A SHELL FOR INTELLIGENT HELP SYSTEMS

Joost Breuker, Radboud Winkels, Jacobijn Sand berg
University of Amsterdam
Herengracht 196
Department of Social Science Informatics 1016 BS Amsterdam

ABSTRACT

The research reported here is part of a project aimed at the construction of an environment for building intelligent help systems. A help system supports the user in handling and mastering an information processing system. Core of this environment is a shell that contains all domain independent procedures and knowledge. A comprehensive help system not only answers questions of users, but also 'looks over their shoulders* and interrupts when appropriate. This means that a help system is equipped with a PERFORMANCE INTERPRETER, consisting of a PLAN RECOGNISER, a DIAGNOSER, and a QUESTION INTERPRETER. Part of this shell and focus of this paper is a generic COACH. In a help system a COACH has two functions: to assist the user with a current problem and to teach the user about the IPS. The proposed COACH consists of three layers: 1) A DIDACTIC GOAL GENERATOR which generates an overlay of domain concepts that may be taught, 2) STRATEGY PLANNER which constructs coaching strategies, and 3) TACTICS which are the terminal elements of strategies. They are the speech acts finally "uttered" by the COACH. In this paper these three layers are discussed in greater detail and are related to empirical research.

I INTRODUCTION

The research reported here is part of the EUROHELP project. *) This project is aimed at the construction of an environment for building intelligent help systems. A comprehensive help system supports the user both in an active and in passive way in handling and mastering a particular 'information processing system' (IPS, i.e. an interactive computer program). Core of this environment is a shell that contains all domain independent procedures and knowledge. The major task of a developer of a help system for some specific IPS will be to load the shell with a representation of the domain concepts (commands, syntax, methods of object reference, etc). This shell is under construction now; specifications have been written and a prototype help system (EUROHELP.P0) for Unix-Mail has been developed and tested (Breuker, in press). Implementation is in LOOPS on a Xerox 1186. In this paper we will focus on the construction of a generic COACH that is part of this shell, after a short description of what a full help system consists of.

•) The research is partially funded by the ESPRIT Program of the European community under contract P280. The project encompasses an effort of about 100 man-years over a 6 year period, of which 2 years have been spent now. Partners in the project are: CRI, DDC (Denmark), ICL, University of Leeds (U.K.), University of Amsterdam, Courseware Europe (The Netherlands).

II FUNCTIONS AND STRUCTURE OF HELP SYSTEMS

The function of a help system is to provide information about the use of some IPS when needed by any type of user. The need can be expressed by the user or be inferred by the help system. This means that a help system should have the role of a human coach, who looks over the shoulder of the user to interpret her performance, interrupts when things go wrong or when there is an opportunity to extend the repertoire of the user, and who is able to answer questions in the context of current use of the IPS. The latter is crucial, because many users -in particular novice users- are not aware of a problem, and if they are, they do not know how to describe it (Fischer et al., 1985). This is one important reason why question-answering help systems, which do not interpret user performance, have a very limited functionality (e.g. UC (Wilensky et al., 1984; AQUA (Quillici et al., 1986)). On line monitoring the performance of the user entails many conceptual and computational problems, but these are not qualitatively different from those in intelligent coaching in general (e.g. Sleeman & Brown, 1981; Anderson et al. 1985; Self, in press).

A. Architecture of the EUROHELP shell

Because user needs may either be identified by the system or by the user, EUROHELP consists of a QUESTION INTERPRETER and a PERFORMANCE INTERPRETER. To circumvent the problem of interpreting questions in natural language (Lehnert, 1978; Wilensky, et al., 1984) question frames are used. For each type of question (Lehnert, 1978; Hartley & Smith, in press) a text frame is presented for which the user supplies the objects.

The question interpretation problem may be reduced by relying on the linguistic competence of the user, the performance interpretation problem presents itself in its full glory. In normal coaching (training) the system presents a problem to the student. However, in IPS performance the user poses the problems (tasks) to herself. Finding out whether something goes wrong is highly dependent on identifying the intentions of the user. Therefore, the PERFORMANCE INTERPRETER contains a PLAN RECOGNISER and a PLANNER, which cooperate in such a way that the former works bottom-up and provides constraints to the latter in generating currently feasible plans. Plans may not only be wrong or impossible, they may be highly inefficient. In almost any editor, for example, a line can be deleted by a simple command rather than character by character.

The tracing of errors is simplified to some extent by the fact that many erroneous actions of the user are not executable by the IPS. However, the variety of executable errors is as complicated as in any 'natural' domain, because users may acquire all kinds of misconceptions, ranging from wrong models of the underlying Virtual machine' to not knowing about a side effect of some command. Although generative diagnosis of misconceptions by systematic perturbations of correct domain knowledge is certainly beyond the current state of the art for domains *of* any complexity (cf Clancey, in press), the identification of errors and underlying misconceptions by the DIAGNOSER is reasonably constrained on one hand by the current goal *of* the user and a detailed USER MODEL. *) Because IPS task performance consists of many fine grained steps very detailed (interpreted) information on the current state of knowledge of the user can easily be obtained. Figure 1 presents a global overview of the architecture of EUROHELP. In the next section we will discuss the COACH into more detail.

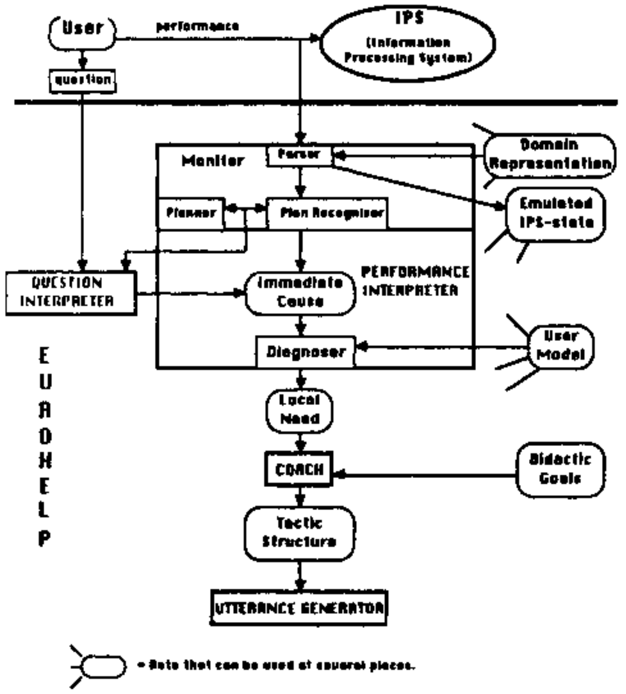

fig 2-1: Architecture of EUROHELP.

III A GENERIC CQACHINO SYSTEM

In a help system a COACH has two functions: to assist the user with a current problem and to teach the user about the IPS. These two functions support one another: correct performance facilitates learning; knowledge about the IPS enables (better) performance. However, the scope of these functions is different. Learning goals are long term goals, while the 'HELP' function is a very local one. Therefore we distinguish 'global needs', i.e. the knowledge to be acquired about a particular IPS (see 3.1.), and *local needs,* which state the current problem of the user (for more details see 3.2). Whenever a local need can be related to a global need, i.e. the user is supposed to be able to learn from the information presented, the COACH should teach; otherwise it simply presents the required information (HELP).

Presenting information consists of a sequence of 'speech acts' or *tactics.* This sequence is the result of a planning process that takes into account what to say when and how, given the identified problem of the user (i.e. the local need). This is what a 'teaching strategy' is about Current intelligent teaching systems (ITS) contain more or less fixed, prewired teaching strategies (e.g. Sleeman & Brown, 1982; Self, in press). In the EUROHELP.PO -a help system for Unix-Mail- coaching strategies were also prewired in the form of fixed frames in which topics (what to say) could be inserted. However, the large variety of potential local needs requires a more generative approach. As literature (e.g. Ohlsson, in press) and empirical data (see section 4) show: there are no *fixed* coaching strategies. They are flexibly generated as a function of the current problem and state of knowledge of the user.

The structure of the COACH consists of the following three layers, which are similar to those proposed by Woolf & McDonald (1984), but with much more functional differentiation.

DIDACTIC GOAL GENERATOR. Didactic goals are overlays of domain concepts that provide a didactic view of the domain; they form the hidden curriculum of a help system and are similar to 'genetic graphs* (Goldstein 1982) (see section 3.1)

STRATEGY PLANNER. The second layer contains a planner that constructs the coaching strategies, and will be discussed in section 3.2.

TACTICS. The third layer contains the tactics, and is a data structure. Tactics are the terminal elements of strategies. They are 'speech acts', consisting *of* a communication act, which embeds a proposition about the 'topic of discourse'. In other words: the 'how' embeds the 'what', e.g. "To give you an example (2dd deletes 2 lines from the current cursor position)". In actual discourse, the communication act is often deleted. A major distinction can be made for tactics which have propositions that contain domain knowledge, and those that refer to the process of communication itself (e.g. questions like: Did you understand what I just explained). Because the 'what' and the 'how^1 are strongly related tactics can be typed both by their communication act, and by their goal, i.e. the kind of change of state of knowledge they try to induce in the user's mind. This goal is dependent on the state of knowledge of the user (e.g. known, unknown), the type of knowledge (operational or support domain knowledge (see Clancey, 1983), communication context), and the type of object within this knowledge (concept, relation, history, state, etc.). Currently, we have distinguished 23 goals for tactics (Winkels et a!., 1986; see also section 4).

A. Didactic Goal Generation

A didactic goal refers to a domain concept, or to parts

*) If the PLAN RECOGNISER is not able to eatablith the currant goal, multiple hypothetet about mitconceptiont can often be ruled out by atking the uter about what the intendt to accomplith

of a concept, if it is a complex one -e.g. a model of how the buffers of an editor function-. Like a genetic graph, didactic goals are related. However, they are not, as in a genetic graph (Goldstein, 1982) some substitute for the domain representation, but only indicate the didactic principle that underlies learning a new concept, given the fact that another concept is already mastered. These relations are based on principles of (machine) learning, and specify 'least effort* knowledge and skill acquisition trajectories through the domain representation. The Didactic Goals do not have to be specified by the developer of a help system for a particular domain, but are derived from the Domain Representation (and the User Model (see below)) by the DIDACTIC GOAL GENERATOR. In principle, the DIDACTIC GOAL GENERATOR matches pairs of concepts and derives from their differences whether it fits one of the didactic relations. This is an exhaustive costly process, but for all relations except one (abstraction-concretion) the various networks can be generated completely once the domain representation is specified. The following relations are identified:

Generalisation-specification The more general a concept the larger its applicability and the less attributes it has. Therefore, ordering concepts (e.g. commands) according to whether one is a further specification of the other provides an optimisation of learning and of scope *of* applicability, these relations are generated on the basis of variabilisation, is_a and part_of hierarchies, disjunction/conjunction of attributes, etc.

Inversion Many actions in an IPS have an inverse, which can be derived from the highest layer of the Domain Representation, which contains all generic actions for a specific domain. Most inversion relations are very local, i.e. only between pairs of concepts.

Divergence-Convergence Because objects in an IPS may have different sizes of 'grain' (e.g. character vs file in an editor) from the point of view of efficient performance (and not of learning) it is more profitable to acquire first skills that handle 'large size' and 'small size' objects, than starting at some medium size. This relation can conflict with the generalisation-specification relation, but initially this relation has priority (ceteris paribus; see below).

Analogy Because EUROHELP has means to map an 'Information System Metaphor' (ISM) onto the domain representation, the mapping relations (analogies) can be used for didactic purposes. The discussion of the ISM is beyond the scope of this paper.

Abstraction-concretion When certain commands are reasonably well practiced by the user explaining underlying objects and models helps to induce a coherent model of the IPS. On the other hand, new abstract concepts are well explained by pointing to their concrete manifestations, the inverse of this relation represents an important didactic principle as well. It is beyond the scope of this paper to describe in detail how this relation is dependent on distinguishing operational from support knowledge (Clancey, 1983). Because commands may share objects, and because the transition from concrete to abstract is also dependent on the *shared* skill with respect to an object, this relation can only be generated dynamically for each user per session.

Because the Didactic Goals are interrelated in multiple ways, conflicts may arise regarding which expansions, i.e. instructing a new concept, should be presented. Therefore, a 'strategic* layer is specified which contains rules for establishing priorities. Currently, an experimental prototype of the DGG is under construction: functional specifications have been written.

B. The Planning of Coaching Strategy's.

Once the current problem of the user, the *local need* has been identified it will be sent to the COACH. Three types of local needs, corresponding to the three functions of coaching are distinguished:

error: when a user issues a non-executable command or performs in a way diagnosed as not intended or non optimal,

occasion for expansion: when the performance of the user provides an occasion to introduce a new concept (a DIDACTIC GOAL, see previous section),

lack of feedback: when the feedback of the IPS is assumed to be insufficient for a user.

Type of local need	coaching function
error	remediate
occasion for expansion	expand
lack of feedback	provide feedback

Besides the performance, a question can, in combination with performance, lead *to* one *of* the three local nseds.

The local need also contains the *immediate cause* (i.e. the thing that triggered it, see above) and a *diagnosis* of the specific lack of knowledge or misconception that explains the local need. This diagnosis thereby provides the topic(s) of the local need and gives already some information about the state of knowledge the user has concerning this topic (from the USER MODEL).

An example of a local need for the Vi domain, representing the fact that a user does not know she used a certain command, is:

```
local_need(error(executable, seHousness(moderate), -> TYPE
          tperformance_history(tp])]),           ->         MMEDATE CAUSE
          tdiagno8ls(lack_ofJcnowledge,                    •> DIAGNOSIS
         topic(concept(p),
         support(unknown), operational(unknown),
          [attribute(main_effect(p)]))J,
         certainty(certain)).
```

In case the local need is for example of type error, the immediate cause will refer to what the error is, and the diagnosis will denote the knowledge that is needed to avoid the error in the future. In the example above, the user accidently executes an unknown command (p), which results in putting back in the text the last piece of text that has been deleted. The topic provided by the diagnosis is the concept of p. The user should be made aware of the main effect of p.

The first decision the Coach has to make, is whether to provide pure help or to coach. If the topic of the local need is expandable from the (assumed) current knowledge of the user, in fact, if it is part of the current DIDACTIC GOALS, real coaching is in order. If not, one can assume that the user will not be able to retain the new information and the Coach will decide to provide help, which in this view is a stripped version of coaching.

Next the STRATEGY PLANNER will choose a top-level strategy to tackle this local need. Top-level strategies are: Remedial, Expansion and State Feedback. There is no fundamental difference between top-level strategies and substrategies: The strategy first picked is the top-level one. In fact, the same strategy can be a substrategy later on. For instance, many remedials imply an expansion of the knowledge of a user.

At this point a potentially recursive cycle of *strategy selection* and *strategy refinement* starts. The top-level strategies point to sets of prestored *skeletal strategies (cf.* Friedland and Iwasaki's notion of "skeletal plans" - Friedlnnd e.a., 1985) These skeletal strategies consist of sequences of substrategies and depending on whether the Help version is wanted or not, they are shallow or not.

Such a set of strategies consists of very general and very specific ones. The general ones are not immediately applicable. They need further refinement to accommodate the current situation. Specific strategies are applicable as they are but require a matching current situation. If the current situation does not match one of the available, specific strategies, a general one will be chosen.

After a plan has been selected, the refinement process starts. Every substrategy will be refined until it maps directly to a tactic, the terminal element that 'can be executed right away*. It may sometimes turn out to be necessary to consult other modules of the IHS, like e.g. the PERFORMANCE PLANNER to construct a plan to achieve some goal (e.g. a repair to undo a current state) or the USER MODEL to find out if a topic is known or not.

Refinement of substrategies is in principle guided by heuristics. Sometimes, however, refinement may be done by another cycle of plan selection, after which of course further refinement has to be done. This can be the case when strategies are inserted, of which also sets of plans exist. For example, a repair substrategy may be inserted in a general remedial plan, and a refinement of that repair may consist of selecting some special repair plan.

An example of a heuristic refinement rule is:

```
IF error(executeble) & error(seriousness(high))
THEN insert_next_to(strategy(repair), strategy(context))
```

which means that after the context-strategy, which for instance explains the immediate cause of an error, a repair strategy is inserted. A repair strategy consists of a plan to undo the effects of an executable error. In case the error is a serious one, the user may worry about this undoing, so this information is provided rather early; in case the error is not a very serious one, such a repair strategy is better delayed until the user has a full understanding of the (immediate) cause of the error.

Ultimately we will thus have a complex structure with tactics as terminal elements. There may be a lot of redundancy or inefficient use of tactics (or their *propositions),* so some pruning will have to take place. The resulting TACTIC STRUCTURE will then be fed through the UTTERANCE GENERATOR. The tactic structure represents the major pragmatic issues; the Utterance Generator supplies additional pragmatic, text-semantic, syntactic and lexical processing.

For now the process of output generation consists of filling slots in textframes which are associated with the tactics. Figure 2 gives a general overview of the Strategy Planner as discussed. A summary of the processing of the COACH is presented in Figure 1.

```
          input                 process                output


layer 1:  Domain Representation--> DGG............> DIDACTIC GOALS


          User Model. . . . . . . . . . . . . . . . .|
layer 2:  (Planner, etc)                              |

                      v  v
          Local Need--> STRATEGY PLANNER. . . .> User Model update
                     /
                   /
layer 3:  TACTICS                        |-> STRUCTURE OF TACTICS


                                                 v
                                         Utterance Generator
```

Figure 2: Input, output and processing of the COACH (components of the COACH are in CAPITALS)

1. An example

For a simple example we will take the local need as presented above. Given the fact that the topic of the local need is unknown, HELP is in order and because the local need type is 'error', the REMEDIAL strategy is chosen. We will assume there is no specific plan for this situation, so the most general one will be selected.

The top-level strategy then, looks like this:

[remedial,[announcement,context,new_information,consolidation,evaluation)]

Of these substrategies, the "context" and the "new information" are the most important ones. The first is for establishing a context for the second, the new information. This context can be found in the immediate cause of the local need, the new information can be found in the diagnosis and besides that in the Domain Representation and the Didactic Goals. Figure 3 shows the refinement of this remedial strategy. The protocol describing the refinement presents information on the local need, and how this information is used in the refinement process. The protocol ends by specifying the tactic structure. The text generated is what the COACH would say in the present situation.

Fig. 3 Protocol of planning a remedial

```
Protocol EUROHELP Strategy Planner - Version 1.0

Where does the local need come from:
1) diagnoser
2) file
Enter your choice:2.
Please enter the filename: ln6.
OK. The local need is:
local_need(error(executable, seriousness(moderate)),
      [performance_history([p])],
      [diafnosis(lack_of_knowledge, topic(concept(p),
       support (unknown), operational(unknown),
       [main_effect(p)]))],
      certainty (certain)).

 »> picking the top-level strategy...
 >» checking the Didactic Goals
 «< assuming concept(p) is not part of didactic goals (operational/support)
 >» consulting the Performance Planner for a plan to undo [p]
 <«  plan([crsr_up,  dd])
 >» consulting the User Model for crsr_up (operational/support).
 <<< known(weak)known(weak)
 >» consulting the User Model for dd (operational/support).
 <«   known(weak)/unknown

rented i al (local__need6)
   clarify
      announcement
         drawing__attention
            (interruption(proposition22)]
      context
         clarification(perf_hist6)
            [instantiation(proposition2S)j
      new_information
         describe(topicll)
            describe_support(topicl 1)
               informationl(topicll)
                  [description(proposition24)]
            describe__operation al (topic 11)
               concretion( topic 11)
                  (operationalisation(proposition25)]
            state__feedback
               describe_state__change(topic 11)
            check__assumption
               [elicitation(proposition26)]
      repair
         describe(topicl2)
            information2(topicl2)
               (direction(topicl2)]

May I have your attention please.
You just did (p] , that is what went wrong.
"p" is put buffer in text.
Practically "p" means what you deleted last will be inserted in the text
Is this you intention?
|: no.
To undo do (crsr_up, dd].
```

2. The current state of affairs

At the moment a prototype of the Coach as described above exists in Prolog, in the near future an Interlisp version will be constructed with some extra features. One obvious necessity is the possibility to postpone part of a strategy, because EUROHELP will have to be concise. Experience with the EUROHELP.P0 and our experiments has taught us users of IPSs do not want to be told too much at one time, otherwise they ignore the information and get on with their work.

IV THE TUTOR EXPERIMENT

In order to evaluate the structure of the COACH as specified before, an on-line help-experiment has been conducted (Winkels, et al., 1986). The structure of the Coach reflects a theory of how tutors will respond to user's needs in handling and learning about an IPS. The experiment is meant to find out whether the tactics proposed are indeed present and sufficient. Besides the identification of tactics per se the clustering of tactics into standard patterns is of importance, for such patterns reflect underlying strategies. Empirical information regarding the existence of such clusters would provide support for the PLANNING OF COACHING STRATEGIES as proposed before.

The domain chosen for this investigation is the domain of text-editing, i.e. Unix Vi because of the concepts involved are fairly complex and efficient performance calls for more or less elaborate planning. Therefore, such a domain provides a rich context for collecting information on how a HELP-system could function as an aid in handling and learning about an IPS.

A. Experimental procedure

In the experiment we tried to mimic the prospective HELP-system as closely as possible. This implies that tutors should not be able to interpret the text the user is working on, because a HELP-system would neither be able to interpret the meaning of a text. Text interpretation is prevented by presenting the tutors with a scrambled version of the text the user is processing.

The Vi-user, and the tutor are in separate rooms. Such separation first forces them to communicate via a terminal and secondly provides an opportunity to think aloud during task performance. Both tutor and user are provided with two terminals, one for communication and one for Vi-processing. A record is kept of both the ongoing dialogue (communication protocol) and the processing *of* Vi (Vi-protocol), and stored for later usage. The users, seven in all, were novices in varying degree. The different tutors participating in the experiment were all very experienced users of Vi.

The users who participated in the experiment were presented with a set of tasks representative of editing tasks: insert a piece of text, interchange two words, move a piece of text, delete a piece of text, replace a word, etc. Before the first session started, all users were provided with a short instruction, explaining basic-commands. These commands were selected in such a way that all tasks presented could be performed, but not in a very efficient manner. This provided the participating tutors with ample opportunity to take the initiative and provide help or instruction (expansions). The tutors were instructed to interrupt whenever they thought appropriate. Both user and tutor were asked to think aloud during experimental sessions. In this way, three different types of data have been collected:

1) communication-protocol
2) record of the Vi-performance
3) thinking-aloud-protocols

B. Results

The first part of the data analysis to be described here was solely concerned with the identification of "tactics". The main source of data for this analysis is provided by the communication-protocol containing all tutorial actions. To code the communication-protocols in terms of tactics, all protocols were partitioned into episodes, and within each episode into speech acts. Following this partitioning, each and every speech act has been coded as a particular tactic. The next figure presents an example of a part of a coded communication protocol.

Tutor/User dialogue	Type	Object type	Goal
"tutor" "2120" "18"			
As a consequence of the i command you gave,	instantiat.	p«rf. history	clarification
you are back in insert mode,	instantiat.	actual state	context ref.
that means all you type in will actually be inserted in the text.	operationalisation	concept	concretion
"tutor" "2204" "26"			
You cannot wipe out a new line (return), by delete or x.	limitation	concept	distinction
But you can join two lines again into one line by moving the cursor to the upper line and then giving the J command.	direction	implication	information
Thus try an arrow up and J.	direction	implication	information
It should be capital J.	direction	implication	remind.
"tutor" "245r "2"			
Excellent.	evaluation	perf. hist.	motivation

Fig A: An example of a coded protocol

Apart from coding the communication-protocols in terms of different tactics, clusters of tactics, reflecting a particular strategy have been identified. Indeed, there appear to exist frequent clusters, reflecting underlying strategies, such as typical REMEDIALS and EXPANSIONS. As has been suggested earlier tactics belonging to a cluster are frequently left out by the tutors for reasons of conciseness and efficiency. The output generated by the human tutors has been compared with the output generated by the implemented COACH, fed with local needs derived from the log-files. It turned out that in general, the tactic-structure generated by the COACH matched the structure generated by the human tutors quite well. Although, the human tutors are much more fluent in their expression, the COACH is more explicit in announcing interruptions and in referring to the performance history, stating for instance: "You iust did x. that is what went wrong". Sometimes this kind of reference is called for, but in other cases the user is quite aware of what she did, and there is no need to remind her.

The thinking-aloud-protocols of the tutors contain additional information on the importance and the nature of planning in tutoring. The protocols provide evidence for a two-stage model; first interpretation of the user's local need, and secondly planning how to transfer the needed information, exactly as has been proposed in a top-down fashion for the planning of the coaching strategies. Especially the last stage poses a problem to the tutors. They do know what they would like to transfer, but need to consider how the information has to be transferred. They construct a sequence of tactics by explicitly planning: "I want to explain this, but first I need to get this across". Even when tutors eventually generate a sequence of tactics that occurs more often, the thinking-aloud protocols indicate that such a sequence is not prestored in memory but has to be constructed anew.

The data obtained within the experiment, both the communication-protocols and the thinking-aloud-protocols support the model for the planning of coaching strategies; basic frames do exist as identified clusters, tactics may and will be deleted. Tutoring does not just mean executing a prewired plan, but means adjusting skeletal plans to current circumstances on the basis of planning. Such planning is evidenced by the difficulties tutors expressed: "I should not have told this now, I just forgot to tell him", "I just tell her what to do, but she will not understand it", "well I hope this is clear, I am afraid I am telling too much". The tactical clusters identified are themes with variations, the latter reflecting the flexibility induced by the planning process, phrased by tutors on how to convey particular information, and by the recurring clusters of tactics. In other words tactical clusters are themes with variations, the latter reflecting the flexibility in the planning.

V CONCLUDING REMARKS

The research described in this paper reflects a methodology which combines and integrates a top-down and a bottom-up approach for the construction of Intelligent Help Systems. Existing ideas on the functionality of the COACH provided a framework for a top-down analysis of COACHING strategies which resulted in the construction of a PROTOTYPICAL STRATEGY PLANNER. The complementing bottom-up approach consisted of gathering empirical data, identifying tactics and clusters of tactics.

Both approaches combined, present a detailed picture of the nature of COACHING strategies. The methodology represents a powerful research tool. First, it provides means to gather and analyse data from different sources. Secondly, it allows for interaction between different levels of analysis. For example, tactics identified in the bottom-up analysis were used by the PROTOTYPICAL STRATEGY PLANNER in the top-down construction of strategies.

Apart from the above methodological considerations, the major research result concerns the nature of coaching strategies. One might easily assume there exist a number of prewired strategies that only need to be retrieved from long term memory when appropriate. But this does not seem to be the case. Human tutors tend to build strategies when confronted with a particular situation. They may be guided by a general strategy, like: "I need to correct this misconception". But how this misconception is corrected is planned on the spot.

If we want our coaching system to be as flexible in the use of strategies as human tutors turn out to be, we need a STRATEGY PLANNER instead of prewired, fixed strategies. The STRATEGY PLANNER as presented here, seems to satisfy the goal of flexibility in that it is capable of generating the same sort of tactic structures as our human tutors.

REFERENCES

1 Anderson, J. R., Boyle, C.F. A Yost, G. (1985). The geometry tutor. Proceedings of the. 2Ih International Conference OJI Artificial Intelligence, Los Altos, Kaufmann, (pp. 1-7).

2 Brachman, R.J. (1978) A structural paradigm for representing knowledge. Boston, BBN-Report 3605.

3 Breuker, J.A., (in press). Coaching in Help Systems. To be published in: J. Self (ed).

4 Baaren, J. van der, (1986). Diagnostic modelling in intelligent HELP systems. COE/EUROHELP/020. Courseware Europe BV, Purmerend (the Netherlands).

5 Clancey, W.J. (1983). The epistemology of a rule based system -a framework for explanation. Artificial Intelligence. 20, pp 215-251.

6 Clancey, W.J. (in press). Qualitative User Models.

7 Duursma, C, Maas, S. & Romeyn, T. (1986). A next step towards an epistemological description of an IPS. Report COE/EUROHELP/016. Courseware Europe BV, Purmerend (the Netherlands).

8 Fischer, G., Lemke, G. & Schwab, T. (1985). Knowledge-based help systems. CHF85 Proceedings, pp 161-167.

9 Friedland, P.E., & Iwasaki, Y., (1985). The concept and implementation of skeletal plans. Journal of Automated Reasoning. I, 161-208.

10 Goldstein, LP. (1982). The genetic graph: a representation for the evolution of procedural knowledge. In: Sleeman, D. & Brown, J.S. (eds), (1982).

11 Hartley, R. A Smith, M.J. (in press). Experiences in explanation giving in the EUROHELP Project. In: Self (in press).

12 Lehnert, w (1978). IM pjsfifisi ol question answering.; a computer simulation ol cognition. Hillsdale, NJ: Erlbaum.

13 Moran, T.P. (1981). The command language grammar: a representation for the user interface of interactive computer systems. International Journal of Man-Machine Studies, 15, 3-50.

14 Norman. D., (1983). Some observations on mental models. In: D. Gentner & A. Stevens (eds): Mental Models. Hillsdale NJ: Erlbaum Associates.

15 Ohlsson, S., (in press). Some principles of intelligent tutoring. University of Pittsburgh. To appear in Instructional Science-

16 Quillici, A.E., Dyer, M.G. & Flowers, M. (1986). AQUA, an intelligent UNIX advisor. Proceedings of *THE llh* European Conference on Artificial Intelligence, Vol II, pp 33-38.

17 Self, J. (ed) (in press). Intelligent Computer-Aided Instruction. London: Chapman & Hall.

18 Sleeman, D. & Brown, J.S. (eds) (1982). Intelligent TutorinR SYStema. New York: Academic Press.

19 Wilensky, R., Arens, Y. & Chin, D. (1984). Talking to UNIX in English: an overview of UC, Communications of the ACM, 27

20 Winkels, R.G.F., Sandberg, J.A.C. A Breuker, J.A. (1986). Coaching strategies and tactics of IHSs. Deliverable 2.2.2, UAM/EUROHELP/06. University of Amsterdam.

21 Woolf, B. & McDonald, D.D., (1984) Context dependent transitions in tutoring discourse. Proceedings oj* AAAI 1984. Los Altos, Kaufmann.