AD-A010 186

AN INTELLIGENT TUTOR: ON-LINE DOCUMENTATION AND HELP FOR A MILITARY MESSAGE SERVICE

Jeff Rothenberg

University of Southern California

Prepared for:

Advanced Research Projects Agency

May 1975

ARPA ORDER NO. 2223

ISI/RR-74-26

May 1975

Jeff Rothenberg

An Intelligent Tutor: On-line Documentation and Help for a Military Message Service

INFORMATION SCIENCES INSTITUTE

UNIVERSITY OF SOUTHERN CALIFORNIA

4676 Admiralty Way/Marina del Rey/California 90291
(213) 822-1511

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: ISI/RR-74-26
2. GOVT ACCESSION NO.:
3. RECIPIENT'S CATALOG NUMBER: AD-A010 186

4. TITLE (and Subtitle): An Intelligent Tutor: On-line Documentation and Help for a Military Message Service

5. TYPE OF REPORT & PERIOD COVERED: Research

6. PERFORMING ORG. REPORT NUMBER:

7. AUTHOR(s): Jeff Rothenberg

8. CONTRACT OR GRANT NUMBER(s): DAHC 15 72 C 0308

9. PERFORMING ORGANIZATION NAME AND ADDRESS:
USC/Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90291

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: ARPA Order #2223

11. CONTROLLING OFFICE NAME AND ADDRESS:
Advanced Research Projects Agency
1400 Wilson Blvd.
Arlington, Virginia 22209

12. REPORT DATE: May 1975

13. NUMBER OF PAGES: ~~39~~ 38

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): -----

15. SECURITY CLASS. (of this report): -----

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: -----

16. DISTRIBUTION STATEMENT (of this Report)

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

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

18. SUPPLEMENTARY NOTES: -----

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

Author-language, computer-aided instruction, documentation, error reporting, help, trial, tutorial, verbosity

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

(OVER)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE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

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. ABSTRACT

The military messoge service proposed by ISI's Informotion Automotion project is designed to provide full documentation, help, ond error-reporting facilities on-line. The Tutor serves these functions by occessing a documentotion (or Help) dota bose which contains multilevel descriptions for every "semontic entity" used in the interfoce between the service ond the user. These descriptions ore expondable with respect to the omount ond type of informotion presented, os well os with respect to the user's level of proficiency ond experience, os indicoted by a User Profile. The Tutor olso provides o facility for on-line computer-aided instruction. It can be invoked explicitly by the user's request for help, or by the Commond Language Processor ond User Monitor in response to unrecognized commands, inefficient operation, or error conditions.

ARPA ORDER NO. 2223

ISI/RR-74-26

May 1975

Jeff Rothenberg

An Intelligent Tutor: On-line Documentation and Help for a Military Message Service

INFORMATION SCIENCES INSTITUTE

UNIVERSITY OF SOUTHERN CALIFORNIA

4676 Admiralty Way/Marina del Rey/California 90291

(213) 822-1511

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

CONTENTS

PREFACE

This report is one of a planned collection of reports that describes the current status and plans of the Information Automation project. It is intended to be read by ARPA personnel, Computer Science personnel, and military personnel interested in computer-based message handling design and implementation. Specifically, this report describes the project element called the Tutor, whose purpose is to provide integrated on-line assistance, documentation, and error reporting.

The Information Automation (IA) project [1] is currently developing methods to automate various information handling tasks, with particular emphasis on message processing for military command, control, and communications [2]. The project is sponsored by ARPA, and is an integral part of both the client's and ISI's overall program to explore the utilization of computer technology and methodology in military environments.

Other project elements are referred to where appropriate, but are not defined herein, since they are described in detail elsewhere. For a more comprehensive discussion of other project elements, the reader is referred to project documentation noted in the references. The primary modules of importance to the present discussion are the Command Language Processor (CLP) [3], the User Monitor [4], and the Executive (Exec) [5].

Preceding page blank

SUMMARY

The military message service provides full documentation, help, and error-reporting facilities on-line.

The IA Tutor serves these functions by accessing a documentation (or Help) data base which contains multilevel descriptions for every "semantic entity" (or "term") used in the interface between the service and the user. These descriptions are expandable with respect to the amount (verbosity) and type (sophistication) of information presented. They are expanded in accord with the user's level of proficiency and experience with the service, as indicated by a User Profile.

In addition to purely textual descriptions, the Tutor provides a tutorial facility for on-line, computer-aided instruction (CAI). Tutorials allow the user to go off and "try" things with the service in a protected mode.

The Tutor can be invoked explicitly by a request for help from the user, or by the Command Language Procesor (CLP), or User Monitor, in response to unrecognized commands, inefficient operation, or error conditions.

The Tutor interacts with the CLP [3] and the User Monitor [4] to suggest to the user ways he can improve his effectiveness with the service. These include replacing command forms with alternate ones, suggesting alternate command sequences, and creating new commands.

In the normal mode, the Tutor "hypothesizes" what help the user wants, and allows him to "refine" or "correct" this hypothesis by asking explicitly for something different.

The Tutor is conceived as an omnipresent helpful adviser, always ready to answer questions, make suggestions, interpret errors, and explain problems. This facility is considered essential to the success of an on-line service in an end-user environment.

1. INTRODUCTION AND OVERVIEW

The Information Automation (IA) project [1] is currently developing methods to automate various information handling tasks, with particular emphasis on message processing for military command, control, and communications [2]. The usefulness of this on-line service depends partly on how well the service itself can help its users. A fully integrated on-line Help facility can greatly enhance the value of such a service. Military users need a service they can use effectively, no matter how little or long ago they were trained in its operation.

It is the purpose of the IA Tutor to help make the service usable by (and understandable to) its end-users.

To perform its task, the Tutor handles four interrelated functions:

- Help
  (answering the user's questions or offering advice)

- Introduction of new features
  (e.g., when the user tries something for the first time)

- Documentation
  (for reference and review of things the user already knows)

- Error reporting
  (to provide helpful, rather than cryptic, error messages)

All functions are handled uniformly, in that the user always has the same options (including various forms and levels of documentation) when interacting with the Tutor: this interaction is referred to generically as "Help".

The term "documentation" is used to encompass all forms of information about the service available to the user (including interactive forms such as tutorials).

It must be kept in mind that the Tutor is a facility for documenting and providing help within the service. The actual documentation strategy can be specified only after a particular target community has been studied in detail and an actual command language has been chosen for that community. Thus the Tutor is designed as far as possible to access tables which contain the actual documentation rather than embodying documentation in code.

The next sections of this document describe the forms of help the Tutor provides across all functions. The functions listed above are then discussed in detail.

The final sections describe the logical structure of the Help data base, the interface between the Tutor and other modules (including the Functional Modules) [5], which allows "table-driven" documentation, and the framework provided for writing Tutorials (procedural documentation).

2. *LEVELS OF DETAIL PRESENTED TO THE USER*

The IA service is designed to present an interface which is tailored and adaptable to its users. To this end, a User Profile is kept for each user (maintained jointly by the User Monitor [4] and the Tutor), which records what kinds of operations the user has performed, how much on-line training he has received, and how well he performs various functions.

One of the prime functions of this User Profile is to allow the Tutor to tailor the help it provides to a particular user.

The next section explains the role of the User Profile in varying the Tutor's responses to the user. This section describes how the Tutor module varies those responses for users at different levels of expertise. The primary axes along which responses vary are

- Verbosity (how much explanation is given)
- Sophistication (what kind of explanation is given)

The following applies to the Tutor module in *all* its functional capacities, whether invoked explicitly by the user or initiated on the user's behalf by the service.

VERBOSITY

Depending on User Profile criteria described below (Chapter 3), the Tutor module may initially respond to the user at any of the following levels (if the initial level provides insufficient help, succeeding levels will be tried until the full gamut has been run):

1. Unobtrusive
2. Terse
3. Intermediate
4. Verbose
5. On-line manual
6. Tutorial
7. Expert advice

This continuum of verbosity is discussed below. The examples given are meant to be illustrative only. They do NOT represent the actual command interface language,

since this is only specified for a particular user community after detailed study. In all examples, capitalized words are terms the service knows about. The user can selectively ask for more information on any one of these terms (see Chapter 4 below).

The example used below is the hypothetical command form:

TRANSMIT (MESSAGE-NAME, [to] ADDRESSEE-LIST)

Unobtrusive

When an experienced user who is already familiar with the system in general and with the feature in question (as indicated by the User Profile) makes a careless or typographical error, he does not want to be interrupted by half a page of explanation about something he already knows. In some cases the CLP can actually correct these errors and continue. In others, the Tutor is called, and it may ignore the mistake entirely, or at most ask the user to repeat whatever he was trying to do by means of a "?" or simply repeating the original prompt.

Example:

The user types the error

TRANSMIT (REPORT1\

The Tutor either just beeps, or types back

"TRANSMIT (REPORT1 ?"

Note that this mode of response ***cannot*** occur for a naive or inexperienced user.

In addition, this response can never result from a ***request*** for help by the user, but only from trivial error conditions.

Terse

Based on the User Profile, the Tutor module chooses terms the user knows, and accesses a previously written sentence fragment (generally a few words). This is really just an expanded prompt.

The message service supports a number of different language forms for each command as described in [4]. The Tutor module always looks at what language form is being used so that the "terse" help provided always corresponds in style to the

particular form. This means, for instance, that if the language form in question has a three-word prompt, the Tutor responds with more than that. The Help data base and language forms (prompts, keywords, etc.) are integrated so the user never has to use multiple terminologies.

Example:

The user hits the HELP key after typing

TRANSMIT (REPORT1,

The Tutor responds with

"TRANSMIT [sends]

(MESSAGE-NAME [report1], [to ?])"

Intermediate

This is intended to be a brief (one or more sentence) explanation in English (as opposed to the "fragments" used in the terse form).

Example:

The user asks for more than the terse response above and gets

"TRANSMIT sends a MESSAGE to a LIST of ADDRESSEES (or to a single ADDRESSEE). It is entered as:
TRANSMIT (MESSAGE-NAME, [to] ADDRESSEE-LIST)"

Verbose

This response begins to provide some context and may be on the order of a paragraph.

Example:

The verbose form of the above example might be

"The TRANSMIT command sends a MESSAGE to a LIST of ADDRESSEES (or to a single ADDRESSEE). The MESSAGE NAME is the first ARGUMENT. The second ARGUMENT is either a single ADDRESSEE, as in

TRANSMIT (REPORT, [to] J6)

or a LIST of ADDRESSEES, separated by commas, as in

TRANSMIT (MEMO,[to] Col. Jones, J6, John Smith).

MESSAGES are sent out to all ADDRESSEES at the same time."

On-line Manual

Essentially, this provides all necessary context, relevant concepts, cross-references, etc. It is basically stored text. However, the Tutor provides the user with ways to select what he wants to see (see Chapter 4 below), so that he need not read through large sections of the manual. (No example is given, since there is generally a large amount of text available in the on-line manual.)

Tutorial

This is a mixed procedural-textual mode which leads the user through a sequence of explanations, questions, tests, examples, and trials. It is discussed in more detail in Chapter 6.

Note that there is a "procedural" aspect to all user interactions with the Tutor, since the user can always select and control what help he gets, even when it is simple text. The tutorials merely carry this procedural aspect further.

Expert Advice

This is not a facetious last resort: The user may have questions which require human intelligence to answer. In such cases, the user is referred to one of several human experts for further help. People he knows (indicated in the User Profile) are given preference if they satisfy the "expert" criterion.

In a large installation, it may be effective to maintain one or two experts on-line who can communicate with users in trouble. Such experts would require access to the tools for interpreting the user's context, history, Profile, etc., normally used by the Tutor.

SOPHISTICATION (access categories)

For documentation purposes, the service is organized around "semantic entities", which include all commands, data-type names and other terms used in the service (see Appendix I). Each of these is documented under several standard headings called "access categories" (see Appendix I). These categories are designed to provide an ordered sequence of increasingly sophisticated explanations. An unsophisticated user may simply want to know what some command does, whereas a sophisticated user may want to know its side-effects or the context in which it works.

The user can explicitly ask for information via any one of these categories (see Chapter 4). Otherwise, the Tutor estimates his sophistication via the User Profile (see Chapter 3) and provides him with documentation from the appropriate categories.

The next section describes how the Tutor selects the level of detail to be presented to the user.

3. VARYING THE LEVEL OF DETAIL PRESENTED

The previous section described the varying levels of detail which the Tutor can present to the user. This section deals with ***how*** the Tutor selects the appropriate levels for a given user. The statistics referred to below are gathered and maintained by the User Monitor. The techniques for gathering them, and the data structures kept, are described in [4].

For ***each*** access to the Help Data Base (new Help request, expansion of keywords in a previous Help response, etc.) the User Profile is consulted for the user's history on the semantic entity in question, and appropriate documentation is selected. The parameters used for varying the Tutor response to a particular user are given below:

- Computer naivete (user experience with computers)
- User experience with this service
- Frequency of use and performance on some particular feature
- Repetitions of a request for help

Each of these measures is discussed in turn below.

COMPUTER NAIVETE

This is an initial measure of expected receptivity to the service, based on the categorizations of users (determined by pretesting) described in Chapter 5 of Ref. [4]. Users who have been exposed to computers previously are less "computer naive", and are likely to be more receptive (unless their experiences were bad, in which case they may be less receptive). In any given target installation, there might be several levels of users with respect to computer naivete (data processing personnel, clerks, generals, etc.), and there would be several corresponding initial Profiles for these classes of users.

The Tutor module will initially have a single Help data base used for all users (though later implementations may develop different documentation for each class of user). This information will be used by the Hypothesizer (described below in Chapter 4) to determine how "deep" into a given explanation the Tutor module should go for a first response to a given user. For example, computer naive users are given tutorials on new topics in preference to simpler documentation which might be enough for a computer programmer using the service for the first time. In addition, computer naivete (sophistication) conditions the Hypothesizer's selection of an access category whenever the Tutor is describing a term to the user.

USER EXPERIENCE WITH THE SERVICE

This will be measured along several axes as per statistics kept by the User Monitor (see [4]):

Total Time on the Service

This modifies the computer naivete measure, under the assumption that the longer the user has been exposed to the service, the more familiar he is with it and the more sophisticated he is in its use. It is a coarse measure, since the user may have spent three months performing relatively few kinds of tasks.

General Knowledge Level

This is measured by the fullness of the Transaction Relative Frequency Distribution Matrix (TRFDM) (Chapter 11 of [4]), for which the User Monitor keeps several measures. It is combined with the user's total time on the service to produce an estimate of his level of experience.

General Proficiency

The User Monitor similarly keeps performance measures over the entire transaction matrix (see Chapter 11 of Ref. [4]), which indicate how well the user does in general with the service. This tells the Tutor module what to expect when introducing a user to something new, and how likely he is to make mistakes.

Familiarity with a Particular Feature

The Transaction Training Statistics (TTS) file in the User Profile provides a measure of the user's knowledge level of a given feature, along with an indication of how much previous help has been provided (e.g., whether or not he has had a tutorial on this subject). In general, the Tutor module tries not to give the user a verbose explanation for something he probably knows about. Note that this is overridden when the user repeats a request for help, indicating that he wants to see more.

Recency

All the "knowledge-level" and "familiarity" information the Tutor accesses about the user is tempered by a time axis, so that if the user has been on leave for three months he is not still expected to remember things he knew intimately before his leave. This is provided free by the User Monitor to some extent, since it keeps statistics within a

sliding window of time which tends to forget old history; however, the Tutor can also look at time stamps associated with when the user last did something.

FREQUENCY OF USE AND PERFORMANCE

These measures in the TRFDM give a good indication of what kind of help to provide. Just as important, they can tell the Tutor when it is *unlikely* that the user is in doubt about something, and this can improve the Tutor's hypotheses as to what the user wants help with.

REPETITIONS OF HELP REQUESTS

In general, if the user asks for help twice in a row, he probably does *not* want to see the same thing twice. If the first response (e.g., to an experienced user) was a terse description, the second response should contain more detail. (There is an important exception to this: especially for verbose outputs (on the order of a screenful of text or more), the user will frequently want to redisplay what he has just seen in order to reread it. The service-wide REDO command will be used for this purpose by the Tutor unless the Screen Control Module [3] supports some way to scroll back what has already appeared on the screen.) This increasing help terminates eventually, since there is only a finite depth to the help offered, but the Tutor always realizes that the user *wants* more help. The ultimate level is to refer the user to a person who can provide further help.

4. SELECTION OF HELP

The essence of the Tutor module is to be helpful. This requires that the user be able to interact with the Tutor to get the kind of help he needs. Before it can help the user, the Tutor must know *what* he needs help with, *what kind* of help he needs (what he wants to know about it), and *how much* help he wants. In addition, the user must always be able to find out *how* to get the help he needs (that is, how to tell the Tutor what he wants).

The Tutor prefers to be *active* in this interaction, asking the user questions to ascertain what he wants rather than having to deal with arbitrary free-form inputs from the user. However, the user is always in control.

EXAMPLE QUESTIONS

The following examples should give some idea of the kinds of questions the user needs answered. These are examples of questions the user may have, *not* of the actual forms he might use in asking them. (These forms are dependent on the command language.)

1. Explain something specific:

   "Explain term x"
   "What *would* happen *if* I did x ?"
   "What does ABORT do (now) ?"
   "What does command "q" do ?"
   "What are the side-effects of doing x ?"
   "Does x have the effect z ?"

2. Explain current state:

   "What do you (the Tutor) want ?" (when interrupted because of some error during command input)
   "What's wrong with what I did (e.g., how did I get here [into Tutor])?"
   "Why did the service produce that last output (e.g. "[confirm]") ?"
   "What did I just do (or what did I do 3 commands ago, or what have I been doing)?"
   "What are my options at this point ?"
   "What *would* happen *if* I did this (what I'm about to do) ?"

3. Explain *how* to do something:

"How can I abort (or undo) what I just did (or something else) ?"
"What else do I need to do to go on from here ?"
"How do I ask you (Tutor) a specific question ?"
"How can I perform some operation (not necessarily an executable command) x ?"

The above groups of questions are handled as follows:

1. The user asks about some term and then selects a particular aspect (access category) of that term (e.g., the side-effects of x).

2. The user asks about the special term STATE (synonyms: WHERE-AM-I, etc.). This is explained under the categories: options, function, etc. (see Appendix I) which allow asking various things about the current state. The CLP is consulted to produce output for this term by looking at what the user is in the process of doing (see Appendix II). The user can also invoke a tutorial at this point to try proceeding in the Protected Mode.

3. The user accesses the "how" category of the Help data base (see Appendix I), which explains how to proceed, or how to use a particular term if one is supplied.

THE HYPOTHESIZER

The Tutor may be invoked either by the user or by the service, as discussed below (see Chapter 5). In all cases, there exists an invocation context which consists of the recent transactions that had occurred between the user and service at the moment of invocation. This includes various aspects of the CLP's "parse state" (see Appendix II), partially completed commands, error conditions, User Monitor "suggestions", and recently used names of relevant data types.

Using this context, along with the User Profile, the Tutor prepares a hypothesis for what the user wanted help with (even if the service invoked Help *for* the user). The normal mode for the Tutor is to prepare a hypothesis for what the user wants, at each invocation and attempt to provide it. (The actual help presented is further conditioned by the User Profile, as described above in Chapters 2 and 3.)

In addition to this "hypothesizing" capability, the Tutor provides a uniform framework of interactions whereby the user can refine, correct or ignore the Tutor's hypotheses and ask for something different. By assumption, the Tutor never provides verbose responses on the first invocation, so incorrect hypotheses will not be overly annoying. Whenever the user is in the presence of the Tutor, he has certain universal options:

- Ask for help about some specific item, regardless of the Tutor's hypothesis (either type the name of a term or point to one already displayed)

- Ask for options versus explanation

This is not normally a Tutor function, since the CLP (see [3]) provides for showing the user the legitimate values that he can type for an argument, in response to a "?" (or similar input). For example, if the user has written (but not sent) messages named REPORT1, MEMO, and MUNITIONS STATUS, and then he enters

"TRANSMIT (?"

the CLP responds with

"[MESSAGE-NAME]:

REPORT 1
MEMO
MUNITIONS STATUS."

However, the Tutor also provides this option in case the user hits Help by mistake when he really meant "?". (It is the intent of the service to be helpful and not to punish the user for mistakes.)

- Refine the Tutor's hypothesis in the following ways:

  More local (e.g., arguments rather than commands)
  More global
  Refer to something done earlier/later than
  that hypothesized
  Different access categories
  Get help with getting Help

- Expand the help shown

Whenever there is documentation shown, whatever the level, the user can always ask for more (or less) along each of several axes (see Appendix I).

- More/less verbosity
- More/less sophistication
- Different access category
- Expand any keyword shown in the currently displayed explanation

5. TUTOR FUNCTIONS

The above sections have described the Tutor/user interface, which is the crux of the Tutor. Chapter 6 discusses tutorials per se. This section outlines each of the major functions of the Tutor, which are performed by means of this interface.

HELP

The basic idea of the Help function is to provide good first approximations to ***what*** the user needs help with, and then to allow ***painless*** interaction for the user to ***refine*** (or correct) these hypotheses. These approximations are based on the User Profile, the CLP's state (the status of the command in progress), and the overall context of what the user is doing.

This function can be factored into cases depending on how "active" the service is in offering help:

Requested by the User

The user has a Help key which he can ***always*** hit to get relevant help and advice about what he is doing. This may include descriptions of commands, menus of allowed arguments or alternate techniques, or accessing a dictionary to check spelling when the user is entering text.

Suggested by the User Monitor in "Background"

On the basis of statistics gathered between sessions, the User Monitor arrives at recommendations for changes in the user's habits and language forms such as Inefficient Dialogue Elements (i.e., language constructs that do not prohibit useful work, but do lead to poor performance), as well as Recurrent Dialogue Sequences that might be replaced with shorter constructs (see Chapter 9 of [4]). The Tutor accesses these "suggestions" through the Potential Dialogue Improvement file (PDI) in the User Profile. The User Profile also contains a preference for each user as to ***when*** such suggestions should be made (the default is at the start of a session, ***or*** on request). The Tutor then suggests these changes to the user at the appropriate time. (The user can always choose to ignore this advice.)

Suggested by the CLP as User Monitor in "Real-time"

On the basis of real-time measurements arising from the CLP's inability to parse an input, the User Monitor may also arrive at recommendations to be made immediately to alleviate the error condition. This is done by replacing the dialogue element that could not be parsed with an alternate form, as described in [4]. The Tutor is then invoked to make suggestions to the user, providing help as if the user had requested it. (In most of these cases, the user can ignore the suggestions and continue with minimal interruption.)

For details of the actions taken by the Tutor in these cases, see Appendix III and [4].

Errors

All modules are expected to detect errors and pass control back to the Tutor, which then acts as if the user had requested help about this error condition.

INTRODUCTION OF NEW FEATURES

This is not identical with "documentation" per se (as covered below), since it involves cases in which the user is expanding what he knows about the service.

There are several cases, all handled by essentially the same mechanisms.

New User of the Message Service

New users are described by default Profiles (which are pre-tailored on-site), and are introduced to the basic service by means of tutorials written for the particular user population. The intent of this introduction is to be self-sufficient after minimal (less than one hour) individual/classroom training which concentrates on using the terminal, editing text, and getting help.

User Requests to Expand His Capabilities

Either out of curiosity, anticipated need, or immediate need, the user may want to find out about something he has not yet done. This is similar to the introduction phase above, except that once the user has used the service, his Profile contains information relevant to further training. The language forms he knows and prefers, and his error rates using certain constructs are used to optimize the introduction of some new topic to facilitate his learning to use it effectively.

The main problem here is one of the user's communicating with the Tutor about what he wants to know. The major cases are as follows:

a. The user asks for an expansion of something he knows, or something related to what he knows - this is the simplest case, since the Help data base is hierarchical, with the User Profile determining how deep the description goes. In this case the Tutor modifies the Profile to allow greater depth of explanation. (Note that documentation has both depth of explanation for any single item and breadth, which extends to related items.)

b. The user asks about something he has heard of from another user - in this case he will use terms that should be familiar to the service, except that user-defined synonyms (and "macros") require that the Tutor ask whom he heard it from (to access the proper private definition).

c. The user asks how to do something. That is, he asks for help by "function", where the terms he knows may not correspond to service functions - this is not handled in the general sense of English-language requests for information, as the service does not support English sufficiently. The main approach is to present menus (see next item).

d. The user selects the item to be explained from a menu, which can zero in on what he is looking for.

User Trying to Do Something He Doesn't Yet Know Enough to Do

It is unclear whether the CLP is even able to detect this case. If not, it may still arise when the user begins something and then asks for help when he finds he can't proceed, which should reduce to case (c) above.

When the user tries a command (or a form of a command) he has never used before, the CLP generates a warning to the Tutor. Based on the User Profile (which reflects not only what commands the user knows, but also how much he likes to experiment) the Tutor gives the user one of several levels of warning, essentially asking if he wants help before trying this.

This case is tricky, because a novice user may make a typographical error that changes the simple command he wanted into some complex command he has never seen. He must not be further confused by the Tutor asking if he really meant the complex command. The only handle on this situation is the user's general level of knowledge (as indicated by the User Profile). This can at best *suggest* that he really meant the simpler command. Also included here are "ghost user" issues (an experienced user

sitting down at the terminal of an inexperienced user, without telling the service so that it can switch Profiles). Should the user be denied the use of the complex command until he has confirmed that he really meant it? Should the Tutor come back and ask if he is really whom he claims to be? These issues are deferred for now.

DOCUMENTATION OF THIS USER'S SERVICE

Much of the documentation issue has already been addressed above. The remaining functions are

On-line Manual

Essentially the same as Help, this presents a different organization to the Help data for this user. It gives a coherent view of the service (or some aspect of it), in a form to be read on-line, using terms this user knows.

Off-line Manuals

The Tutor has the capability to tailor off-line manuals in terms known to a particular user, providing summary sheets, full-scale manuals, or levels of detail in between.

Translation Among Users

Though the differences between the services seen by any two users at the same level of proficiency is not expected to be great, it may still be helpful to provide a translation capability for users to talk with each other (either in messages concerning the service, or over their terminals) so that each one sees terms and forms he is familiar with. Since the service is in some sense primitive-based, this is relatively straightforward and can be provided by the Tutor if warranted. This is an area for possible research and is not included in the initial implementation.

ERROR REPORTING

There are three kinds of errors which must be fielded for the user: unrecognizable or malformed commands detected by the CLP, errors returned by the Functional Modules being invoked, and system errors (e.g., resource limits).

The Tutor is responsible for telling the user about ***all*** errors so that he never sees an uninterpreted error message. In line with the overall Help approach, the actual message shown to the user is determined by his Profile so that experienced users get more succinct (though never cryptic) messages than new users. In addition, the expected frequency of occurrence of an error determines how much explanation is given at any level: thus a rare error is explained carefully even to an experienced user.

It is imperative that *all* errors of any kind return control to a single responsible party (namely, the CLP) which in turn invokes the Tutor. The Exec provides several crucial facilities for allowing modules to report their status when an error occurs, including an error-stack which the Tutor can interrogate and a status-text area for each process to record its current state (see [5]).

The Tutor requires of *each* service module (including Functional Modules) that it

- Report all errors to a (non-Tutor) Error-Handler (which saves state as necessary for possible recovery).
- Define recovery procedures (if any) for each error (these are only of concern to the Tutor in that it must be able to talk to the user about them).
- Define the semantics of the error and the recovery procedures for the Tutor in the same way that Semantic Primitives are defined by Functional Modules.

Errors are explained to the user just as if he had requested help on the subject--that is, more explanation is always available, including advice from the Tutor on how to cope with the error. This approach assumes that when an error occurs, the user doesn't merely want to know that it occurred, but also wants to do something about it (recover, circumvent it, etc.).

The Tutor does not consider its responsibility to extend to error recovery per se. That is, issues such as where control is returned after an error is explained to the user, how much of the original state is saved, etc. are not of direct concern to the Tutor. However, the Tutor must explain such things to the user.

The three error types require somewhat different handling, as discussed below.

Command Syntax Errors (recognized by the CLP)

This case involves the CLP, User Monitor and Tutor interaction (see [4] and [3]). The details are contained in Chapter 9 of Ref. [4], which describes the User Monitor. Basically the CLP detects and reports such errors, the User Monitor may offer suggestions for what action to take, and the Tutor makes the suggestions to the user. The Tutor consults the User Profile here (as always) and avoids getting in the way of an experienced user who is just making typographical errors. The help provided is conditioned also by the state of the CLP's processing of the command, which indicates where the user made the mistake (or what the CLP doesn't understand).

When the user makes a mistake, the Tutor module is responsible for advising him on how to correct it or suggesting how he can perform the required action. This may involve getting suggestions from the User Monitor (which may in fact have invoked the Tutor). In order to get relevant suggestions, the Tutor may interact with the user to determine, for instance, whether the problem involves the current command or the response from the previous command. The Tutor then passes this information back to the User Monitor to get an appropriate suggestion.

Functional Module Semantic Errors

These consist basically of semantic errors which the CLP cannot detect, such as referring to a nonexistent message. The CLP/Functional Module/Tutor interfaces allow the Functional Module to return an error to the CLP which the Tutor can explain to the user in terms he understands. The definitions of functional module commands include descriptions of error conditions and recovery procedures so that the Tutor can communicate meaningfully with the user on this subject.

System Errors

These include all resource conflict or limit errors (such as lack of file space, unavailability of some device, etc.). In some cases, the user need never be informed of the error, since the CLP can find a way around the problem and produce the result the user wanted. However, in other cases the user must figure out his own solution; in these cases, the Tutor must provide advice on how the user can recover. This is greatly enhanced if ***all*** system limit-errors are guaranteed to be "soft" so that the user is warned before the limit is actually reached.

6. TUTORIALS

The topmost level of documentation contained within the service proper (not including referring the user to an expert for advice) is the Tutorial. It is characterized by being more procedural than the simpler forms of documentation, even though it still makes use of previously written text elements. It also provides a special protected environment for the user to "try" things without risk of erasing or sending messages by accident. The Tutor module provides this environment indirectly, by conditioning the actions of the CLP and Exec. The Tutor also provides a basic CAI language [7] for writing tutorials.

The intent of the tutorial facility is to provide a documentation mechanism which makes the user more active (by responding to questions and trying things out) so that he will overcome any reluctance to use the service.

ENVIRONMENT

The Tutor essentially does two things to enable the Protected (or Tutored) Mode. It insures return of control from the CLP and guards against permanent modification of data (via the Exec).

Control

The service is already assumed to return control to the CLP under all conditions. The primary effect of this mode is simply to tell the CLP to return control to the Tutor whenever the user executes a command in this mode.

The only real extension required by this mode is that the Tutor will want to time the user's response and force control to return to the Tutor after a certain elapsed time, regardless of what the user has done. (Even if the CLP normally performs a similar function, the Tutor may want different time limits for different commands, and has to be able to set this time explicitly.)

The user can escape from this mode by means of the ALERT-CLP function, (assumed to be a single keystroke or control character) which *always* returns him to the top level, where he can talk directly to the CLP.

Since the CLP is always parsing, whether the user is in the Tutor simply for help or is in the Tutored Mode, a legal command to the CLP is always recognized. It is at the

CLP's discretion, based on its own interpretation of the User Profile, whether to perform the command or not. For example, the user may have asked for help on one command and may suddenly realize he wants to perform some other command. Of course, he can always get out to the CLP (by using ALERT-CLP), but if he simply types the legal command, the CLP still parses it and may just do it if the user is a sophisticated one. Otherwise, the CLP can either ignore the command, object to it, create a new invocation of the Tutor to deal with it, or (normal case) simply report the action to the present invocation of the Tutor so that it can deal with the spurious input in the context of the original request for Help.

An exception to this discretion arises when the user is typing an answer to a Tutorial's Input MATCH statement: in this case, the input must be passed on "quoted" by the CLP (see Appendix II below).

Similar situations arise for commands the user has never typed before. These issues are left to the discretion of the CLP.

Data Protection

In the Tutored Mode, the Exec allows the user to perform ***almost any*** function without risk. (The ALERT-CLP function, for example, ***must*** always remain enabled, so the user can still perform a "dangerous" function (e.g., erase text or send a message unintentionally) by first performing ALERT-CLP). For the majority of functions, however, this mode protects the user from inadvertently deleting his own (or anyone else's) messages, sending practice messages to his commanding officer, and the like.

This mode is made somewhat tricky (for the service) by the requirement that when the user tries something in the Tutored Mode it should perform ***as far as possible*** as it would if he were not in this mode. The alternative is to simulate the results and pretend to the user that the service has done what he asked, but this is avoided wherever possible in the belief that sooner or later this leads to discrepancies between how the service appears to the user when he is "in" as opposed to "out" of the Tutored mode.

The Exec provides this protection by catching all "dangerous" commands whenever the Tutored Mode is on, and either copying any text that gets changed or simulating the action of the command, as appropriate.

TUTORIAL LANGUAGE

The tutorial language is intended to run in the environment described above. It provides facilities for creating interactive lessons.

The language is described under three types of facility: Input, Output, and Control.

Input

Pattern matching is provided to allow a lesson-writer (author) to specify fairly flexible allowed responses in a MATCH statement. The basic technique is to match keywords provided by the author. These can be AND'ed, OR'ed, or required in sequence. Sequences may allow other words to be embedded (the default) or not. Initially, this facility will be kept as simple as possible, while providing the power to build patterns to match reasonable sets of inputs.

In addition to alternate terms and synonyms allowed by the author of the tutorial, the service thesaurus (which supplies synonyms for terms) and user synonyms (from the User Profile) will be automatically accessible for matching, unless the author turns this feature off.

(When spelling correction is available elsewhere in the service, it will be provided here also.)

Output

The author can provide explicit output and he can also access the full Help data base for output. This allows lessons to make use of the Help documentation for their own purposes. When this is done, the author can select all axes (verbosity, sophistication, etc.) explicitly, can invoke the normal Tutor processes for selecting on the basis of the User Profile, and can enable or disable the normal facility for the user to interact with the Help documentation (expanding, etc.).

Control

When the author provides a response to be matched, several things happen. The user's response (whether it matches or not) is saved in an optional string-variable supplied by the author. The MATCH statement either succeeds or fails, and a MATCH-switch is set for later testing, indicating whether the last MATCH succeeded or failed.

The author thus has the option of splitting up possible response cases into different MATCH statements or separating them later on the basis of the saved response. In addition to the matching condition, the author can supply a time limit so that the match fails if the user does not respond quickly enough.

- A conditional (testing the MATCH-switch) is provided for program flow control.

- The TRY command allows the user to go off to the CLP and try something in the Tutored mode, while setting a time limit after which control returns automatically to the Tutor.

- The tutorial language is embedded in BLISS (the implementation language). Lessons can be written with minimal knowledge of BLISS, while computation and additional facilities are available to sophisticated authors. The intent of this language is to permit qualified CAI authors or service designer/implementers to generate tutorials quickly and effectively.

This CAI capability rounds out the Tutor's repertoire of interactive instruction and Help facilities.

Preceding page blank

APPENDIX I: HELP DATA BASE

The basic form of the Help data base can be visualized as a four-dimensional cube, each of whose axes divides the total documentation space into several discrete levels.

The entire service is documented in terms of "semantic entities" which can be thought of as one axis of the 4-cube. Some of these entities are "semantic primitives" which are defined independently (that is, using "pure" English), while some are defined in terms of other entities.

The semantic entities include

1. All commands and arguments (data-types,
2. Alternate names for entities, given by the thesaurus (see [3])
3. Error-condition terms
4. Terms concerned with dialogue forms
5. Terms for internal service concepts
6. External user concepts (described in terms of the service)
7. Names of user-defined synonyms and macros (when the user defines a macro or synonym for himself, he is asked (by the CLP) to supply a short description for his own use at a later time, which the Tutor adds to the data base for that user, to allow documenting extensions to the transaction matrix.

Each semantic entity is documented by a 3-cube of documentation elements, as defined below.

For any given semantic entity, documentation can be expanded along three additional axes:

- Verbosity
- Sophistication
- Context

These axes are discussed in turn. It should be noted that the hypercube is a ***logical*** model rather than an implementation. The actual documentation will not contain as many discrete "packets" as implied by the 4-cube concept.

VERBOSITY

This has already been discussed above. Documentation is a mixture of noise (descriptive) words which are "pure" English, and keywords (semantic-entity-names) which can be expanded recursively to generate verbosity as needed.

Included in this continuum (as its top levels) are Tutorials and expert advice.

SOPHISTICATION

For each semantic entity, documentation is organized into several categories of "access", which the user can ask for explicitly. In addition, these categories are given an ordering, so that the Hypothesizer can select the appropriate category depending on the user's sophistication (as given by the User Profile)

The categories defined initially are

Function

Describes the basic use or meaning of the term. For a command, this is just the overall function performed by the command.

How to Use

Explains how to invoke functions and how to use data types properly.

Form

This discusses the various forms of dialogue available for commands or data types. In order to insure consistency, this documentation is actually written once for each dialogue form rather than once for each command, and the description of a form is phrased in terms of the command the user is inquiring about.

Options

This describes more advanced uses of commands, options in specifying data types, etc.

Side-effects

Discusses the less obvious effects of commands, or implications of the use of a term with respect to the rest of the service.

Uses

Discusses where a term appears (what a command is used for, where a given data type is used, etc.). This shades over into ***Context*** below.

CONTEXT

This is logically an additional axis of documentation for each entity, but it is provided by a separate access category which explicitly mentions relevant contextual and background information. Since the user can selectively expand items of a description shown to him, this allows a menu-like exploration of context information.

APPENDIX II: THE TUTOR/CLP INTERFACE

Rules are supplied by the Tutor which allow the CLP itself to ask the Tutor to supply the user with help on a particular semantic entity when it detects a problem.

The CLP (and Exec) supply several functions for the Tutor which provide access to the parse state (the CLP's partial recognition of the command the user typed) and the invocation context (all those names and operations which the user has recently referenced).

In all cases, the CLP returns the *name* of a semantic entity (command, argument data type, etc.) which the Tutor can look up in the User Profile to determine appropriate responses based on the user's experience with that term.

The returned item also carries with it an indication of status (completed correctly/incorrectly, incomplete, etc.).

The Tutor is also able to ask, for any partially completed command:

"What are you expecting" (waiting for, allowed successors, etc.)

Each of the following functions takes an argument <item> which can be

command
subcommand
argument
module/service
<context-name data-type>

(The latter allows asking for the last reference to something of a particular data type, e.g., the last message read, the last date specified, etc.).

The access functions are

last (completed) <item>
current (in progress) <item>
previous <item>

"Levels" are defined as going up from arguments or subcommands toward higher level commands, and the following are also provided:

next <item> up
next <item> down
<item> at level n

The CLP parses inputs typed to the Tutor, just as it does for any service module. Several cases are worth distinguishing, however, since the user may type non-Tutor commands when in the Tutor.

User Types Tutor Commands

This is a normal case, where the user types some command to the Tutor itself (e.g., asking for a description of some term). The CLP merely parses the command and passes it on to the Tutor.

User Types a Legal Non-Tutor Command

Here the user may be trying to execute some non-Tutor command, ignoring the fact that he is "in" the Tutor. (Of course, he can always get out of the Tutor with the ALERT-CLP or ABORT functions, but he may not bother with them). The CLP parses this command, recognizes it as legal and not for the Tutor, and decides, at its own discretion (on the basis of the User Profile) whether to honor it (aborting the Tutor) or not. If not, the CLP alerts the Tutor to the fact that a non-Tutor command has been entered. (This is similar to case "b" below.)

User Types to a Tutorial Language Input (MATCH) Statement

a. pure text (CLP ignores it)
b. legal command

This last case represents a tutorial asking the user a question whose answer is a legal non-Tutor command. The Tutor must in this case alert the CLP to pass the input on as text to the Tutor (effectively "quoting" it) without executing it, but the CLP can also perform some cursory parsing to let the Tutor know if the command would have been correct. (In this mode then, the user ***must*** use the ALERT-CLP or ABORT functions to exit the Tutor and again speak directly to the CLP in order to execute a non-Tutor command.)

APPENDIX III: THE TUTOR/USER MONITOR INTERFACE

It is the Tutor's job to maintain the Transaction Training Statistics (TTS) file in the User Profile. This file is updated each time the Tutor helps the user, and it contains a history of what kind of help (and how much) the user has received for each semantic entity (***not*** just commands).

Dialogue Remedy

When the User Monitor determines in real time that a dialogue element is ineffective (e.g., leading to poor performance), it notifies the Tutor, which takes action as follows:

First, the Tutor must determine whether the user was having trouble with the response from the previous command or with the input for the current command. This is done by a simple procedure: the Tutor asks the user.

The algorithm for determining what remedy to suggest in either case is given in Chapter 9 of Ref. [4], and is performed by the User Monitor itself.

In the case of "background" determination of inefficient dialogue elements, the Tutor is concerned with the combined results of the User Monitor's measures of frequency, performance, and knowledge (the Tutor maintains this last statistic itself in the Profile).

Whenever a dialogue element combines low values for knowledge ***and*** [Frequency ***or*** Performance] the Tutor attempts to provide more training.

In addition, the mean and variance of performance alone are examined with respect to changing the dialogue form of a command or splitting it into two commands (see [4]), and Recurrent Dialogue Sequences are examined with respect to suggesting compounds (macros) to the user. Note that the actual macro-building facility is provided by the CLP, not the Tutor proper.

REFERENCES

1. Oestreicher, D. R., J. F. Heafner, J. G. Rothenberg, ***Connect: A User-Oriented Communications Service***, presented at ACM Annual Conference, San Diego, Calif., November 1974.

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

3. Abbott, R. J., ***A Command Language Processor for Flexible Interface Design***, USC/Information Sciences Institute, ISI/RR-74-24, January 1975 .

4. Heafner, J. F., ***A Methodology for Selecting and Refining Man-Computer Languages to Improve User's Performance***, USC/Information Sciences Institute, ISI/RR-74-21, September 1974.

5. Mandell, R. L., ***An Executive Design to Support Military Message Processing Under TENEX***, ISI/RR-74-25 (in preparation).

6. Tugender, R., D. R. Oestreicher, ***Basic Functional Capabilities for a Military Message Processing Service***, ISI/RR-74-23 (in preparation).

7. Rubin, Sylvan, "A Simple Instructional Language," ***Computer Decisions***, Nov. 1973, p. 17-18.

BIBLIOGRAPHY

1. Grignetti, Mario C.; Gould, Laura C.; Bell, Alan; Hausmann, Cathy; Passafiume, Joseph J., ***Mixed-Initiative Tutorial System to Aid Users of the On-line System (MLS)***, Semiannual Progress Report (Phase I), Bolt Beranek and Newman, Inc, May 15, 1974.