Contribution to CHI85 addressing the topics: intelligent interfaces, cognitive issues in user modeling and individual differences among users.

Knowledge-based Help Systems

Gerhard Fischer, Andreas Lemke and Thomas Schwab

Research Group on Knowledge-based Systems and Human-Computer Communication
Department of Computer Science, University of Stuttgart
Federal Republic of Germany

Abstract

Our research goals are to understand the nature of, construct and evaluate intelligent interfaces as knowledge-based systems. In this paper we demonstrate the need for help systems as an essential part of human-computer communication. Help strategies are based on a model of the task (to understand what the user is doing or which goals he/she[1] wants to achieve) and a model of the user (to guarantee that these systems are non-intrusive and that they pay attention to the needs of individual users).

We illustrate that passive and active help systems have to be constructed as knowledge-based systems. Two operational systems (PASSIVIST and ACTIVIST) are described to show the usefulness of this approach.

1. Help Systems

The increased functionality of modern computer systems (required by the many different tasks which a user wants to do) will lead to an increased complexity. Empirical investigations have shown that on the average only 40% of the functionality of complex systems are used. Figure 1-1 is based on our empirical investigations through careful observations (e.g. persons using systems like UNIX, EMACS, SCRIBE etc. in our environment) and describes different levels of system usage which is characteristic for

[1] In the remaining part of the paper we will use the pronoun "he" as a generic notation for a user

many complex systems. The different domains correspond to the following:

D_1: the subset of concepts (and their associated commands) which the user knows and uses without any problems.

D_2: the subset of concepts which he uses only occasionally. He does not know details about them and he is not too sure about their effects.

D_3: the mental model (Norman 82; Fischer 84) of the user, i.e. the set of concepts which he thinks exist in the system. A passive help (see section 1.1) system is needed to gradually master the commands in D_2 and D_3.

D_4: D_4 represents the actual system. Passive help systems are of little use for the subset of D_4 which is not contained in D_3, because the user does not know about the existence of these system features. Active help systems (see section 1.2) which advise and guide an user similar to a knowledgeable colleague or assistant are required that the user can incrementally extend his knowledge to cover D_4.

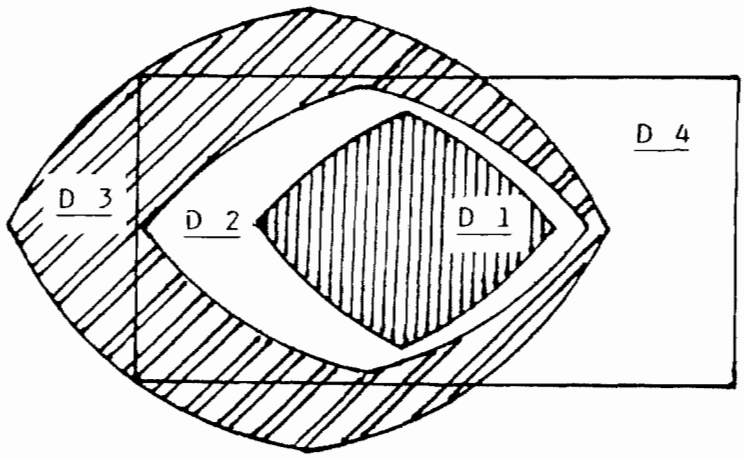


Figure 1-1: Different levels of system usage

Unlike tutorial systems help systems **cannot be structured in advance** but must "understand" the specific contexts in which the user asks or needs help. They provide information only in relevant situations and eliminate the need to learn a lot of things in advance (i.e. at times when it is unknown if they ever will be used and when it is hard to imagine an application). A general discussion of user support systems is given in Fischer/Lemke/Schwab (Fischer, Lemke, Schwab 84).

1.1 Passive Help Systems

Keyword Based Help Systems. Assuming a user knows the name of a command but not its details, keyword based help systems (including synonym lists and pattern matching capabilities) are a quick, easy to implement and sufficiently reliable choice.

Keyword based help systems can be used for explanation of terms, description of commands and function keys, and search for related concepts. They are of little help clarifying general topics where it is difficult or impossible to describe the needed information with a single word. Existing keyword help systems are static and do not take the user's special situation or his level of expertise into account.

Natural Language Based Help Systems (see section 3.1). Observation of human "help systems" suggests that often a dialog rather than a single question and answer is necessary to identify the user's problem or to explain a solution. Natural language based help systems offer the following possibilities:

* the user has **multiple ways** to formulate a problem. Natural language provides much more flexibility than the best synonym lists.
* with natural language **failures are soft**. Most of the user's utterances give at least a hint to where the problem lies.
* **misconceptions** of the user can be identified from his utterance. They give an important clue for building a user model.
* the user can not only ask a specific question, but he can describe his goals, his intentions and his difficulties.

A problem for novices is to find the appropriate words to describe their problems based on a lack of experience in talking about concepts of the used computer systems.

1.2 Active Help Systems

There are different system aids which can be considered as very simple active help systems (e.g. canned error messages). The systems which we envision do not only respond to errors but notice -- based on a model of the task and a model of the individual user -- **suboptimal actions** of the user which serve as a basis for individual help.

The following example (in the context of working with the UNIX file system) should illustrate our notion of an active help system: a user working in two different directories of the hierarchical file system may be observed typing several times a sequence of commands like:

```
cd /usr/src/ucb/lisp/lisplib
...
cd /users/andi/lisp/help/version3
...
```

to switch back and forth between two directories. An active help system might offer the

advice to use the commands *pushd* and *popd* which maintain a stack of directories and allow to exchange the current and previous directory.

It is not always clear which solution is suboptimal and should trigger an activity of the help system. Other solutions (e.g. definition of aliases for the two *cd* commands) might be equally straightforward. A **metric** is necessary to judge how adequate an action of the user is. Except for simple problem domains (e.g. a game where an optimal metric can possibly be defined (Burton, Brown 76)), optimal behavior cannot be uniquely defined. Our actual implementations (see section 3) of help systems are constructed for an editing system and the metric chosen is the closeness between the system's primitive operations and the concepts of the task domain (which is often related to the number of keystrokes). This is in many cases a questionable metric and in our future work we will represent more appropriate metrics, e.g. like defining the right relation between cognitive and physical effort.

If a user and a help system rely in their understanding on a very different metric the same difficulty like with human help occurs: **the help system "forces" the user to do something which he does not want to do**. A possible solution to this problem might be to make the metric visible and to allow the user to change it; but we must be aware that this increases the control of the user as well as the complexity of the system and it will be of little use if we do not find adequate communication structures for it.

2. Knowledge-based Human-Computer Communication

Knowledge-based systems are one promising approach to equip machines with some human communication capabilities. Based on an analysis of human communication processes we have developed the model shown in Figure 2-1.

The system architecture in Figure 2-1 contains two major improvements compared to traditional approaches:

- the **explicit** communication channel is widened (e.g. we use windows, menus, pointing devices etc. in our systems; see Figures 3-2 and 3-3)
- information can be exchanged over the **implicit** communication channel.

The four domains of knowledge shown in Figure 2-1 have the following relevance:

1. **knowledge of the problem domain**: research in Artificial Intelligence has shown that

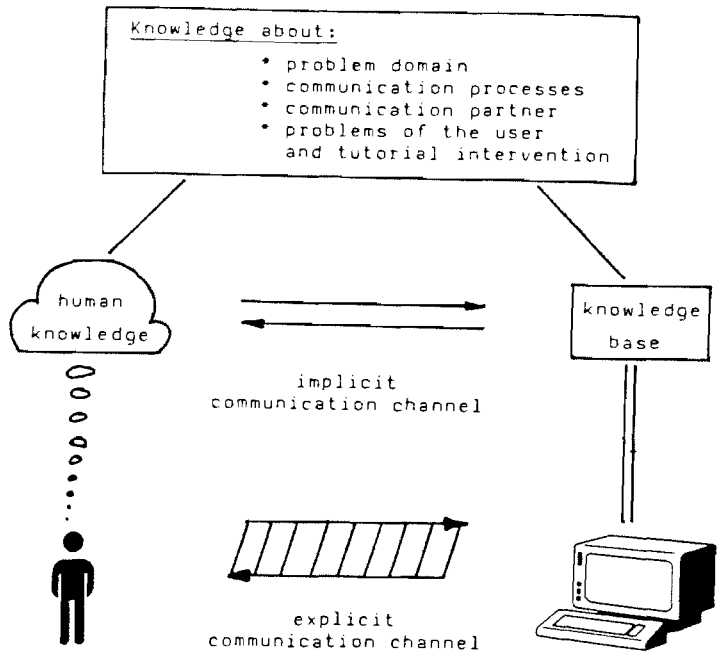


Figure 2-1: Architecture for knowledge-based Human-Computer Communication

intelligent behavior builds upon large amounts of knowledge about specific domains (which manifests itself in the current research effort surrounding expert systems).

Knowledge about the task domain imposes constraints on the number of possible actions. A model of the task domain describes reasonable goals and operations. In UNIX if a user needs more disk space it is in general not an adequate help to advise him to use the command *"rm *"*[2] (Wilensky et al. 84).

The user's goals and intentions can be inferred in situations where we understand the correspondance between the system's primitive operations and the concepts of the task domain. If the user of an editor repeatedly deletes characters up to the beginning of a word this can be recognized as the higher level concept *delete-beginning-of-word.* In ACTIVIST (see section 3.2) these concepts are modeled as plans. This mapping should be bidirectional. Given a problem of the task domain, a help system must be able to indicate how it can be solved using the primitive operations.

2. knowledge about communication processes: the information structures which control the communication should be made explicit, so the user can manipulate them.

[2] The command will delete all files in the directory

3. knowledge about the communication partner: the user of a system does not exist; there are many different kinds of users and the requirements of an individual user grows with experience. Most existing help systems do not take an individual user's special situation or his level of expertise into account. To pay attention to individual differences the following knowledge structures have to be represented:

* the user's conceptual understanding of a system (e.g. in an editor, text may be represented as a sequence of characters separated by linefeeds which implies that a linefeed can be inserted and deleted like any other character).
* the user's individual set of tasks for which he uses the system (a text editor may be used for such different tasks as writing books or preparing command scripts for an operating system).
* the user's way of accomplishing domain specific tasks (e.g. does he take full advantage of the systems functionality?).
* the pieces of advice given and whether the user remembered and accepted them.
* the situations in which the user asked for help.

One main task of an active help system is to monitor the user's behaviour and reason about his goals. Sources for this information are: the user's actions including illegal operations. This is based on the following hypotheses (Norman 82): *"a user does not make arbitrary errors; all operations are iterations towards a goal."* Classification of users with the help of stereotypes can be used to make assumptions about the expected behaviour of the user (Rich 79).

4. knowledge about the most common problems which users have in using a system and about tutorial invention: this kind of knowledge is required if someone wants to be a good coach or teacher and not only an expert; an user support system should know when to interrupt a user.

Help systems must incorporate tutorial strategies which are based on pedagogical theories, exploiting the knowledge contained in the model of the user. Strategies embodied in our systems are (Fischer 81):

* **take the initiative** when weaknesses of the user become obvious. Not every recognized suboptimal action should lead to a message.
* **be non-intrusive**. Only frequent suboptimal behaviour without the user being aware of it should trigger an action of the system.
* **give additional information** which was not explicitly asked for but which is likely to be needed in the near future.
* assist the user in the **stepwise extension** of his view of the system. Be sure that basic concepts are well understood. Don't introduce too many new features at once.

3. Prototypical Implementations

In this section a passive and an active help system for the editor BISY (Bauer 84) are described which we have developed and implemented. BISY is an EMACS-like, screen oriented editor developed in our research group. BISY was chosen for the following two reasons: editing is a task domain which is complex enough but well understood and BISY is integrated in our working environment (where we have a wide variety of tools at our disposal: LISP, OBJTALK, a window system etc.). Therefore we had access to the information structures needed to add a help system as an additional feature.

The current implementation of the two help systems can only deal with *cursor movement* and *deletion* tasks. In this domain BISY offers a rich set of operators. In addition to character oriented commands there are higher level operations for words, lines, paragraphs and lists. The systems' level of understanding is limited to these concepts. Concepts of subject domains in which editors are used (e.g. *address* in a letter) are not handled in the current implementation, but are subjects of future research work.

3.1 PASSIVIST: An Example for a Passive Help System

PASSIVIST (Lemke 84) is a natural language based help system for BISY. The first step in the design of this system was to get an impression of the real needs of the user. In several informal experiments a human expert simulated the help system in editing sessions with users of different expertise. The results indicated a fairly diverse set of problems ranging from finding keys on the keyboard up to complex formatting tasks.

PASSIVIST provides help to requests like (translated into English):

- *How can I get to the end of the line?*
- *I want to delete the next word.*

PASSIVIST uses a help strategy in which each step of the solution is presented and explained to the user who then executes this step and sees immediately the resulting effects. Help is currently given as text generated from sentence patterns according to the goal structure of the problem solving process. We are planning to augment this by displaying key sequences and subgoals graphically.

3.1.1 A sample session with PASSIVIST

Assume the user wants to delete a line. He hits the help key, a help window pops up and he is prompted for a question.[3]

Question: How can I delete the next line?

Recognized words: delete next line
Ignored words: How, can, I, the ?

The system indicates what it has understood. This is a common technique found in human discourses. It allows to correct misconceptions and to build a model of the understanding of the partner. The current implementation of PASSIVIST covers only a limited subset of domain-specific natural language utterances, but it recognizes far more complicated descriptions than a simple keyword analysis can do.

The system analyzes the sentence, identifies the user's problem and computes a solution by using primitives of the editor. The system first gives a description in natural language (using more or less canned text) and then tells the user the actions which he has to take:

First, move the cursor to the desired line.

Use Command: cursor-down Use Key: down-arrow

The user hits the mentioned key and sees the cursor moving down one line while staying in the same column.

The next step of the solution is rather complicated and is therefore explained by the system. The help system has knowledge of certain stereotypical problems and knows where it should give an explanation.

There is no primitive operation to delete a line as a whole. First you have to make it empty and then delete the left or the right line delimiter.

In order to make a line empty, move to its beginning and then delete the rest of the line.

Use Command: set-cursor-to-beginning-of-line Use Key: ^A

When the user hits ^A the system explains the next step which is to remove the rest of the line.

Use Command: rubout-line-right Use Key: ^K

---

[3] the user input is underlined; system output is in italics.

The user hits ^K and the line becomes empty. The last step is to remove one of the line delimiters.

> *The command* **rubout-line-right** *with the cursor at the end of the line merges the current with the following line. This is used to delete the empty current line.*
>
> *Use Command: rubout-line-right Use Key: ^K*

3.1.2 Implementation

PASSIVIST is implemented in OPS5 (Forgy 81). OPS5 is an interpreter for production systems which has been used in building expert systems.

Flexible parsing using OPS5 is achieved by a rule-based bottom-up method. The consistent structure of the system as a set of productions and a common working memory allows the use of the same knowledge in several stages of the solution process. For example, knowledge about the state of the editor is not only used to select a possible solution for the user's problem but also to aid to disambiguate the user's utterance. In the phrase *the last line* with the cursor being at the beginning of the editing buffer it is clear that the user means the last line of the buffer (and not the previous one).

Both the model of the user and the model of the editor state are represented as a set of clauses in the working memory of the production system. In the following example the system's model of the editor state describes a situation where the cursor is at the beginning of the first word in the last line of the edit buffer.

```
(Cursor  ^Word:                 at-beginning
         ^BeginningOfLine:      t
         ^EndOfLine:            nil
         ^InFirstLine:          nil
         ^InLastLine:           t
         ^BeginningOfBuffer:    nil
         ^EndOfBuffer:          nil)
```

Figure 3-1: A model of the editor state

The following rule (an English-like description of the corresponding OPS5 rule) represents the systems knowledge about deleting the end of a line:

```
IF   the goal is to delete a string
        AND  the string is the end of the current line
        AND  the cursor is not at the end of the current line

THEN remove the goal from the working memory
        AND  create a new goal to propose the command "rubout-line-right"
```

3.2 ACTIVIST: An Example for an Active Help System

ACTIVIST (Schwab 84) is an active help system that pays attention to suboptimal user behaviour. It is implemented in FranzLisp and in the object-oriented knowledge representation language OBJTALK (Laubsch, Rathke 83).

The help system can deal with two different kinds of suboptimal behaviour:

1. the user does not know a complex command and uses suboptimal commands to reach a goal (e.g. he deletes a string character by character instead of word by word).

2. the user knows the complex command but does not use the minimal key sequence to issue the command (e.g. he types the command name instead of hitting the corresponding function key[4]).

Like a human observer the help system has four main tasks:

* to **recognize** what the user is doing or wants to do.
* to **evaluate** how the user tries to achieve his goal.
* to **construct a model of the user** based on the results of the evaluation task.
* to decide (dependent on the information in the model) **when to interrupt** and in **which way** (tutorial invention).

In ACTIVIST the recognition and evaluation task is delegated to 20 different **plan specialists**. Each one recognizes and evaluates one plan of the problem domain. Such plans are for example *"deletion of the next word", "positioning to the end of line"*, etc..

A plan specialist consists of:

1. an transition network (TN), which matches all the different ways to achieve the plan using the functionality of the editor. Each TN in the system is independant. The results of a match are the **used editor commands** and the **used keys** to trigger these commands.

2. an expert which knows the optimal plan including the **best editor commands**

[4] In BISY a command can be bound to a function key

and the **minimal key sequence** for these commands.

3.2.1 Recognition Task

All TNs are active and try to recognize their plans simultaneously. An editor command issued by the user causes a state transition in every TN. The input for the TNs is the command and the buffer state after execution of the command. The buffer state is defined by the position of the cursor with respect to words, lines and the buffer as a whole (see Figure 3-1). The detailed structure of our TNs are described in Fischer/Lemke/Schwab (Fischer, Lemke, Schwab 84).

3.2.2 Evaluation Task

Whenever a plan is recognized by the associated TN the result of the recognition process is compared with the stored "best" solution for this plan. Other commands than the proposed ones are considered as a bad solution if the user uses the combination of several concepts whereas the task could have been achieved by using a single concept (e.g. deleting a line by a series of character deletion commands; see also the example with "pushd" and "popd" in section 1.2). In case he uses the recommended commands his action will only be evaluated as good if he also uses the minimal key sequence.

3.2.3 Modeling the User in ACTIVIST

For each plan there is a knowledge structure which models the user and which contains the following slots (see also Figure 3-2, in which the indicated abbreviations are used):

- *done* (abbreviation: D) is the number how often the plan was done by the user (in any way).
- *good-done* (abbreviation: G) shows how often the plan was done with the optimal commands and the minimal key sequence.
- *wrong-command-used* (abbreviation: COM; value before the arrow) shows how often a wrong command was used when the use of the proposed command would have reduced the number of pressed keys. *keys1* counts the unnecessary keys (value after the arrow).
- *wrong-keys-used* (abbreviation: KEYS; value before the arrow) shows how often the proposed commands were used but not with the minimal key sequence. *Keys2* counts the unnecessary keys (value after the arrow).
- *messages-to-user* shows how often a message concerning this plan was given to the user.

3.2.4 Help Strategy

The help strategy of ACTIVIST is variable. All limits (which determine when the system talks to the user) are parameterized which allows to experiment with different tutorial strategies.

The output of help messages is based on the following global strategy:

* the time between two messages shall be at least *min-message-delay* seconds to prevent information overflow. For a beginner it is very frustrating to be continuously criticized.
* a message concerning a special plan is given only immediately after the plan was done wrong - not at a later time.
* the message concerning the same error will only be given *max-messages* times to be non-intrusive and to accept that the user wants to do something in his way.

The help activity is directed towards essential rather than sporadical errors. Therefore criteria to give a message to the user concerning a special plan are:

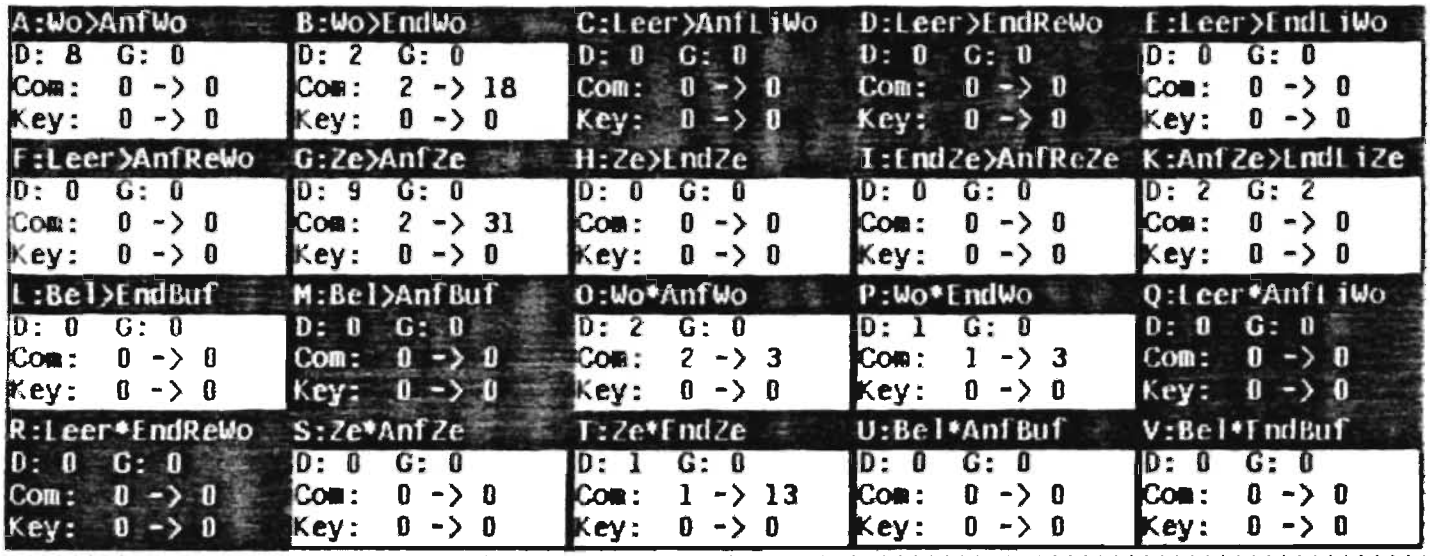
* a plan was done at least *wrong-command-limit* times with the wrong commands and the number of unnecessary keys is greater than *keys1-limit* or
* the proposed commands are triggered at least *wrong-command-limit* in a suboptimal way and the number of unnecessary keys is greater than *keys2-limit*.

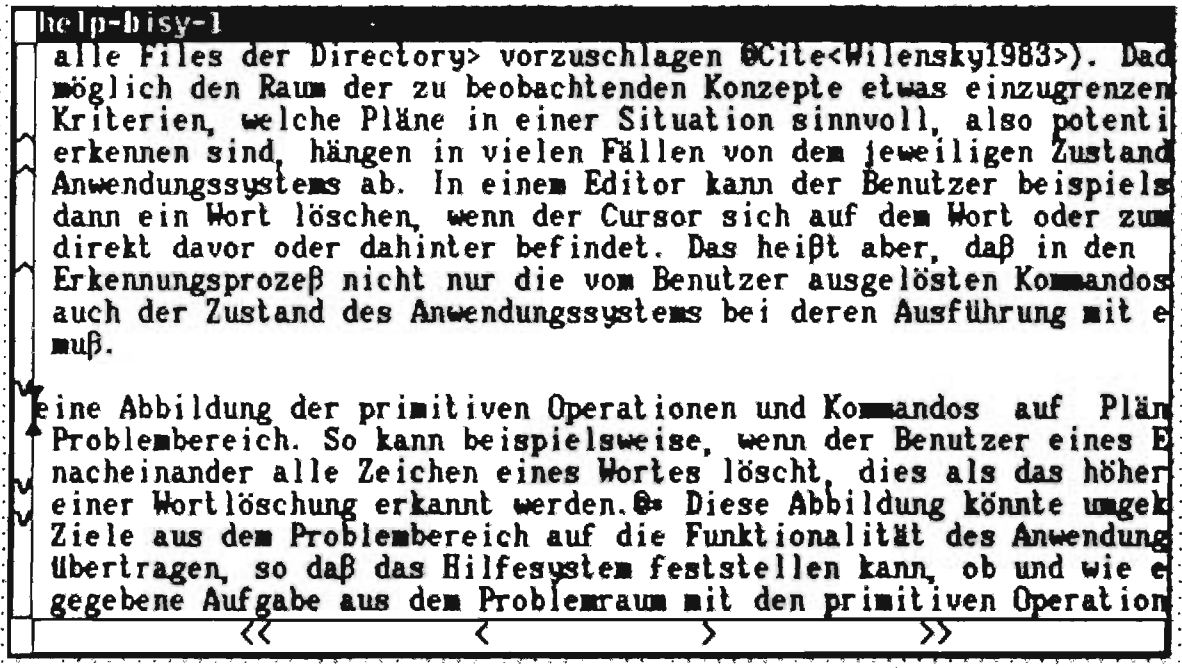
A plan which was executed *good-used-limit* times in the optimal way will not be watched any more. The help system assumes that the user is familiar with this feature.

3.2.5 A sample session with ACTIVIST

Figure 3-2 shows a typical situation during a session with the editor. In the upper part of the screen the user model is shown that ACTIVIST has built up. For each plan there is a pane which shows the performance of a specific user concerning this plan. Panes with black background indicate that the corresponding plan is not watched by the active help system (the abbreviations have the meaning described in section 3.2.3).

In the dialog window under the editor window a help message given to the user can be seen. The user has executed the command *set-cursor-to-beginning-of-line* by typing in the command name. ACTIVIST gives the hint, that this command is also bound to the key ^A.





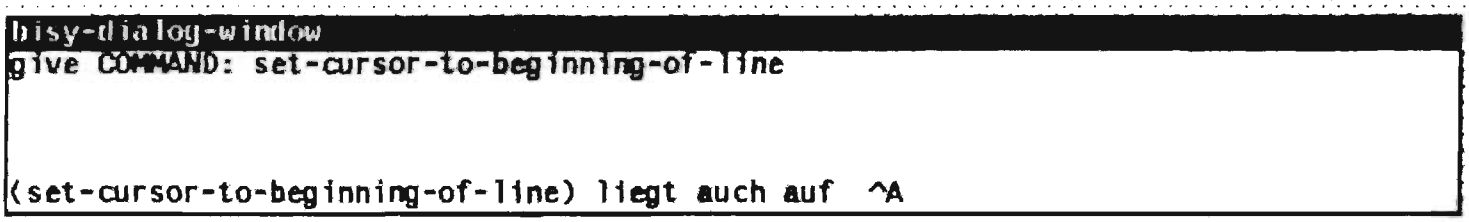


Figure 3-2: The user model in ACTIVIST

In Figure 3-3 the recognition task for the plan **"delete the left part of the current word"** is monitored in a window. This window shows the user model, the proposed command (with the optimal keys) for this plan and the state of the plan recognition.

The user has executed three times the command *rubout-character-left* with the DEL-key. The cursor is now standing between the first and the second character of the word. The recognition is still going on. If the user invokes once again *rubout-character-left* the plan will be recognized. Then the evalution will begin: The used commands will be compared with the optimal commands for this plan and ACTIVIST will recognize the first kind of suboptimal behaviour (as described above).

4. Future Research

In realistic applications the number of possible user intentions and actions which can be watched simultaneously will quickly reach the limit of the available computational power. Similary to a human tutor the help system is unable to watch all plans simultaneously; therefore it is necessary to concentrate on some plans. Relevant criteria can be based on the following:

- A plan which was executed in the optimal way several times by the user need not to be observed any more.
- Very complex plans are not relevant for a novice user.
- The user can indicate his insecurity in a special domain by questions to the passive help system; these plans can then be observed in detail.
- The user can decide which plans shall be watched.

Currently our systems work primarily bottom-up, based on data-driven observation of the user behavior. Model-driven predictions (based on a diagnostic model of the most common problems which users have) should be integrated into our system and they could be used to focus the attention of our help systems.
Another way (compared to providing good help systems) may be: to make systems so transparent and so suggestive that there is no need for help at all (e.g. the use of the mouse in our systems has greatly reduced the complexity of many system components).

The current approach in building user support systems can be characterized that these systems are still constructed as addons to existing systems. Our long ranging vision of how computer systems should be developed is not to write code, but to represent knowledge domains from which we can generate arbitrary projections being used as code, as explanations, as documentation or as help.

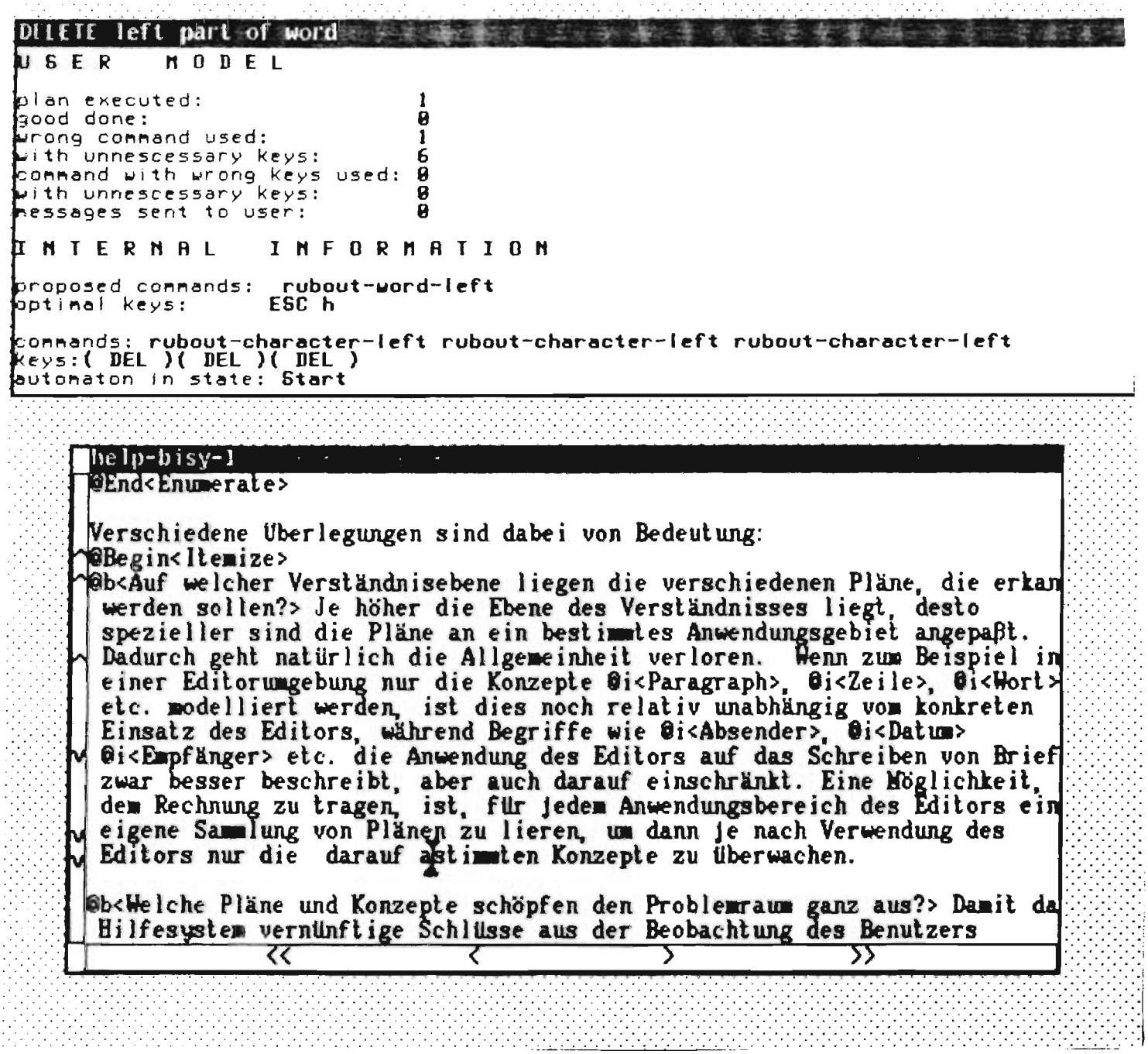


Figure 3-3: Monitoring the recognition task

References

(Bauer 84)
J. Bauer: *"BISY. A Window-Based Screen-Oriented Editor, Embedded in ObjTalk and FranzLisp"*. Institutsbericht, Projekt INFORM, Institut für Informatik, Universität Stuttgart, January, 1984.

(Burton, Brown 76)
R.R. Burton, J.S. Brown: *"A tutoring and student modeling paradigm for gaming environments"*. In *Proceedings for the Symposion on Computer Science and Education*. Anaheim, Ca., February, 1976.

(Fischer 81)
G. Fischer: *"Computational Models of Skill Acquisition Processes"*. In R. Lewis and D. Tagg (editors), *Computers in Education*, pp 477-481. 3rd World Conference on Computers and Education, Lausanne, Switzerland, July, 1981.

(Fischer 84)
G. Fischer: *"Formen und Funktionen von Modellen in der Mensch-Computer Kommunikation"*. In M.J. Tauber (editor), *Psychologie der Computernutzung*. Wien - München, 1984. Schriftenreihe der Österreichischen Computergesellschaft.

(Fischer, Lemke, Schwab 84)
G. Fischer, A. Lemke, T. Schwab: *"Active Help Systems"*. In T.Green, M.Tauber, G.van der Veer (editors), *Proceedings of Second European Conference on Cognitive Ergonomics - Mind and Computers, Gmunden, Austria*. Springer Verlag, Heidelberg - Berlin - New York, September, 1984.

(Forgy 81)
C.L. Forgy: *"OPS5 User's Manual"*. Technical Report CS-81-135, CMU, 1981.

(Laubsch, Rathke 83)
J. Laubsch, C. Rathke: *"OBJTALK: Eine Erweiterung von LISP zum objektorientierten Programmieren"*. In H.Stoyan, H.Wedekind (editors), *Objektorientierte Software- und Hardwarearchitekturen*, pp 60-75. Stuttgart, 1983.

(Lemke 84)
A. Lemke: *"PASSIVIST: Ein passives, natürlichsprachliches Hilfesystem für den bildschirmorientierten Editor BISY"*. Diplomarbeit Nr. 293, Institut für Informatik, Universität Stuttgart, 1984.

(Norman 82)
D. Norman: *"Some Observations on Mental Models"*. In D. Gentner, A. Stevens (editors), *Mental Models*. Hillsdale, N.J., 1982.

(Rich 79)
E. Rich: *"Building and Exploiting User Models"*. Ph.D. Thesis, Carnegie-Mellon University, 1979.

(Schwab 84)
Th. Schwab: *"ACTIVIST: Ein aktives Hilfesystem für den bildschirmorientierten Editor BISY"*. Diplomarbeit Nr. 232, Institut für Informatik, Universität Stuttgart, 1984.

(Wilensky et al. 84)
R. Wilensky, Y.Arens, D. Chin: *"Talking to UNIX in English: An Overview of UC"*. *Communications of the ACM* 27(6), pp 574-593, June, 1984.