AN EXPERT SYSTEM THAT VOLUNTEERS ADVICE

Jeff Shrager
Carnegie-Mellon University
Department of Psychology

Tim Finin
University of Pennsylvania
Department of Computer Science

I. Motivation and System Overview

This paper describes the design and implementation of an expert system that provides novice users with help in using the Vax/VMS operating system. The most interesting feature of our advisor is that it follows the user's interactions with the system and volunteers its help when it believes that the user would benefit from advice. The user need not ask for help or raise an error condition. The adivsor recognizes correct yet inefficient command sequences and helps the beginner become more proficient by indicating how these tasks may be done more efficiently.

What are the the inefficient command sequences that we are trying to recognize? There are several dimensions to inefficiency in operating system interactions: Operating systems provide many features (such as wild cards in file names, lists of verb targets, etc) which are meant to minimize the user's work (e.g., typing). Consider:

```
$PRINT PROGRESS.MEM
$PRINT MEETING.MEM
etc...
```

rather than:

```
$PRINT *.MEM
```

On another dimension, we measure inefficiency in terms of system resources. The system typically provides special functions which perform operations much more sparingly than more general means would permit. Contrast:

```
$COPY NOTES.* OLDNOTES.*
$DELETE NOTES.*.*
```

with:

```
$RENAME NOTES.* OLDNOTES.*
```

The DCL expert that we have constructed recognizes less efficient sequences and constructs help messages that provide either immediate advice or a pointer to a manual or online HELP entry. Following is a sample of the DCL expert's behavior (the first two lines are entered by the user):

```
$COPY TEST.TXT EXP1.TXT
$DELETE TEST.TXT.*
%If you mean to be changing the name of
% TEST.TXT  to EXP1.TXT  you might have
% simply used the command:
%       $RENAME TEST.TXT EXP1.TXT
% The HELP command can tell you more
% about RENAME.
```

II. General Approach

The difficulty of this task is, of course, to recognize when some sequence of commands constitutes a plan that a person is using to achieve a goal. We have approached this problem by collecting a catalog of "bad plans" which novice users often use to achieve common goals. The problem thus reduces to matching command sequences to descriptions of generic plans from the catalogue. In this application, the matching process is complicated by the following issues:

- *Non-contiguity*: The individual commands which make up a sequence might be spread out over a session. Each of the intervening commands may or may not affect the goal which the overall sequence is meant to achieve.

- *Ambiguity*: The mapping from plan to goals is many to many. A given sequence may match several plans. A given plan may be realized by several sequences.

- *The necessity of Extensional Knowledge*: A given sequence may have side effects or use information not directly expressed in the syntax of the commands. In order to recognize which of several possible goals is being attempted one may, for example, have to expand "wildcard" patterned filenames and select names from the current directory which are refered to by the command at hand.

III. Specific Method

Our goal recognition heuristic is driven by an expectation parser [1]. A KL-One like network [2] describes the commands that form the heads of sequences to be recognized. Upon instantiation of any new entry in the net, some action takes place as specified by the parsing object for which the new instance represents a case. The actions can activate, deactivate, or modify other objects in the network. Since the contents of the net direct the parser, changing it can substantially effect the way in which future commands are processed and the actions to take place when they are parsed. In particular: parsing objects are added which recognize commands that come later on in the sequence whose head was just recognized. The action taken on recognition of the last command in a sequence typically invokes a help message. The actions have access to the network contents and thus can tailor the message to the case just recognized (or any other information contained therein since the contents of the net represent everything of which the user has demonstrated knowledge).

IV. A Detail Example

Another example may make this more clear: Several VMS commands permit the user to specify a file into which the output of the command will be directed. For example:

```
$DIRECTORY NOTES.* /OUTPUT=OLDFILES.TXT
```

A more general way of routing output to a file involves the separate commands ASSIGN and DEASSIGN to attach the selected file to the output channel as:

```
$ASSIGN SYS$OUTPUT OLDFILES.TXT
$DIRECTORY NOTES.*
$DEASSIGN SYS$OUTPUT
```

The following recognition sequence takes place: Upon recognition of an ASSIGN command, parsing objects for both the DIRECTORY command and the DEASSIGN command are added to the network. If a DEASSIGN command were to be entered at this point, the DIRECTORY object would be deactivated and the DEASSIGN object would deactivate itself. No goal is recognized. If, on the other hand, a DIRECTORY command were entered, the first DEASSIGN object would be detached and a new DEASSIGN object activated whose action represents a successful goal recognition and would trigger an appropriate help message. Note the necessity of the interim DEASSIGN command in order that an ASSIGN + DEASSIGN pair is not mistaken for one with a DIRECTORY command between them.

V. Implementation Details

In implementation, the user is communicating via a Franz Lisp program which sends the commands to VMS and then runs those that succeed through the goal recognition parser. Only commands, not the inputs of programs, are trapped. Commands which cause errors are not processed because we assume, first, that they will probably be immediately reentered. Also, the error will certainly cause the parser trouble if the problem was syntactic and will cause the goal recognition processor trouble if the error was semantic. The working system recognizes 5 complete goals and various short-cut strategies (combined command targets, etc).

VI. Discussion

There are implications of this research beyond the additional power that it affords expert systems in general. Since the knowledge network contains an object for each command that has been issued one can think of that database as a user profile. We have made only marginal use of this potential by arranging the actions so that advice is only given for commands that the user has not already used. This is done by, for example, removing the head parsing object of the COPY + DELETE = RENAME sequence when the beginner uses a RENAME command (this action is a part of the actions associated with the RENAME parsing object). Other novice-user aids could be tailored to this dynamic user profile.

Our system essentially encodes the pattern recognition knowledge of a consultant and applies these patterns to user input. When inefficiencies are recognized, the automatic observer can point the way to a more effective use of system capabilities by refering the user to the help system or some other expert. Otherwise, it can generate a help message of its own using the context of the particular sequence to construct useful dialogue. Our work represents an attempt to extend the range of interactive advisor systems. User aids currently in service help only users who cause errors which would invoke an error recovery system or those who know how and when to ask for help. We have provided a means by which the system can automatically tutor intermediate level users who do not make trivial errors but who are not using commands in an effective way. We therefore bridge the gap between the introductory user aids and more technically oriented expert advisors.

ACKNOWLEDGMENTS

This work was performed while the first author was enrolled in the masters program in the Department of Computer Science at the University of Pennsylvania. The paper outlines a Master's thesis available as technical report number MS-CIS-81-1 from that department. This work was supported, in part, by NSF grant number MCS 79-08401. Thanks to Ira Winston for construction of the Lisp/DCL interface, lots of useful advice, and excellent Vax systems support. Also, thanks to Peter Buneman for guidance in the morals of software engineering.

REFERENCES and ABBREVIATED BIBLIOGRAPHY

[1] Riesbeck, C. and R. Schank; Comprehension by Computer: Expectation Based Analysis of Sentences in Context; Yale University CS report no. 84.

[2] Brachman, Ron; A Structural Paradigm for Representing Knowledge; BBN Report no. 3605, July 1978.

Ball, Eugene and Phil Hayes; Representation of Task Specific Knowledge in a Gracefully Interacting User Interface; CMU, in AAAI 1980.

Mark, William; Rule Based Inference in Large Knowledge Bases; USC/Information Sciences Institute, in AAAI August 1980.

Genesereth, Michael R.; The Role of Plans in Automated Consultation; Laboratory for Computer Science, MIT.