Institut für Informatik · Universität Stuttgart · Herdweg 51 · D 7000 Stuttgart 1

INFORM
Institut für Informatik
Universität Stuttgart
Herdweg 51
D 7000 Stuttgart 1
Telefon (07 11) 2078-

June 1984

GERHARD FISCHER, ANDREAS LEMKE, THOMAS SCHWAB

A C T I V E H E L P S Y S T E M S

To appear in:

Green, Tauber, v.d.Veer (editors): Cognitive Ergonomics, Mind and Computer. Proceedings of the Second European Conference on Cognitive Ergonomics, Mind and Computer, September 1984, Gmunden, Austria; Springer Verlag, Heidelberg-Berlin-New York

INFORM: Forschungsgruppe für Wissensbasierte Systeme und Mensch-Computer-Kommunikation.
(Gefördert vom Bundesministerium für Forschung und Technologie)

Table of Contents

List of Figures

Active 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

Good on-line help systems are of crucial importance for the computer systems of the future. An increased functionality (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 computer systems are used. Passive help systems (which require that the user requests help explicitly from the system) are of little use if the user does not know the existence of a system feature. Active help systems should guide and advise an user similar to a knowledgeable colleague or assistant.

1. Introduction

The purpose of computer-based man-machine systems is to direct the computational power of the digital computer to the use and convenience of man. There has been great progress towards this goal and the dramatic price reduction in hardware has led to totally new possibilities. But other aspects have not kept pace with this progress, especially how easy it is (not only for the expert but also for the novice and the occasional user) to take advantage of the available computational power to use the computer (for a purpose chosen by him/herself[1]).

Most computer users feel that computer systems are unfriendly, not cooperative and that it takes too much time and too much effort to get something done. They feel that they are dependent on specialists, they notice that software is not soft (i.e. the behavior of a system cannot be changed without a major reprogramming of it) and the casual user finds himself in a situation like in instrument flying: he needs lessons (relearning) after he did not use a system for a long time.

---

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

Our goal is to create symbiotic, knowledge-based computer support systems which handle all of their owner's information-related needs (these needs will be quite different for different groups of users).

Our system building efforts can be characterized with the following metaphor: the user of an interactive system is like a traveler on a modern highway system with (maybe ill-defined) goals, methods, heuristics and decision points; the user support systems serve as tour guides who provide assistance if requested and who point out interesting places (based on knowledge about the traveler and the country-side).

2. User Support Systems

The user-support systems which we envision must be build as knowledge-based systems. First the architecture of knowledge-based systems will be described and then several kinds of user support systems will be briefly characterized.

2.1 Knowledge-based Human-Computer Communication (HCC)

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:

1. the explicit communication channel is widened. Our interfaces use windows with associated menus, pointing devices, color and iconic representations; the screen is used as a design space which can be manipulated directly (see Figure 4-1).

2. information can be exchanged over the implicit communication channel. The shared knowledge base eliminates the necessity that all information has to be exchanged explicitly.

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

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

3. knowledge about the communication partner: the user of a system does

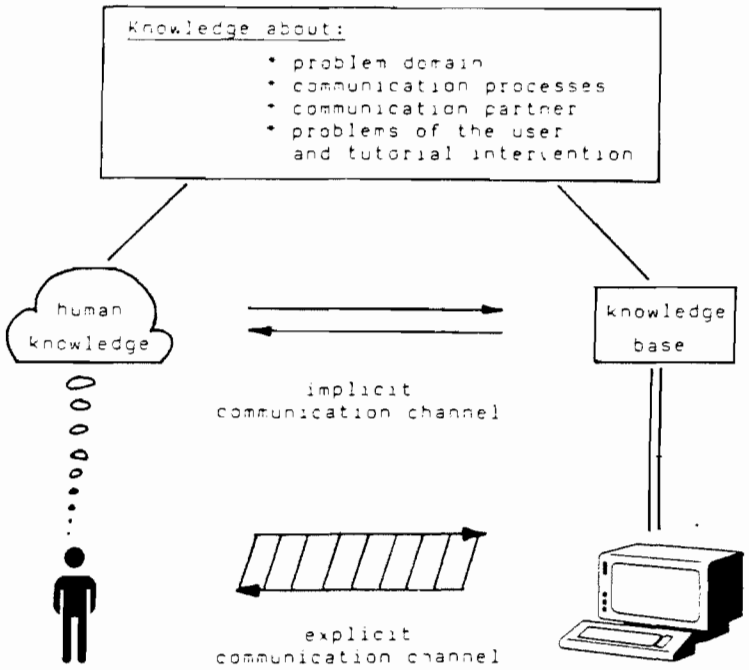

Figure 2-1: Architecture for knowledge-based HCC

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.

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.

Knowledge-based communication allows to replace canned information structures (like they were used in traditional CAI) with dynamically generated information based on the contents of the underlying knowledge base.

2.2 Tutorial Systems

Tutorial (or instructional) system should assist a learner; the material is presented based on psychological and pedagogical considerations. The learner (not familiar with the concepts and with the structure of the system) is unable to articulate his goals. The interaction is determined to a large extent by the system and therefore the information can be structured in advance. A (not too sophisticated) example of this sort of system is the LEARN system embedded in UNIX. In our work we try to enhance tutorial systems with modern HCC techniques, e.g. the dynamic generation of visual representations (see Figure 4-1; (Nieper 83)).

2.3 Explanation Systems

Information processing using the implicit communication channel (see Figure 2-1) implies that a system infers information, makes assumptions and draws complex conclusions. Explanation systems should provide insight into complex computations. The user has communicated a goal to the system and he wants to have feedback what the system has done. In a system which makes extensive use of defaults (e.g. the document preparation system SCRIBE) it is important that the user can ask the system to explain which default assumptions exist and how they contribute to a certain outcome.

2.4 Documentation Systems

We consider a documentation system as the kernel of a software production environment. It contains all the available knowledge about the system combined with a set of tools useful to acquire, store, maintain and using this knowledge. It serves as the communication medium between clients, designers and users throughout the entire design process. A valid and consistent documentation during the programming process itself is of special importance in incremental design processes to provide answers to the questions: what has been done, what is the next thing to do, how does the current implementation compare with the specifications, etc.. DOXY (Lemke, Schwab 83) is a prototypical implementation of our general approach (Fischer, Schneider 84) to build computer-supported program documentation systems.

2.5 Help Systems

Help systems should assist the user in specific situations. The user knows his goals but he cannot communicate them to the system (passive help systems) or he does not know that the system offers better ways to achieve the task (active help systems). Contrary to tutorial systems help systems cannot be structured in advance but must "understand" the specific contexts in which the user asks or needs help. The interaction is much more initiated by the user (in passive help system) or mixed-initiative (in active help systems) than in tutorial systems.

In the remaining part of the paper only help systems will be discussed and our system building efforts in constructing them will be described.

3. Help Systems

An increased functionality (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 computer systems are used. Figure 3-1 is based on our empirical investigations through careful observations (e.g. persons using systems like UNIX, EMACS etc. in our environment) and describes different levels of system usage which is characteristic for many complex systems. The different domains correspond to the following:

D1: the subset of concepts (and their associated commands) which the user knows and uses without any problems.

D2: 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. A passive help system can help him to take advantage of the commands in D2.

D3: the mental model (Fischer 83) of the user, i.e. the set of concepts which he thinks exist in the system.

D4: D4 represents the actual system. Passive help systems are of little use for the subset of D4 which is not contained in D3, because the user does not know about the existence of a system feature. Active help systems 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 D4.

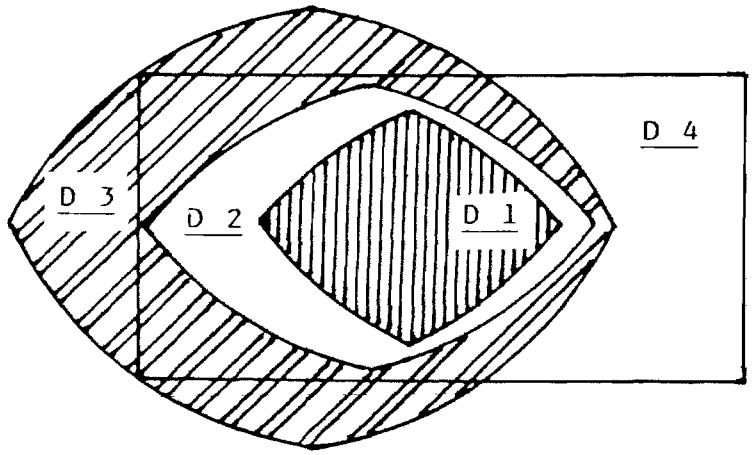

Figure 3-1: Different levels of system usage

Based on these findings it is obvious that a good help system must have a passive and an active component. Nearly all existing help systems are passive help systems where the user takes the active part and the system responds to his requests. Possible interaction techniques are menus, keywords and natural language. Some help systems are built as browsers in a large network of

descriptive information where the user moves around and searches answers to his problems.

3.1 Passive Help Systems

3.1.1 Keyword Based Help Systems

Assuming a user knows the name of a command but not its details, keyword based help systems are a quick, easy to implement and sufficiently reliable choice. Many of these help systems still fail when the user does not know the exact name. The success rate can be improved by adding synonym lists and pattern matching capabilities which make is possible to express problems in different ways.

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. Therefore in many situations too much information is provided.

3.1.2 Natural Language Based Help Systems

Natural language provides a wider communication channel between the user and the system than menu selection or keyword based interaction. 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:

1. the user has multiple ways to formulate a problem. Natural language provides much more flexibility than the best synonym lists.
2. with natural language failures are soft. Most of the user's utterances give at least a hint to where the problem lies.
3. misconceptions of the user can be identified from his utterance. This gives an important clue for building a user model.
4. 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.

Current natural language systems implement only a restricted subset of natural language. It is an unresolved problem how to describe this subset to the user. It may well be that it is easier for the user to learn a formal way of expressing his needs than learning the restrictions in the natural language interface.

3.2 Active Help Systems

There are many different systems aids which can be considered as active help systems. Canned error messages occurring every time the user does something wrong are the simplest form of an active help system. The active help systems which we envision do not only respond to errors but notice -- based on a model of the user and the task -- suboptimal actions of the user. In an operating system the user may issue the sequence of commands:

```
DELETE PARSE.PAS.1
DELETE PARSE.PAS.2
DELETE PARSE.PAS.3
```

leaving the file PARSE.PAS.4. An active help system (Finin 83) might offer the advice to use the command "PURGE PARSE.PAS" which deletes all the old versions of a file.

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 5) of help systems are constructed for an editing system and the metric chosen is the number of keystrokes. There is no doubt that this is a very crude and questionable metric. 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.

4. Requirements for Help Systems

This section describes issues which are relevant for both types of systems described in the next section.

4.1 The User as an Information Processing System

Design guidelines must be based on a thorough understanding of the strength and weaknesses of the human information processing system. In our work we have paid special attention to the following issues:

1. the **power of the visual system** as a very efficient chunking method should be utilized. In Figure 4-1 (Nieper 83) we show a visual representation of a LISP data structure which gets generated automatically from the symbolic representation. Given the following lists

   l1 <== ((eins 1) (zwei 2))
   l2 <== ((drei 3) (vier 4))

   the two operations APPEND and NCONC generate on the surface the same result

   (append l1 l2) => ((eins 1) (zwei 2) (drei 3) (vier 4))
   (nconc l1 l2) => ((eins 1) (zwei 2) (drei 3) (vier 4))

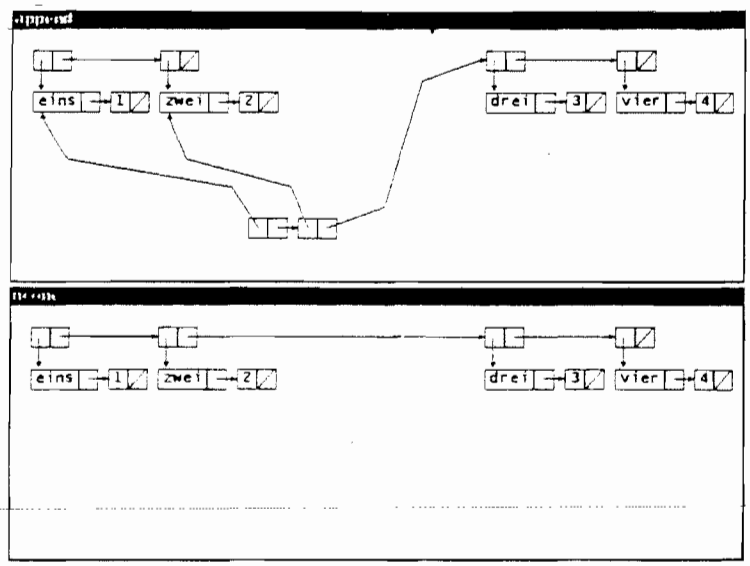

Figure 4-1: Visualization techniques to assist in LISP programming

The visualization in Figure 4-1 shows what really happened: APPEND is a nondestructive operation but uses up two more memory cells whereas NCONC is a destructive operation.

2. **recognition memory** provides different access mechanisms compared to recall memory; therefore menu-based system and property sheets (like

in the STAR interface) may be helpful for the novice and the casual user.

3. the **scarce resource** in human-computer systems is not information but human attention; this implies that techniques for intelligent summarizing, prefolding of information etc. have to be developed based on a model of the user.

4. our **short term memory is limited**, i.e. there is a very finite number of things which we can keep in mind at one point of time. Help systems describe and illustrate features about certain aspects of a system; in many cases it is necessary to see both information structures at the same time which requires window systems.

5. our systems must be **consistent and uniform** for the user and consistency must be based on an adequate model of how a system will be used. Help systems are part of the system and therefore they should have the same interface as the other parts of the system. Otherwise the well-known situation will occur that we need *a help systems to use the help system.*

4.2 Modeling the User

Many existing help systems do not take an individual user's special situation or his level of expertise into account. The following knowledge structures have to be represented:

1. 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).

2. 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).

3. the user's way of accomplishing domain specific tasks (e.g. does he take full advantage of the systems functionality?).

4. the pieces of advice given and whether the user remembered and accepted them

5. 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).

4.3 Modeling the Task Domain

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 83).

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 5.2) 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 (e.g. in the domain of text editing the help system finds the sequence of operators *delete-region insert-region* for the user's goal of moving a piece of text).

4.4 Help Strategies

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

1. **Take the initiative** when weaknesses of the user become obvious. Not every recognized suboptimal action should lead to a message. Only frequent suboptimal behaviour without the user being aware of it should trigger an action of the system.

2. **Be non-intrusive**. If the user does not accept our suggestions, let him do the task in his way.

3. **Give additional information** which was not explicitly asked for but which is likely to be needed in the near future.

4. 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 (Fischer 81).

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

5. Prototypical Implementations

In this section a passive and an active help system for the editor BISY are described which we have developed and implemented. BISY (Bauer 84) is an EMACS-like, screen oriented editor, also developed in our research group.

BISY was chosen for the following reasons:

* Editing is a task domain which is complex enough but well understood.
* 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 it was easy to add a help system as an additional feature.
* Editing systems are an often used tool in our work and therefore it is easy to get feedback about the usefulness of our help systems.

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.

5.1 PASSIVIST: An Example for a Passive Help System

PASSIVIST (Lemke 84) is a natural language based help system for the editor 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.

A passive help system has the choice between the following help modes:

* **Explain:** The system gives hints (e.g. classifies the problem, reminds the user of certain facts) to assist him to solve the problem himself.
* **Solve:** The system provides and explains a complete solution.

* **Demonstrate:** The solution of the user's problem will not only be explained but each step will be demonstrated.

* **Execute:** Sometimes a solution may be too complicated to present to a beginning user. In these cases the system simply does what the user wants to do.

PASSIVIST uses a combination of the first and the third 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 (see section 4.1). Help is currently given only as text, generated from slightly abstracted canned information structures. We are planning to augment this by displaying key sequences and subgoals graphically.

5.1.1 A Sample Request to 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. In this case the user can see that the system's understanding is limited.

The system analyzes the sentence, identifies the user's problem and computes a solution by using primitives of the editor which are available to the user. 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.

Command: cursor-down 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

---

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

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.

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

^A means hold down the **CTRL** *key, hit the* **A** *key and release the* **CTRL** *key.*

It is assumed that the user is a beginner which has no experience with computer key boards. Therefore, for the first two times CTRL-keys are explained.

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

Command: rubout-line-right Key: ^K

^K means hold down the **CTRL** *key, hit the* **K** *key and release the* **CTRL** *key.*

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

Command: rubout-line-right Key: ^K

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.*

5.1.2 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.

The german word 'letzt' may mean the 'previous' as well as 'the last in a region'. In the phrase

die letzte zeile (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.

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; examples are

a) the system's model of the user (the number indicates the frequency how often the concept was explained to the user):

```
(User KnowsNotationOfKeySequences 2)
(User KnowsNotationOfCtrlKeys 1)
```

b) the system's model of the editor state:

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

To answer a question of the user the system has to do the following:

1. build a model of the editor state
2. read and scan the question
3. parse the question into an internal representation of the user's goals
4. compute solution
5. present and explain solution

The following rule represents the systems knowledge about deleting the end of a line. If there is a goal matching the condition part (the two clauses labeled *<goal>* and *<region>*) and the cursor is not at the end of a line (otherwise another rule triggers) the system proposes the command rubout-line-right.

```
(production DeleteEndOfLine
   {<goal>    (Goal delete <side>  ^Active: t)}
   {<region>  (Side <side>  ^Part: end  ^Object: line)}
              (Cursor  ^EndOfLine: nil)
 -->
   (remove <goal> <region>)
   (make Goal issueCommand rubout-line-right  ^Active: t))
```

Sometimes there are better solutions than those found by the system. If there is only a word or a single character between the cursor and the end of the line, other commands (*rubout-word-right, rubout-character-right*) can be used. The system needs a metric to choose an optimal solution (see section 3.2).

5.2 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 the object-oriented knowledge representation language OBJTALK (Laubsch, Rathke 83).

A user action is considered optimal if it is done with a **minimum number of keystrokes**. 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:

1. to **recognize** what the user is doing or wants to do.

2. to **evaluate** how the user tries to achieve his goal.

3. to **construct a model of the user** based on the results of the evaluation task.

4. 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 automaton, which matches all the different ways to achieve the plan using the functionality of the editor. Each automaton 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** and the **minimal key sequence** for these commands.

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

5.2.1 Recognition Task

All automata are active and try to recognize their plans simultaneously. An editor command issued by the user causes a state transition in every automaton. The input for the automata 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.

The automaton in Figure 5-1 can recognize the plan "delete the left part of the current word".

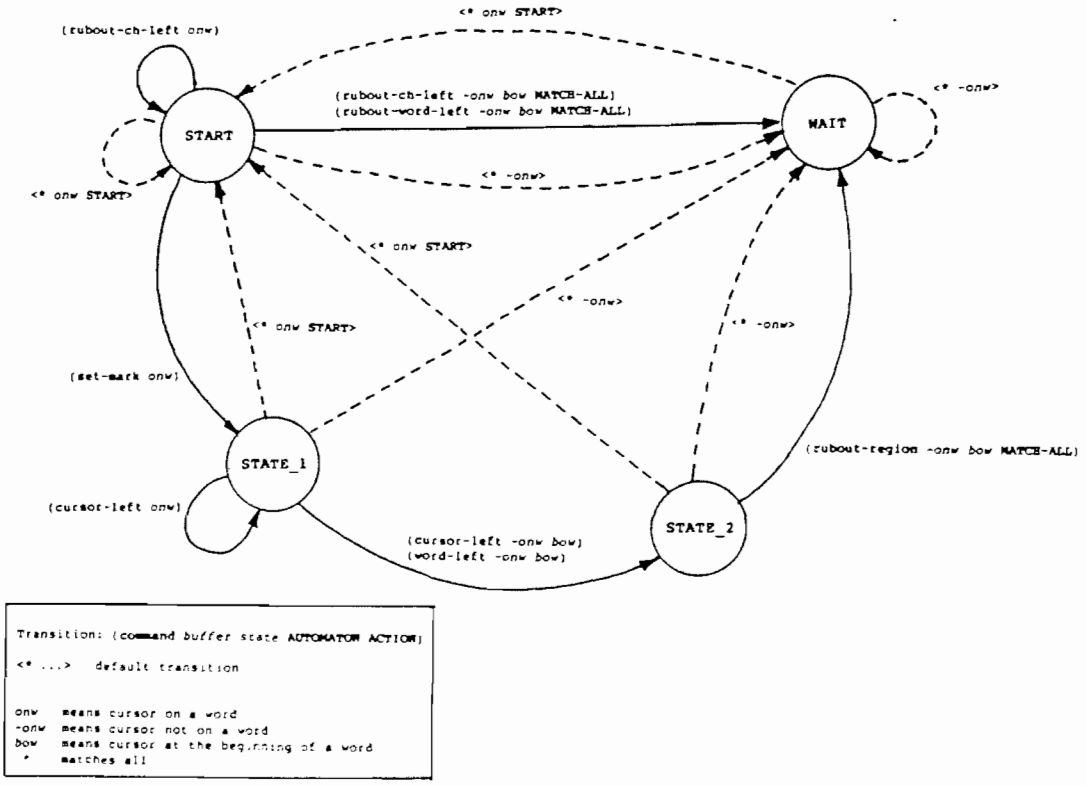

Figure 5-1: Automaton: rubout word left

Initially the automaton is in one of the states *START* and *WAIT*.

* *START:* the preconditions for the plan are satisfied. In this case the cursor must be in the middle of a word (i.e. the *onw* predicate is true).

* *WAIT*: the plan cannot be immediately executed. The cursor is not at a word.

The transition *WAIT --> START* initiates the *START-ACTION*. The internal memory

of used commands and keystrokes is initialized. While the user executes the plan the automaton follows the solid lines and records the used commands and keystrokes. The automaton action *MATCH-ALL* means that the plan is completely recognized and the evaluation task is initiated. For a command which is not part of the plan a default transition (dashed line) leads the automaton to one of the initial states (only depending on the buffer state).

5.2.2 Evaluation Task

Whenever a plan is recognized by the associated automaton 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 needs more keystrokes than for the stored solution (e.g. if a word consists exactly of one character the proposed command *rubout-word* is not better than *rubout-character*).

In case he uses the recommended commands his action will only be evaluated as good if he also uses the minimal key sequence.

5.2.3 Modeling the User in ACTIVIST

For each plan there is a knowledge structure which models the user and which contains the following slots:

* *plan-executed* is the number how often the plan was done by the user (in any way).
* *good-done* shows how often the plan was done with the optimal commands and the minimal key sequence.
* *wrong-command-used* 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.
* *wrong-keys-used* shows how often the proposed commands were used but not with the minimal key sequence. Here *keys2* counts the unnecessary keys.
* *messages-to-user* shows how often a message concerning this plan was given to the user.

5.2.4 Help Strategy

The help strategy of ACTIVIST is variable in the way that all limits are changeable 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 can be 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 shall be concentrated more on essential than sporadical errors. Therefore criteria to give a message to the user concerning a special plan are:

1. a plan was done at least *wrong-command-limit* times with the wrong commands and the number of unnecessary keys is greater than keysl-limit or

2. 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.

6. Future Research

Our general research goals to improve human-computer communication with knowledge-based systems have shown that user-support systems in general and help systems specifically are of crucial importance to the success of computer-based systems. Many interesting questions have come up and we will mention the most important ones.

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:

1. A plan which was executed in the optimal way several times by the user need not to be observed any more.

2. Very complex plans are not relevant for a novice user.

3. 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.

4. 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.

Help systems are currently 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 construct rich knowledge-structures from which we can generate arbitrary projections being used as code, as documentation or as help.

Instead of providing good help systems it may be more fruitful to try 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; see figure 4-1).

For help systems to be truly useful, the following conditions must hold: the amount to build them must be feasible (e.g. we need support for knowledge acquisition, consistency maintenance, etc.) and the help system itself must be easy to use.

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.

(Finin 83)
T.W. Finin: *"Providing Help and Advice in Task Oriented Systems"*. In *Proc. of the Eighth IJCAI*, pp 176-178. , 1983.

(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, Schneider 84)
G. Fischer, M. Schneider: *"Knowledge-based Communication Processes in Software Engineering"*. In *Proceedings of the 7th International Conference on Software Engineering*, pp 358-368. Orlando, Florida, March, 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.

(Lemke, Schwab 83)
A. Lemke, T. Schwab: *"DOXY: Computergestützte Dokumentationssysteme"*. Studienarbeit Nr. 338, Institut für Informatik, Universität Stuttgart, 1983.

(Nieper 83)
H. Nieper: *"KÄSTLE: Ein graphischer Editor für LISP-Datenstrukturen"*. Studienarbeit Nr. 347, Institut für Informatik, Universität Stuttgart, 1983.

(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, -Mellon University, 1979.

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

(Wilensky 83)
R. Wilensky: *"Planning and Understanding"*. Addison-Wesley, Reading, Massachusetts, 1983.