Reprinted from —
AFIPS — Conference Proceedings
Volume 45

Montvale, N. J. 07645

User interface design issues for a large interactive system

by RICHARD WILLIAM WATSON
Stanford Research Institute
Menlo Park, California

ABSTRACT

User interface design issues are discussed for a large interactive system. The assumptions about the user environment are explicitly described. Issues discussed include command language syntax, command recognition and completion, subsystem organization, user extension capabilities, user options, and various forms of prompting, help and feedback. These issues are discussed within the context of an existing system, the NLS system.

INTRODUCTION

The large interactive system user interface issues discussed in this paper reflect experience at Stanford Research Institute (SRI) over the past twelve years in the evolution of the user interface to the NLS system. NLS is a prototype collection of tools in a growing workshop of tools and services to aid knowledge work.[1,4] NLS provides facilities to support activities such as document creation, study and publication, message handling, information filing and retrieval, and software engineering. We expect the number of tools and the vocabulary that controls the use of these workshops to grow. We further expect that the use of such workshops 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 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.

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 to interactive systems 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 teleprinters 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 (DNLS), are described in References 2 and 3. NLS can also be used from typewriter terminals (TNLS). In this paper, 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 specification. Forms of general NLS information portrayal are described in Reference 1.

HIGH LEVEL ASSUMPTIONS UNDERLYING THE DESIGN OF THE NLS USER INTERFACE

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

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" langauges. Two, when in trouble, he will invoke help or tutorial functions in a standard way.

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, their knowledge, 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.

Ease of communications 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.

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]

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 user 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.

Range of workstations and symbol representations

The range of workstations available to the user will increase in scope and capability. These workstations will support use of text with large, open-ended character sets, pictures, voice, mathematical notation, tables, numbers, and other forms of knowledge. Even portable hand-held consoles will be available. Indeed the multiplicity of possible terminals 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 workstation both in the form of user interface aids and facilities, and in the form of frequently used tools.

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 workstation 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.

Tools embedded in a computer network

The computer-based tools of a knowledge workshop will be provided in the environment of a computer net-

work, 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, such as a compiler, might exist on a remote 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. 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.

The key point to emphasize is that even when hardware costs decrease to the point where a user can perform 90 percent 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

Problem orientation of the command language and tolerance for ambiguity

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.

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 constructs that are similar to his thought processes, natural problem solving vocabulary, and language forms. The machine will then break down the request into smaller steps as required for internal processing.

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.

Many people hope to allow natural language to be used in making statements to the machine. This capability will require models of the user and task domains for understanding.

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.

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

SOME COMMAND LANGUAGE CONSIDERATIONS

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 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 automatically assumed by the system 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 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, prompting 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 paper 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 prompting, 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 or operation, such as DELETE, can naturally be applied to many entities, such as STATEMENT (a 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 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×m commands. For example one can command DELETE STATEMENT, DELETE NUMBER, DELETE FILE etc.

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

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:

(1) A single-character mode allowing high-speed single-character recognition of the most commonly used command keywords; less commonly used command keywords 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 in the first character. This mode is generally preferred by experienced users because of the conciseness 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.

(2) A demand mode requiring a special character to initiate recognition: This has proved to be popular for new users of typewriter terminals, particularly those with experience using the TENEX operating system, under which NLS currently runs.[13]

(3) An anticipatory mode requiring the user to type just enough characters for the command to be uniquely specified; the system then automatically fills in the remainder of the command word.

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

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 to an 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]

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

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 the order of the operands 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.

Related to argument specification is the problem of choosing argument delimiters. There is a need for the following delimiting functions.

(1) Delimiting command words

(2) Delimiting arguments
(3) Delimiting optional arguments or command word fields
(4) Delimiting commands
(5) Selecting arguments from a display screen, and confirming the selections

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. Therefore, except for using a special character to indicate an optional argument, or command word, a single code is used for the other functions 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 default to be inconvenient. One of the buttons on the mouse also serves this function.

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

There are three flavors of command completion.

(1) Command Accept: 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.

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

(3) Insert: 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.

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 workstation types whenever a design decision must be made between language forms possibly favoring one type or the other.

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

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 as switching subsystems switches vocabularies, 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 in such a way 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 has 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 that the user can control as he wishes for the amount of prompting, feedback, recognition mode, 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 Command Meta Language described below, makes possible tailoring the user interface to specific users or groups of users.

All operations that have a natural inverse com-

mand have been given one (although NLS still does not have an "undo" facility).[14] A general undo/redo facility has a number of technical difficulties and its value might be questioned. However, the ability to undo or redo the last one, two, or three commands would clearly be useful.

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 NLS system primitives as needed. They can use the Command Meta Language to specify a user interface if desired.[12] 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]

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.

HELP, STATUS, AND PORTRAYAL FACILITIES

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 problems are dealt with for typewriters. Let us examine some of the types of information that the user needs in order to keep his bearings.

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.

(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 return to previous points or views as well as to move on.

(2) Subsystem or Tool Space—In workshops containing many tools and commands, the user needs to know which tool or tools are active, which ones he was in previously and their order, and which ones he can enter from here.

(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.

The NLS display screen is organized into windows, as described in some detail in Reference 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 of his information space. Items selected from the screen by pointing at them with the mouse are indicated with an appropriate feedback mark. With typewriter terminals, one does not have this two-dimensional random display capability. While the same information can be given to the user, less can be given automatically, or at least the information must be given in an altered form.

(1) Information Space—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 under various headings, his mail, or single documents. Files in turn are hierarchically organized as a tree of information nodes containing text, graphics, or both.

Files can contain cross citations to specific points within other files or the same file, 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 fed back 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 References 1 and 4.

(2) Subsystem or Tool Space—NLS is viewed as a collection of tools (subsystems) that can be used cooperatively. 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, and so on. 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—Several levels of feedback and types of "help" are available to the user in formulating a command to the system. Each is described below. The Help database, to be described, 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 commandword or field, it generates what we call "noise words" set off in parentheses so the user can distinguish between what he has entered and what the system has added. For example, INSERT STATEMENT (to follow), MOVE WORD (from) aid the user to remember to designate a statement in the first case, or select the word to be moved. In the latter case, after selection, the characters (to) will be fed back to prompt for the destination of the move. The noise words aid the user in remembering what to do next. 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. In DNLS the prompts are displayed on the line below that used to feedback the state of the users command specification and appropriately positioned horizontally. In TNLS the prompts appear in the command specification feedback as appropriate. 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 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 up to this point to take the user to an initial point that describes the specific command and field where he is located.[10]

Once in the Help Database, a simple set of command conventions and the organization of the database allow the user to easily examine related subjects or move to higher level descriptions.[10] There are many unanswered questions about the best structure of a help database, how to mesh online and offline documentation properly, and what forms of accessing mechanisms to provide for novices and skilled users. We are just beginning to review our experience with online help facilities to this point.
(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.[11]

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.[14] 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 data bases needed to implement the user interface have been modularized and isolated as a "Frontend" that can run on a separate computer, such as a minicomputer 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, a command language interpreter, and two data bases; 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, and so on. The Grammar is generated from a high-level description of the user interface written in a language special for this purpose we call Command Meta Language.[12]

Given this particular system organization, it is 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.

ACKNOWLEDGMENT

This work sponsored by the Defense Advanced Research Projects Agency and Air Force Systems Command's Rome Air Development Center, Griffis AFB, N.Y.

REFERENCES

1. Engelbart, D. C., and W. K. English, "A Research Center for Augmenting Human Intellect," *AFIPS Conference Proceedings*, 1968 FJCC, Vol. 33, pp. 395-410.
2. English, W. K., D. C. Engelbart and M. A. Berman "Display-selection techniques for text manipulation," *IEEE Transactions on Human Factors in Electronics*, Vol. HFE-8, No. 1, March 1967, pp. 5-15.
3. Engelbart, D. C., "Design Considerations for Knowledge Workshop Terminals," *AFIPS Conference Proceedings*, 1973 NCC, Vol. 42, pp. 221-227.
4. Engelbart, D. C., R. W. Watson and J. C. Norton, "Augmented Knowledge Workshop," *AFIPS Conference Proceedings*, 1973 NCC, Vol. 42, pp. 9-21.
5. White, J. E., "A High-Level Framework for Network-based Resource Sharing," *AFIPS Conference Proceedings*, 1976 NCC, Vol. 45.
6. Postel, J. B. and J. E. White, *Notes on a Distributed Programming System*, Augmentation Research Center, Stanford Research Institute, Menlo Park, California, March 1975.
7. Roberts, L. G. and B. D. Wessler, *The ARPA Network*, Advanced Research Projects Agency, Information Processing Techniques Office, Washington, D.C., May 1971.
8. *L-10 Users' Guide: Content Analyzer*, Augmentation Research Center, Stanford Research Institute, Menlo Park, California.
9. Irby, C. H., "Display Techniques for Interactive Text Manipulation," *AFIPS Conference Proceedings*, 1974 NCC, pp. 247-255.
10. Lehtman, H. G. and K. Kelley, et al., *Query/help Software and Data Bases*, Knowledge Workshop Development Final Report RADC-TR-75-304, Augmentation Research Center, Stanford Research Institute, Menlo Park, California, June 1974.
11. Grignetti, M. C., C. Hausmann and L. Gould, "An Intelligent" Online Assistant and Tutor—NLS Scholar," *AFIPS Conference Proceedings*, 1975 NCC, pp. 775-781.
12. Irby, C. H., "The Command Meta Language System," private communication.
13. Bobrow, D., et al., "TENEX, A Paged Time Sharing System for the PDP-10," *Commun. ACM*, Vol. 15, pp. 135-143, March 1972.
14. Teitelman, W., et al., *INTERLISP Reference Manual*, Bolt Beranek and Newman Inc. and Xerox Corp., 1974.