metering code lies entirely within the TENEX monitor.

200000

The meters must be sampled at intervals to get meaningful statistics. This sampling is done by a user program executing a special JSYS (system call).

200000

The user program that does the sampling is a portion of Superwatch. The meters are collected and written on a file in a raw form.

200000

The collection process must not impose much load on the system, otherwise it would not be possible to get a realistic measurement of the system as it runs.

200000

The load imposed by Superwatch while collecting is a function of the collection interval. In the 15-minute interval we use to obtain our daily records, Superwatch uses 0.1 % of the CPU.

200000

Superwatch also reads the raw sampling information and prints it in a form suitable for studying.

200000

Since there are many meters, and many statistics that can be computed from them, the user may specify just what he wants to have.

200000

The user may have the statistics printed either in tabular form or in graphic (histogram) form.

200000

Examples of Superwatch graphs are shown on the following pages.

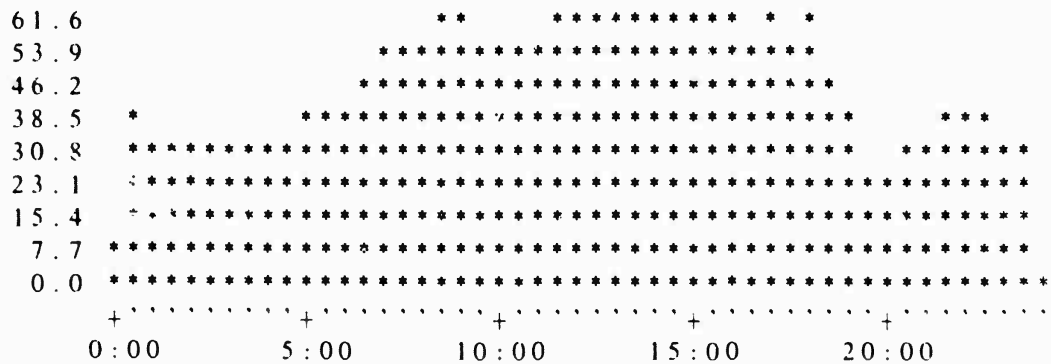
200000

IV Workshop Foundation

20 System Measurement Tools

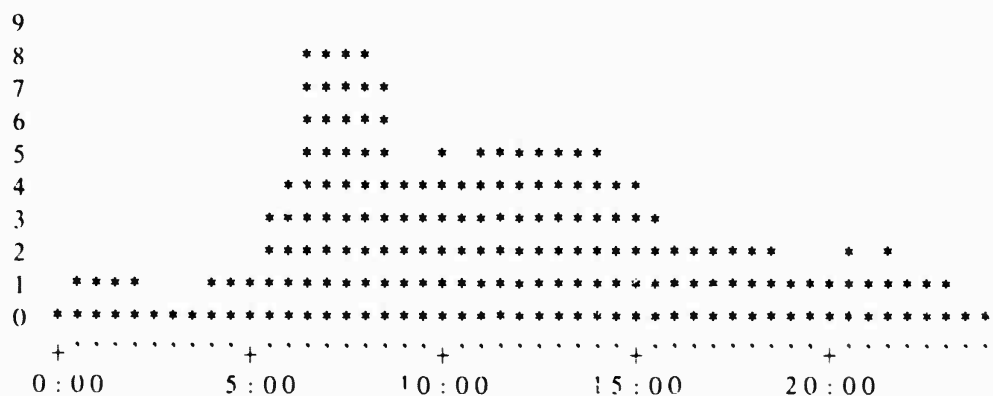
TIME PLOT OF AVERAGE PER CENT OF CPU TIME CHARGED TO USER ACCOUNTS FOR WEEK OF 1/20/74

x axis labeled in units of hr:min, xunit = 30 minutes



TIME PLOT OF AVERAGE NUMBER OF NETWORK USERS FOR WEEK OF 1/20/74

x axis labeled in units of hr:min, xunit = 30 minutes



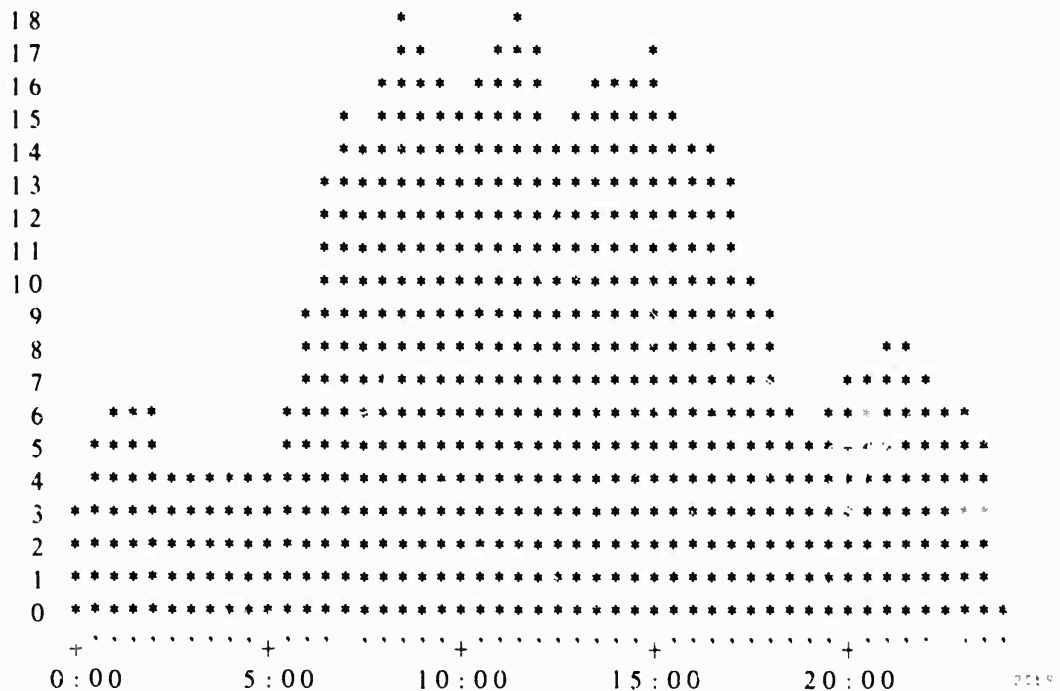
IV Workshop Foundation

20 System Measurement Tools

TIME PLOT OF AVERAGE NUMBER OF USERS FOR WEEK OF 1/20/74

X axis labeled in units of hr:min, xunit = 30 minutes

22118



Capabilities

22119

The statistics gathering and printing features of Superwatch have several useful aspects: 22119a

The files generated by the sampling process are relatively compact and it is reasonable to keep them on tape. Individual statistics can be printed by returning the file in question and using the printing features of Superwatch. This avoids keeping the information on paper. 22119b

We have arranged our TENEX startup procedure so that Superwatch is started whenever TENEX is started. The collection interval is set to 15 minutes, and it runs until TENEX crashes or is halted. Thus we have a record of TENEX performance (and by the absence of information, a record of downtime). 22119c

When a user wishes to have some statistics printed, he specifies: 22119d

The day in question.

The time interval, including a whole day.

The set of statistics he wishes to see.

IV Workshop Foundation

20 System Measurement Tools

Whether he wants averages, a tabular printout of values for each interval, or both.

The user can also specify that he wants to see a graph of a particular statistic over the period in question. He then has the choice of a plot of the statistic versus time of day, or a distribution of the statistic for that period.

Another mode of Superwatch printout allows users at display terminals to see a real-time histogram plot of a set of statistics.

In addition, three unusual features were implemented in the Superwatch system:

A Program Counter (PC) sampling feature is used to measure program execution times. The user specifies a lower bound and a range size for the region of his program he wishes to measure.

When the PC sampling feature is enabled, the TENEX monitor checks the user (or system) Program Counter at clock interrupt time. This represents a random sample of the program execution location. The counter in the monitor that represents the appropriate range is incremented. This results in a distribution that approximates the distribution of program execution time.

In a similar fashion, the location of instructions causing page fault traps, or the address of the fault reference, can be recorded in a table.

Both the PC and Fault samples are printed in tabular form, with the share of events in each range indicated.

The PC sampler has been useful in identifying both TENEX and NLS bottlenecks--in some cases down to a few instructions. It is also very valuable to know that a portion of NLS, for example, is not critical at all and may be coded in a wasteful but useful way.

We used the PC sampler extensively to increase NLS efficiency by carefully coding low-level character manipulation routines and the call/return routines.

Our Use of Superwatch

We maintain a daily record of system performance for both our ARC system and the OFFICE-1 system. Analysis personnel obtain a brief summary and check performance periodically.

Graphs of important statistics are printed, including the use of NLS (display NLS, teletype NLS, broken down by local or network user)

In the event of unusual or poor performance, systems personnel look at the statistics and identify the problem, if necessary.

We also use the statistics to determine our need for more computer resources.

IV Workshop Foundation

20 System Measurement Tools

NLS MICROANALYSIS TOOLS

Motivations for NLS Analysis Techniques

NLS is a very large system which is written by several programmers and is changing rapidly. As development goes on, we wish to increase its efficiency.

The system is complex, and exhaustive checkout before releasing new systems is not feasible.

In this environment, sophisticated system debugging aids are very desirable. We have only scratched the surface, but have found the following very useful.

NLS Flow Tracing Facility

The flow tracing facility runs as an NLS user program (does not involve special NLS source code) and does two things to NLS:

It replaces the procedure call and return routines so that calls and returns can be recorded.

It enables two interrupt characters so that the tracing can be turned on and off at any time.

When enabled, the flow-tracing code records the following things on a file in condensed form:

Calls

Returns

New page references

Execution time at each of above events

Another program reads the condensed file and prints it in a suitable text form.

It is printed in a hierarchical form with each call causing subordination and each return ending the subordination.

The user has control of how much information he sees. The depth into the structure is controlled, as well as the total output. Also, he can elect to see only one branch in a large file.

We have found the tracing facility very useful for:

Finding bugs

Identifying time sinks

Making code more localized so that fewer pages are referenced to perform a given task

IV Workshop Foundation

20 System Measurement Tools

Understanding the operation of a given command in NLS.

NLS Cross-Reference Facility

Sysguide Creation

As a by-product of the cross-reference facility, a file called "Sysguide" can be created.

The Sysguide file has a statement for each procedure and each data record definition. The name of the statement is the same as the name of the symbol. The statement contains a file link to the source procedure or record definition. In addition, some documentation about the purpose of that procedure is included.

The Sysguide is used to locate procedure source code. This is necessary since there are some 100 source files of NLS source code.

In the current version of NLS, a JUMP TO NAME EXTERNAL command can be used to jump through the Sysguide file to a given procedure or record definition.

Cross-Reference Creation

Basically, the Cross-Reference system is used to obtain a data structure describing the procedure and data references in NLS. It is used to find out who calls whom, who references what data, etc.

In creating a cross-reference, the user specifies memory bounds for references and bounds for referenced locations. Also, an opcode filter is specified to indicate the type of references that are of interest.

The Cross-Reference system then reads the actual object code for NLS (or any subsystem) and creates a data structure representing the reference network.

Printing from Cross-Reference Data Base

The user prints part or all of the cross-reference database by using one of several commands in the Cross-Reference system.

The entire cross-reference may be printed, but it is usually very large.

The user may specify a symbol and ask to see all references to it, or (if it is a procedure name) all references from it, or both.

In addition, the user may ask to see references to and/or from each reference, and so on, to a specified depth.

Alternatively, the user may ask if a reference (to/from) one procedure will result in a reference (to/from) another given procedure.

IV Workshop Foundation

20 System Measurement Tools

Our Use of the Cross-Reference Facility

The Sysguide file is the most frequently used part of the Cross-Reference Facility.

The Cross-Reference system has been used on a limited basis and proven to be useful. However, it is not used as much as it might be, probably for these reasons:

- 1) It is very easy to read and do searches in NLS source code using the NLS system itself. This reduces the need for a cross-reference system.
- 2) It takes about five minutes of CPU time to create the cross-reference database. The NLS system is frequently changing and a cross-reference database is not created for each new NLS system. Hence there is rarely an up-to-date cross-reference database.

Summary

We have learned a great deal and benefited in several ways by developing these analysis tools:

These tools have been very helpful in understanding TENEX operation, configuring our system, developing and tuning NLS.

We have learned what analysis tools are the most necessary: the real-time measurement of operating system performance, PC sampling, and program tracing.

We know to a large degree just how we want those tools to operate.

While this has not been an exhaustive attempt at tool building, we have found it very worthwhile.

Appendix HIGHLIGHTS OF THE PREVIOUS REPORT

TEAM AUGMENTATION

In the period June 1970 through June 1972, our work toward Team Augmentation fell into five areas: improvement of our dialog support system, the initial work on our handbook, our baseline record system, development of basic NLS, and reorganization of our laboratory staff.

Dialog Support System

As with the XDS-940 Journal system, the PDP-10 Journal system developed in that period served as an initial open-ended information storage and retrieval system, oriented toward recording the thoughts, notes, designs, work pieces, and reports published by users.

ARC and Network personnel used the Journal system daily.

From April 1971 to June 1972, approximately 1600 documents were generated at ARC and submitted to the Journal.

The PDP-10 Journal system provided for automated entry of online documents in contrast to the essentially manual technique used on the XDS-940.

Delivery of Journal submissions to authors and recipients was automated on the PDP-10 System.

Online Journal documents may now be reached through NLS by simply using the catalog number as a file name.

The improved access to Journal documents resulted in increased linking between Journal documents, whereby dialogs began to involve a number of documents, all interlinked.

Handbook

We began development of a "Handbook," a "super-document" that contained the beginnings of an up-to-date, large, detailed, highly cross-referenced and well-indexed description of ARC project-team activity.

Such a document would have provided ARC, as a team tackling complex system-development projects, with the highest possible visibility over its working environment.

Toward the end of the contract period we set up a team to design a Handbook system which we intended to construct, index, and maintain this document.

Appendix

21 Highlights of the Previous Report

Baseline Record System

We constantly face more opportunities for changes or additions to our evolving system than we have resources to carry out. Therefore we have attempted to use NLS to find ways to make more effective, coordinated analysis of our ideas, and of our people, system, and material resources.

The result of such coordinated analysis was the adoption of a current visible plan, or "baseline" of expected events, agreed upon system developments, their external configurations, and resource allocations.

The information relative to the planned system developments was contained in our Baseline Record.

The Baseline Record was a special subcollection of the Journal. It consisted of a series of files specially formatted to contain task and resource allocation information, including particularly files of plans, specifications, analyses, designs, etc.

The 1972 Baseline Record system was concentrated on the recording of information relevant to individual tasks being performed under consideration by various ARC staff members.

There were then over 200 tasks of various magnitudes to consider in our planning and operational environment at any time. These ranged from simple bug-fixing to complex design or implementation tasks that may require the efforts of several people over many months.

We developed a set of programs with an initial data storage system that organized information recorded about these tasks with features that permitted routine summary views to be produced and that also made available flexible, user-created views of the Baseline task information.

Procedures were developed for data collection and input and for view production that aided in weekly updating of the Record. These views were produced in hard copy and were also entered into the Journal.

We were not satisfied with the 1972 Baseline Record System.

We felt that our ARC users were not well guided and trained in Baseline Record System use and the initial system did not produce views that were useful enough, mainly because most of the needed data were not in the system.

Although we used ARC's Baseline Record System on a current task-by-task basis during the past year, we still needed to develop a more complete, "higher level" picture of what new ARC system developments (functions, features, stages...) we wanted and expected to see. Among other considerations, this included better definition of activity goals.

Appendix 21 Highlights of the Previous Report

Basic NLS

In the contract ending 1972, we took several steps to further augment the software engineer. In fact, we coined the acronym SEAS (for Software Engineer Augmentation System) to give specific system orientation towards the end of developing a full and balanced set of tools, techniques, methods, principles, etc. for augmenting software engineers.

The developments described below were part of an accelerating activity--an important part of our near-future plans in the next contract period involve a greater level of activity here.

TNLS and DEX

A new and effective typewriter version (TNLS) has found wide use, both at ARC and at sites on the ARPA Network.

Improvements were made in the display version (DNLS), and a first version of an offline mode (DEX) was introduced.

Changes that made possible cross-file editing allowed any two passages to be involved by a given command.

In TNLS, addresses in a command may be "links" that could call any passage in any file on the system.

In DNLS, split screens allowed the user to view any two passages and control cross-file editing visually.

Viewspees made possible selective assimilation of information from one file into another.

New special purpose subsystems were developed or improved.

These included a sort-merge system, a user program system, and the output processor.

Language development continued.

In 1972, the primary language systems developed and in use at ARC were the Tree-Meta Compiler-compiler System and the L10 Programming language system, which was written in Tree-Meta.

Work took place on a Modular Programming System (MPS) in collaboration with a group at the Xerox Palo Alto Research Center.

Internal Organization

During 1972, several ARC organizational arrangements were introduced, centering, in the early part of the period, mainly on line-activity structure and associated roles.

Appendix

21 Highlights of the Previous Report

The creation of pusher (task leader) roles for tasks and coordination roles for system architecture, methodology, and personnel resources placed the responsibility more directly on selected individuals.

Pusher roles were defined in the framework of the developing Baseline management system. Coordinating roles were also carried out in this environment.

In the fall of 1971, we set up a four-man Executive Management Committee (EMC) to carry out much of the day-to-day operating management.

During the first half of 1972, Dr. Engelbart established a new, broader overall organizational structure.

This structure consisted of three main activities that cover our framework and goal setting, line operation, and personal and organizational development needs.

These activities were called: FRAMAC, LINAC, and PODAC.

FRAMAC was to discuss and define the ARC intellectual framework and set longer range goals and plans.

LINAC was to carry out activities within the framework that move us toward the goals, including more detailed, shorter range planning.

PODAC institutionalizes continuing personal and organizational development.

NETWORK INFORMATION CENTER: OPERATIONS AND DEVELOPMENT

The ARPANET could be viewed as a collection of resources, people, hardware, software, data, and special services that could be brought together for short or long periods to work cooperatively.

Built upon hardware and fundamental software connections are the processes that assist users to find the geographically distributed facilities they needed to solve or study problems and to allow scattered people to work together effectively in tasks of mutual interest.

We see the Network Information Center (NIC) as one part of the ARPANET experiment that was interested in the latter problems.

The NIC helps to create and sustain the sense of community needed in an experiment such as the ARPANET.

The NIC was not a classical information center because it provided a wider range than bibliographic and library services.

Appendix

21 Highlights of the Previous Report

The NIC Public

One of the problems in the design of an information service was to determine the clientele and its needs. Our initial analysis showed us four main needs:

Reference and General Network Information,

Collaboration Support,

Document Handling and Creation, and

Training.

The clientele for NIC appeared initially to be people developing and building the Network, who were to be followed by those whose research or development interests would be intimately connected with Network resources or who would be experimental users of various Network resources.

NIC Services

To meet the above goals, the NIC services available at the end of the report period, May 1972, through the Net were:

Online:

(1) Access to the typewriter version (TNLS) of the Augmentation Research Center Online System (NLS) for communique creation, access, and other experimental uses.

(2) Access to Journal, Number, and Identification Systems, which allowed messages and documents to be transmitted to Network participants

(3) Access to a number of online information bases through a special Locator file using NLS link mechanisms.

Offline:

(1) A Network Information Center Station set up at each site with:

(a) A Station Agent to aid in use of the NIC

(b) A Liaison to provide technical information about his site

(c) A Station Collection containing a subcollection of documents of interest to Network participants.

(2) Techniques for gathering, producing, and maintaining Network Functional Documents, such as:

Appendix

21 Highlights of the Previous Report

- (a) Current Catalog of the NIC Collection
- (b) ARPA Network Resource Notebook
- (c) Directory of Network Participants
- (d) NIC User Guide.
- (3) General Network referral and handling of document requests.
- (4) Building of a collection of documents potentially valuable to the Network Community. (In the beginning we tried to collect documents valuable to network builders.)
- (5) Crude selective distribution to Station Collections.
- (6) Training in use of NIC services and facilities.

NETWORK PARTICIPATION

Our Network participation outside of NIC activity was in two main areas: protocol development through work in several protocol design communities and general Network coordination through membership on the short-lived Network Working Group Steering Committee and its successor, Network Facilitators Group.

COMPUTER FACILITY

Hardware

At the end of 1971, we transferred our computer operations from an XDS-940 to a PDP-10 computer. The transfer effort was described in our interim report for the first year [1].

Hardware activity during 1972 focused on additional tuning of the new configuration, maintenance, troubleshooting and operation of the facility, and some upgrading of critical parts of the system.

Our hardware configuration contained a number of old, one-of-a-kind pieces of equipment brought over to the PDP-10 system from the previous XDS-940 system. These pieces of equipment proved difficult to maintain and studies were launched on how to replace or upgrade this equipment. A new BBN network interface and a new DEC RP-02 disc system were installed in the spring of 1972, replacing older unreliable equipment. Hardware upgrading of our display system and its special core box was begun to provide temporary relief until a replacement system could be planned. An additional 32K of core was to be added shortly. Studies leading to recommendations to add another channel, disc controller, and set of disc drives were completed.

Appendix 21 Highlights of the Previous Report

System Software

TENEX

We cooperated actively with BBN and other users in debugging and maintaining TENEX, and developed a few new features, both visible to users and internal to the system.

Within the system:

We abandoned TENDMP for loading the monitor from DECTAPE and used instead DTBCDT from DEC.

We added a JSYS, a jump to a monitor subroutine, to say that padding (sending rubouts) was required for fast terminals when a CR or LF was output.

We made many changes to the teletype routines to accommodate our displays.

To greatly simplify startup, we changed the starting address of the monitor from 100 (which goes immediately to DDT) to SYSGO1.

We ceased to add code to existing files when we got new monitor releases. Instead, we defined additional files that were assembled with each group of files and, where possible, made our additions in these new files with JRSTs and CALLs to the new code.

We modified the system so that if CHECKDSK does not run successfully, then nothing else, e.g. AUTO-STARTUP jobs, could run (except for the operator's console and one special dial-up line) until the disk was fixed and CHECKDSK was run successfully.

In the user's view:

We set up an advise command so that one terminal may control a job loaded at another terminal.

We added routines that log out a user who does nothing for a certain time, and that refuse entry if the system is overloaded.

REFERENCES

- [1] (21d1a) No Author. Network Information Center and Computer Augmented Team Interaction. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. 11-FEB-72. (8277.)