Question-Answering for Intelligent On-Line Help: The Process of Intelligent Responding

RACHEL M. PILKINGTON
The University of Leeds, United Kingdom

This article examines the process of specifying a question-answering help facility in the context of UNIX mail. The specification was based upon experimental expert-user facilitative dialogues. These dialogues were analyzed using a classification scheme developed for the purpose. The scheme provides a meta-language for describing patterns of intent and rhetorical structure in dialogue. Using the scheme as a tool, common patterns in expert-user dialogue emerged, providing insights into both tutoring strategy and the linguistic forms required to generate help output. These were refined to form a set of frames consisting of stereotypical patterns of rhetorical predicates to answer different types of questions which could be adapted to the needs of a particular user. The significance of the system developed from this specification is discussed.

THE NEED FOR HELP SYSTEMS WITH A NATURAL LANGUAGE CAPABILITY

It is well known that there are problems associated with using computer systems. The learning curve is not only steep at first, but often flattens out before real skill is acquired. It follows that a help facility's role is to encourage users to seek out the full functionality of the system they are learning and not just to "ease them in" at the beginning (see, e.g., Streitz, 1988). An increasingly popular solution to the easing-in problem has been to try to make the interface "so intuitive that users would require little or no assistance in learning to use" it (Ringle & Halstead-Nussloch, 1989, p. 228). This approach explains the success of menu-driven window-based applications and the increasing popularity of icons and graphical representations.

I gratefully acknowledge the help of all those involved in the Eurohelp project, and in particular, Roger Hartley, Andrew Cole, and Ken Tait of the Computer Based Learning Unit, and Colin Tattersall, formerly of the Computer Based Learning Unit and now at PTT Research, Groningen, The Netherlands.

Correspondence and requests for reprints should be sent to Rachel M. Pilkington, Computer Based Learning Unit, University of Leeds, Leeds, W. Yorkshire LS2 9JT, United Kingdom.
For example, the “desk-top metaphor” approach adopts a vocabulary (folders, files, waste-baskets), that is familiar and easily identified with the more conventional “paper office.” However, we use a software environment in order to improve efficiency, so the software is not merely quantitatively different but qualitatively different from the paper office. It follows that there is a “conflict between the extent of the functionality we want to have in the computer system and...compatibility between existing mental models and system realizations” (Streitz, 1988, p. 181). As Ringle and Halstead-Nusloch put it, the more complex operations, characteristic of powerful systems, cannot easily be reduced to pictorial or menu-based command structures. For operating systems like UNIX with over 600 commands, it is not possible to develop a different icon for each of them.

This is the main reason for continuing to try to build natural language interfaces to deliver intelligent on-line help. Natural language has the advantage of combining maximum transparency (ease of understanding) with maximum flexibility (in the kinds of concepts that can be expressed).

Conventional computer-based help systems do not make full use of these properties of language. This is because the text has to be prestored, which means that the author must guess every possible user’s task in advance. If the user guesses wrongly, then the task will not appear in the menu and the corresponding text will not be relevant when it is displayed. This approach is not at all like talking to a human expert. Many writers have commented on the need for more “reactive” help “dialogues” that can focus in on the needs of the user over successive diagnostic question and answer turns (O’Malley, 1986; Pollack, 1986; Wilensky, Arens, & Chin, 1984).

A good illustration of this process is provided in Ringle and Bruce’s (1982) study in which researchers, who had little or no prior knowledge of each other’s work, each had the goal of explaining their research to the other. Misunderstanding was so frequent that Ringle and Bruce suggested conversation failure is the norm rather than the exception. This is not as serious a problem as it sounds because the “checking move” (a signal of misunderstanding or failure), precipitates repair work by one or both parties. Failure cues include amplifying, refining, or summarizing what the other has said, making an assertion, or asking a question, while repair moves involve displaying each other’s beliefs for examination.

In a study nearer to the help context, Moore and Swartout (1989) also found that advice seekers frequently did not understand the instructor’s response but asked follow-up questions requiring clarification, elaboration, or reexplanation. As Moore and Swartout pointed out: “the problem is that both expert system explanation and natural language generation systems, view generating responses as a one-shot process” (p. 1504), and this is simply not good enough.

---

1 Trademark of AT&T Bell Laboratories.
Like Moore and Swartout, the work presented here was motivated by the desire to describe formally the "reactive" help dialogue needed by on-line help systems. The model is based upon empirical data, collected using the "Wizard of Oz" technique. In this technique a human expert simulates the advisory system and by analyzing the keyboard interaction between expert and user, insights can be gained into the structure and style of the help required.

Thus, Grosz (1978) looked at collaborative and facilitative dialogues to reveal how dialogue focus is maintained in relation to a task goal and Guindon, Sladkey, Brunner, and Conner (1986), like Grosz, found evidence for a relationship between dialogue and task structure. It is thus clear that task-based dialogue has a structure across utterances and, just as text structure is signalled by certain linguistic devices, so is dialogue structure.

In the work reported here we analyzed help dialogue to formalize a model of reactive help. This model served as a design specification for an on-line help system (see Pilkington, 1988). This specification guided the implementation of PORSCHE (Producer of Rhetorically Structured Help; Tattersall, in press). This article describes the empirical basis for PORSCHE and shows how and why PORSCHE represents an advance in intelligent help systems.

**INTELLIGENT EDUCATION SYSTEMS**

A system intended to help users acquire new concepts ought to be guided by good tutoring principles. Durchastel (1990), in reviewing the use of the computer as a cognitive tool, made a distinction between "power tools" and "assimilatory tools." Power tools attempt to go beyond assimilation, in the Piagetian sense, and are aimed at accommodation. The problem with purely assimilatory tools is that the framework for thinking in a particular subject may not have been acquired. In this case, the structural pegs on which to hang newly encountered material are not already present, and new information cannot be assimilated without loss or error. Thus, novices need a "structured approach" (see Ausubel, 1963), so that when new material is introduced they know how to assimilate it.

Similarly, Britton, Glynn, and Smith (1985) argued that comprehension can be helped by removing from the learner some of the cognitive load imposed by the material. This is achieved if relevant schemata are activated in advance, termed prefetching. Prefetching is contrasted with demand fetching, which refers to activating schemata during processing. Prefetching has the advantage that task performance need not be interrupted to fetch the schema. Therefore, one means of reducing cognitive load is to present expository units in a way that reduces demand fetching, so minimizing loss and error. Power tools attempt to address this need by presenting information in a structured way. Such tools need many powerful components.
Thus, Hartley's (1973) "classic formulation" of the Intelligent Educational System (Cummings & Self, 1990) consists of three obligatory subsystems: (1) domain knowledge; (2) the learner model, a model of what the system believes the user already knows; and (3) strategic knowledge, knowledge about how to teach the subject. To support these subsystems a "state of the art" help system such as UC (Wilensky et al., 1984), requires many modules. An input mechanism for users' queries, a planning component to respond to requests for direction, a user model, a text-construct module, and an output generator.

The time taken to design and develop such large and complex systems often makes their value as marketable—as opposed to merely experimental systems—questionable. As Ringle and Halstead-Nussloch (1989) pointed out, natural language processing components involve much greater processing and memory demands than more conventional help systems and in some cases they can dwarf the application package they are designed to support. To address this criticism, system design must be aimed at creating a tool that aids the construction of other systems. Such tools can then be used to reduce the amount of time taken to develop a new application.

For example, UC's modules depend upon a knowledge representation that requires hand-crafted didactic information for teaching purposes. The construction of this material is time consuming and reduces the potential portability of the design specification to other applications.

Keeping system design generic, and providing genuine context sensitivity, are often competing goals. As Tattersall (1990) pointed out, lack of real context-sensitivity can damage the utility of a help system. For this, the system must be able to check the current state of the application before answering a user's question. This is necessary so that it can (a) ensure that the user is informed of preconditions which have been satisfied, (b) ensure that any hypothetical examples used to illustrate commands would not result in harmful effects if interpreted literally, and (c) offer responses that "remain appropriate in the face of errors" (see Woolf & McDonald, 1984). It is in these respects that even sophisticated help systems like UC remain a compromise.

It is very difficult to design a system that responds appropriately in the face of errors because identifying errors is difficult. If we have to define and store all the possible misconceptions a student may have about a system, then remediation seems intractable unless domains are very simple. This is why Cummings and Self (1990) argued in favor of dialogue systems. An intelligent dialogue would allow the system to ask the user questions and to check the user's understanding of particular concepts.

In UC, user modeling is limited to stereotypical classes of user. This carries the assumption that all beginners know a subset of what intermediates know, which, in turn, is a subset of what experts know. However, Draper
(1984) showed that this is not valid for UNIX command vocabulary because even experts remain novices in some aspects of the domain and these aspects differ among experts. No assumptions can safely be made about what a "beginner" or "expert" will know. As Self (1988) suggested, a good approach to take, in these circumstances, is to get the user to tell you what he or she knows. However, error diagnosis is far from the only reason for wishing to use dialogue. Cummings and Self (1990) suggested that meta-level thinking is an essential component of expertise encouraged by task-based discussion. Eventually, aspects of the discussion are internalized in the form of self-questioning skills. If the staging of dialogue (the structure imparted through the order and style of presentation) directly contributes to this process, then intelligent help system design must adopt appropriate text and dialogue forms. To do this, one must first know what these forms are.

STRUCTURING DIALOGUE
AND TEXTUAL RESPONSES TO SUPPORT LEARNING

For teaching purposes, there are two essential dimensions of content to be included, the first is the identification-specification dimension, which names, defines, and differentiates between concepts of interest. The second is the causal-contiguous dimension, which describes the interactions between agents and objects, which in turn, cause state changes and events of interest.

Textual exposition is staged to accommodate these dimensions using rhetorical predicates (Meyer, 1975). Meyer (1984) suggested that skilled readers approach texts with knowledge about how text is organized. Meyer noted that readers employ a structural or organizational strategy, that is, provided that readers are interested in following the writer's argument, then the structure used by the writer will also be used by the reader to form a gist of the text in their head, and this then forms the basis of the reader's subsequent recall. The model necessitates the identification of superordinate-subordinate relationships, so that parallel schemata can encode the text in an organized framework. Staging, which focuses on these relationships, increases the "levels effect": The finding that propositions placed higher in the content structure of the passage are better recalled. The author can thus focus attention on propositions central to the argument and give them special emphasis to promote their recall.

McKeown's (1985) work with the discourse planner TEXT and the response regenerator MUMBLE (McDonald, 1987) concentrated on the production of paragraph-style descriptions of a declarative nature adopting identification-specification style rhetorical predicates. TEXT generates answers to questions (in the form of a short text) using rhetorical predicates plus focusing rules. The knowledge base is constructed so that an area can be
selected according to its conceptual closeness to the question. The relationships in the selected area determine which schemata are employed to stage the response. TEXT represents a substantial achievement in its generation of responses on-line. Schemata include attributive, identification, constituency, and compare-contrast rhetorical predicates; these correspond roughly to Meyer's (1975) attribution, representative identification, constituency identification, and analogy predicates. However, in TEXT, the problems involved in generating process descriptions were not elaborated upon. Later work (Paris & McKeown, 1987) produced causal and spatio-temporal descriptions, although the examples given of these types of description are markedly shallower in rhetorical structure than those for attribution and constituency that were produced by TEXT, and attention to the combination and embedding of these different schemata in ways that support learning, seems lacking. Thus, these answer frames do not appear to express fully the kinds of relationships on the causal-contiguous dimension required to serve facilitative or Socratic styles of discourse and dialogue, and a system aimed at task-based help or diagnostic troubleshooting needs these schemata.

In Mann and Thompson's (1987) rhetorical structure theory, the concepts of node and satellite are used to describe text structure. These notions loosely resemble Reichman's (1985) claim and support respectively, although Meyer's (1975) predicates of attribution, response, and covariance are most commonly used to express the node or claim, and explanation, equivalent, specific, analogy, manner, replacement, and representative are routinely used as satellites (support). The node or claim is the central idea or kernel to the text that the writer wishes to be accepted by the reader. In order to increase the probability that this idea is understood and accepted, some sort of satellite support is offered. This support may be evidence (of a concrete or perceptually orientated type), or it may be a causal chain, or a specific familiar example that illustrates a general principle. The value of both Meyer's (1975) graph structure and Mann and Thompson's rhetorical structure theory lies in the way in which they describe how one schema—the dominant schema—serves to organize the text as whole, yielding subschemata in a hierarchical decomposition. It is this structure that allows the reader to see how each part of the text relates plausibly to the next, and this is why I want to capture this structure for use in computer-based explanation.

Hovy (1988) was among the first to attempt the shift away from using rhetorical predicates as an analytical tool—for describing the structure of discourse—towards using them as a formal tool to generate text in computational systems. Hovy's work addresses the problems discussed in relation to McKeown's (1985) earlier work, particularly in terms of purpose and intention in help texts. For this Hovy made use of Cohen and Levesque's (1985) work on speech act theory. Hovy acknowledged that the major prob-
lem associated with his approach is knowing how much satellite information to include in a response, or rather, how to evaluate recursively the overall organization of responses as the content of a response is compiled.

Moore and Swartout's (1989) system Program Enhancement Advisor (PEA), which uses the Explainable Expert System (EES) framework, forms a further extension of this work not only by addressing process and goal-based text but also by making the text planner explicitly responsible for content selection. In other words, it doesn't just structure a response that has been predetermined, it decides what will go into the response as well.

Like Moore and Swartout (1989), my model of the text-planning process assumes that you cannot collect all and only the information which you want to include in your response without a simultaneous consideration of how it is to be organized: In other words, structure determines content, to some extent, just as content determines structure, and the two processes need to go hand in hand. Although my approach has much in common with Moore and Swartout's work, it has developed in parallel to it, rather than from it, and has been based upon the rhetorical predicates of Meyer (1975), not the rhetorical structure theory of Mann and Thompson (1987).

The model described here deals with the types of textual structures required in help contexts. These organizational schemata are derived from an empirical study of this context and they perform at the level of the "structure planner" that is antecedent to a "text generator." The schemata employed in PORSCHE are recursive in that they know how to embed each other coherently. Their "content predicates" search for the information they require in the knowledge base, and, based upon the content that they collect, other "organizational predicates" act. Content critics may further organize the response by compressing, censoring, or expanding the text according to user-model information or organizational criteria. In this way the problems of how to organize satellite information and how to decide how much satellite information to give, are controlled. Moreover, unlike McKeown's (1985) TEXT, no links that relate to explanation need to be stored in the domain-specific knowledge base. This increases the generality and portability of the PORSCHE system.

**THE EMPIRICAL BASE FOR THE MODEL:**

**EXPERIMENTAL PROCEDURE**

The experiment engaged a human expert to give instruction to a novice user. By "expert," I simply mean one familiar with most of the functionality of the application, whereas by "novice," I mean one who has not used the application before. The application is the UNIX-based vi text editor. Expert and user were seated in two different rooms and were each provided with two terminals. On one terminal they could "talk" to each other by typing
on the screen, this was called the communication terminal. The other terminal was called the task terminal. In the case of the user the task terminal was used to run vi and the user performed tasks using vi on it. In the case of the expert, the task terminal was not used, but simply displayed the contents of the user’s task terminal with two important differences. First, the text on the screen was scrambled so that the expert could not read the words the user was editing. This scrambling was within each word, so it did not affect visual structural features like word spacing and paragraph and line breaks, and the expert could also see the user’s cursor as he or she moved through the file. Second, a status line at the bottom of the screen showed the last 20 key presses the user made. The status line was the only unscrambled text the expert saw. The purpose of the scrambling was to prevent the expert from understanding the text that the user was editing and using this knowledge in responses. This would be beyond the scope of the help system envisaged.

The experimental situation is shown in Figure 1. All key presses on the communication terminals and the user’s task terminal were recorded, using the program LOG. The communication process was maintained using the program PHONE. The user and expert could not talk to each other (i.e., type to each other) at the same time, each had to wait until the other had finished typing.

The user was given a list of editing tasks to complete and a set of typed instructions to help format the text. These were graded in complexity from 1 to 5 and involved inserting and replacing text, or copying and moving sentences (including the insertion of new lines, reformatting, and moving lines across files). The user was asked to “think aloud” into a tape recorder in order to help identify problems. Users were asked to comment on the task, plans for doing the task, problems, and how they were overcome. They were particularly asked to comment on the help provided by the tutor. The tutor was also asked to think aloud and explain his strategy and tactics when giving advice to the user. These data served to supplement the communication terminal log file.

THE DEVELOPMENT AND APPLICATION OF THE ANALYSIS SCHEME

In this section, the process of analyzing the experimental protocols is explained. There are three levels of analysis: strategic (tutoring aim), tactical (methods used in response), and linguistic (rhetorical predicates in the method).

Meyer’s (1975) application of rhetorical predicate analysis divided the text into simple sentence propositions, organized under “lexical predicates,” which related their arguments together rather like a case grammar. Next, these lexical predicates were subordinated to the rhetorical predicate, which
EXPERT types requests

USER types commands

EXPERT talks

USER talks

tape recorder

tape recorder

slave terminal

communication terminal

communication terminal

task terminal

VAX computing system

scrambler

runs vi editor, maintains phone link, logs sessions

Figure 1. Diagrammatic representation of experimental procedure

links two or more propositions together. In this way links between rhetorical predicates form a graphed hierarchy of ideational relationships. Meyer’s account of the process is sketchy with respect to identifying the rhetorical predicate link. My analysis proceeded in the following way: First, the text was segmented into propositions in the manner described by Kintsch, Kozminsky, Streby, McKoon, and Keenan (1975).
To create a file you should type vi \(<filename>\), where filename is the name of the file you wish to edit.

The propositions are:

\((\text{to-create, you, file}), (\text{should-type, you, vi, }\langle\text{filename}\rangle) (\text{is, }\text{filename}) (\text{of, file, name}), (\text{wish, you}) (\text{edit, you, file}).\)

This does not tell us anything about the interrelationships among these content propositions: The sentence is composed of phrases and clauses. This can be illustrated by enclosing phrases in round brackets and clauses in square brackets.

\[\begin{array}{l}
(\text{To create}) (\text{a file}) [ (\text{you}) (\text{should type}) (\text{vi, }\langle\text{filename}\rangle) (\text{where}) (\text{filename}) (\text{is}) (\text{the name}) (\text{of the file}) (\text{you wish to edit})].
\end{array}\]

Moreover, the presentation of these clauses (each containing a predicate and subject plus none, one or more objects), has a structure. This can be indicated by nesting subordinate clauses as follows:

\[\begin{array}{l}
(\text{To create}) (\text{a file}) [ (\text{you}) (\text{should type}) (\text{vi, }\langle\text{filename}\rangle) (\text{where}) (\text{filename}) (\text{is}) (\text{the name}) (\text{of the file}) (\text{you wish to edit})].
\end{array}\]

The indentation of the second major clause indicates a subordinate topic.

These common clause patterns were described by Hoey (1983), who divided relations into logical sequence and matched relations. The preceding subordinate clause is a "matched" clause, functioning as a further refinement. In this example, attention is focused on the main "logical" clause, whereas the matched clause is used to ensure reference is clear. The structure adds meaning beyond the bare propositions. It tells us that the focus is the action of creating a file, and the rest of the sentence ensures that the object referent filename is unambiguously defined.

Once the clause boundaries and type of clause had been identified, Meyer's (1975) list of rhetorical predicates was consulted. The appropriate question to ask at this stage is "Which predicate can link clauses to paraphrase the utterance?" However, it was quickly found that clue words of stylized forms of expression for the predicates were not always present in the data. This means that an element of judgment is necessary to answer that question. Because both consistency and reapplicability need to be ensured, the predicates were defined over a range of typical forms. This refinement of Meyer's (1975) original list of predicates into a list of predicate forms is given in the Appendix. Once this was done it was sensible to ask, "If the utterance were to have been expressed using one of the predicate forms, would it mean the same thing?" This was the judgment used to decide the predicate. If the answer to this question was no in all cases, then the utterance was used to generate an alternative predicate or predicate form. This process continued until all the data propositions had
INITIATE Instruct

**goal orientation**

instrument-achievement

- To create a file
- replacement (concretize)
- you should type vi <filename>

Inform

**membership**

- where <filename> is the name of the file
- you wish to edit

*Figure 2. Graphing the response*

been captured by one form. In the example used earlier, an instrument-achievement relationship exists between the clauses [To create a file] and [you should type vi <filename>]. The main clue words to this pattern are the modal, should, attached to the action, and the infinitive, to create, used to signal the goal. The matched clause is signalled by the use of where as a subordinate conjunct. Such relationships are identified by repetition, either direct or indirect. Indirect repetition includes the substitution of semantic equivalents or pronouns. The repetition of filename indicates how this particular clause is in matched relationship with the previous clause. There are two common matched relationship pattern: The first is the generalization-example pattern, the second is the preview-detail pattern. Thus, matched patterns are often involved in comparative discourse (common in expository text) and the coordinating conjuncts such as and, or, and not often signal them. The example response can be represented graphically as in Figure 2.

In the graphed response, indentation to the left indicates subordination to the predicate above and to the right. Rhetorical predicates are underscored. Vertical lines link predicates to their propositions and to other predicates at the same level. The rhetorical predicates are not entirely equivalent to those used by Meyer (1975) (see Appendix for their description). The graphing of responses in this way makes their structure transparent and enables the identification of common patterns in the sequencing of predicates. Such common patterns were themselves labelled and their frequency
in the data was analyzed in relationship to the type of educational support offered, either elaborative, explanatory, comparative or concrete. The predicates used in a response were found to vary in systematic ways with both the type of content and the type of question. Thus, rhetorical predicate analysis serves as a meta-linguistic tool to describe the data. It is used to assess the frequency with which the expert adopts methods and aims (patterns in moves) to serve a dialogue purpose.

RESULTS:
RHETORICAL PATTERNS IN HELP UTTERANCES

Five patterns of rhetorical predicates were observed in expert responses. These are given the labels Inform, Instruct, Explain-evaluate, Explain-explore, and Compare-contrast. The Inform method adopts the predicates of membership (an is-a relationship) and attribution (which relates a concept to its properties). The mood is declarative and the aspect timeless. The Instruct method employs goal-orientation and instrument-achievement predicates with an imperative mood to give instructions. Explain-evaluate adopts cause-consequence and condition-consequence predicates in the past tense to explain a state of affairs. The Explain-explore method uses the same predicates in a conditional or future tense to express possible worlds as opposed to the actual world. Compare-contrast uses difference and similarity predicates to differentiate between concepts and may subordinate other predicate patterns. For example, a Compare-contrast method may wrap around two or more chunks of response, each framed using Inform, Instruct, or Explain methods, as shown in Figure 3.

Each of these methods can be captured by an answer frame. These frames are shown in Figure 4 and consist of lists of rhetorical predicates in a preferred sequence. Round brackets surround optical slots. Chevrons mark the slots to be filled with content propositions. At the points where comparisons may be brought in or alternative frames used, it is normal to finish talking about one subject, as far as possible, before introducing a new one. However, a comparison will be made wherever the similarity or difference occurs, so, it is possible to stop to make a comparison and return to the controlling topic later. A reintroductory phrase will then be adopted to refocus the referent. This is true whenever the focus returns to a higher level focus space (adopting a predicate further to the left schematically), or whenever a new topic entity is introduced.

INTENTION IN UTTERANCES

The analysis assumes that if the user initiates the dialogue it is because it is part of his or her strategy for achieving a goal. The first step, in the analysis, was to try to describe the intent of the question or utterance. The problems
Compare-contrast

**differentiation**

**Explain-explore** (similarities)

**instrument-achievement**

**you can yank**

**adversative**

**not only words but**

**collection**

**many different specs**

**Inform** (differences)

**representative**

"ay/xxxx"

**attribution-effect**

**will yank**

**setting-trajectory**

...to the next occurrence of the string xxxx

**representative**

"ay$

**attribution-effect**

**will yank**

**setting-trajectory**

...from the cursor to the end of the line

and so on

**Instantiate** (dialogue record)

...with regard to the first example

**Inform**

**attribution-effect**

**xxxx on its own will search**

**setting-trajectory**

...for the next occurrence of the string

**Figure 3. Using Compare-Contrast in combination with another method**
involved in using the actual words of questions to interpret the intentions of the asker were discussed by Lehnert (1978). It is not possible to map the who, why, what, and how of question asking onto a speaker's intention. For this reason, questions are grouped into functional categories on the basis of response expectations.
As competent speakers of a language, an utterance has a perlocutionary force (Searle, 1971), on the basis of which a label can be ascribed. This label addresses what the speaker seems to the hearer to want in the response.

**Enablement:** The speaker's intention is to elicit a plan of action to accomplish a task, typically "How do I...?"

**Evaluation:** The speaker's intention is to elicit causes and explanations, typically "When I did... why did/didn't... happen?"

**Exploration:** Where the speaker's intention is to check or seek approval for intended actions or plans, typically "Is it possible to...?"

**Elaboration:** Where the speaker's intention is to gain access to specific domain knowledge of the hearer, typically "What's the command/object/mode...?"

These categories are not intended to be exclusive, a single question may have more than one intention, and some intentions are more explicit than others. However, these categories capture the main focus of each user utterance. In addition, user utterances often included specific contextual information and such statements were categorized according to whether they referenced a user's goal, the system state, an action, or a previous dialogue utterance. Such information often modified the intent of the question, or was not even accompanied by a question. For example, the following question has four premodifying propositions that change the main focus of the question away from exploration towards evaluation.

I have input the text (action) in what I believe to be insert mode (state) but when I press the escape key I do not appear to leave insert mode (action and not-expected-effect) what action should I take (general exploratory question)?

In such a case it makes sense to record the question as a specific evaluation question. At the level of teaching goals, patterns in response are less marked than at the level of methods (frames). The use of methods in combination to serve the expert's purpose was highly flexible, though tuned to the context and user variables. At this level, the classification can do no more than suggest a broad aim based upon a top-down assessment of the perlocutionary force of the response. Educational responses—responses intended to teach the topic—were classified as either Initiate, Consolidate, Remind, or Repair. The think-aloud protocols assisted in this classification by indicating the expert's beliefs about the user's knowledge. An example of each follows. Initiate responses introduce a new topic and typically employ the identifying rhetorical predicates used with the Inform method.

1. **Initiate** answer in response to enablement question:

To create a file you should type `vi <filename>`, where filename is the name of the file you wish to edit.
Given the enablement question, we might have expected only the Instruct method and not the tagged Inform. A more elaborate use of Inform with Instruct occurs in Example 5. The Consolidate strategy aims to strengthen a concept that has already been introduced by relating it to more familiar concepts or well-practiced commands. Compare-contrast or Explain methods are characteristic of Consolidate. The answer graphed in Figure 3 is a Consolidate answer. In the following example the permanent and temporary buffers are contrasted.

2. Consolidate answer to enablement question:

Use \texttt{gc<buffer-name> p e.g., \texttt{ap} for the previous example. (NB. You can also use capital p, the difference is as for the temporary buffer.)}

A Remind response references a topic that is well known to the user. This may be obvious from the question e.g., “What was the yank command again?” or from the talk-aloud “He’s forgotten that, I’ll just remind him.” In such cases the tutor’s help may be reduced below that of noneducational responses, for example,

3. Remind response to enablement question:

“Use the yank command. Remember, capital Y!”

A Repair response is used when the tutor infers inappropriate or incorrect use of commands. Typically, the Explain-evaluate method is followed by an Instruct method.

4. Repair response to evaluation question:

What actually happened was, you initially tried to enter the number 3 while in command mode, you realized this and went into insert mode. However, vi allows the prefixing of commands with a number, so, you prefixed a command with number 3. I see the 333333...3 are all in one line, so you can position your cursor at the start of them and use the \texttt{x} command to delete them.

In addition to answering questions cooperatively, the tutor had the goal of teaching the user about the system, and the selection of answer methods fulfilled both roles. This process is best illustrated by examining responses that caused a conflict between the two goals. For the question “How do I put the text on sheet B into the file?” which is an enablement question (so, from its perlocutionary force, we expect an Instruct method), there is a conflict for the tutor between answering the question cooperatively and teaching about the domain (the expert knows that the user is unfamiliar with the concept of modes). The tutor does not answer “type \texttt{i}, type your text, then press escape” (which would be a perfectly cooperative answer if the user already knew about the two different modes). Instead, the tutor initiates using the Inform method and only uses the Instruct method once
the necessary "cognitive pegs" for organizing the commands in memory have been instantiated.

5. There are two different modes in vi, one is command mode, where various instructions are given to the editor, the other is insert mode, which will put characters typed by you into the file. To get into insert mode you should type i and to leave insert mode you should type escape—the escape key.

The answer employs Compare-contrast wrapped around the Inform method followed by the Instruct method. In general the tutor prefers to begin with the more general and abstract content of the response and becomes more specific to incorporate the task or the user's goal. This is in accordance with Brown and Yule's (1983) suggestion that ordering is not just a point of style but has consequences for coherence. However, the type of question is influential. In general the more specific the question and the more competent the user the more likely the tutor is to reverse this order and begin with specific instruction. The difficulty for the tutor is that he must first make assumptions about what is relevant (decide what the user knows and what is needed for the task) and then he must decide how to express it.

On the other hand, the difficulty for users is finding a way to express what they want. Restricted by the lack of face-to-face contact, users could not show the tutor what they wanted to do—by pointing at areas of the screen—and so had to adopt a vocabulary to try to explain their intended actions. Because the user is often unaware of the ways in which the system operates, this vocabulary is often inappropriate. For example, a user request to know how to delete a return may be motivated by the need to add something to the end of the previous line, or remove space (created by deletion of characters in the first line). In the latter case it is necessary to join the line to the next one. The command J for join or A for append-at-end-of-line are very different (conceptually and in terms of system function) and the tutor will need to follow up the initial question to decide which is wanted.

The key advantage of the human expert or tutor over a conventional help system is this ability to extend question answering into dialogue and so focus on the issues that require explanation. In order to do so, two kinds of noneducational response were used. The first, the help-response, signals that the tutor is willing to Instruct the user, but because he needs more information, he doesn't know how best to approach it. Instruction is partial or incomplete and generally ends with a tagged question. The second noneducational response, the excuse response, makes no attempt at instruction, instead the tutor responds with a question or statement indicating that he is not yet ready to answer. For example "Are you sure you are in command mode?" The role of these moves in extended dialogue is illustrated below:
6. I cannot seem to get back to editing file vib, I get a message saying no write since last change (:editl overrides).
7. The error message is saying that the file you are now editing has not been written to disk, but if you really want to do the command :e then you can use :el to override it. Let me orientate myself, what file are you in at the moment?
   [expert elicits follow-up]
8. I'm sure I'm in via, but can I check this?
   [user elicits follow-up]
9. Yes the :f command will tell you the filename
   [Expert addresses subtopic that user has elicited]
10. Yes it's via
    [user closes subtopic]
11. Do you want to transfer the contents of this file into vib?
    [expert elicits further follow-up]
12. No, just the contents of a couple of named buffers, which contain some text from via.
    [user addresses subtopic that expert has elicited]
13. Ok use the :el vib command which will override the error message and let you edit file vib.
    [closes both the topic and the interaction by addressing the task-goal]

The preceding dialogue illustrates that answering intelligently includes knowing how to elicit a question you can answer. The need to adopt the preceding role is common with Repair answers to evaluation questions and suggests a method for diagnostic troubleshooting. Up to 40% of expert-user interactions required this kind of follow-up.

**SUMMARY STATISTICS**

Using the question and response categories just described, an impression can be gained of their respective frequencies within the dialogue. The data reported come from six different users, each completing at least four editing sessions, resulting in approximately 10 hours of dialogue. The frequency of question types is presented in the following. From Figure 5 it can be seen that the user's main aim in initiating dialogue was to ask for instructions (32/80 enablement questions plus 18/80 specific exploration questions, giving a total of 50/80 or 63%). In general, too, most questions (more than 2/3) provided some kind of context; users' goals and/or their actions and their effects. Indeed, approximately 20 user utterances could not be classified as questions but consisted only of contextual statements. This was sometimes at the request of the expert. These additional context statements and their type are shown in Figure 6.

Despite the high proportion of enablement questions, the expert did not limit himself to instruction, but provided abstract and concrete support,
Questions Asked

![Bar chart showing frequency of questions asked with categories: enablement, evaluation, exploration, elaboration.]

*Figure 5. Questions asked*

Additional Question-Context

![Bar chart showing frequency of additional context with categories: user's action, goal, task state, dialogue, system state.]

*Figure 6. Additional context given with questions*

Together with contextual statements aimed at checking the user's intention. When in doubt, the expert asked the user. Figures 7 and 8 show the aims and methods used in responses and, Figure 9 shows the rhetorical predicates in responses.
Initiate was the most frequent aim; its purpose is to introduce new concepts and employs Instruct and Inform methods. That Initiate and Repair proved to be the most frequent aims suggests the type of help required by users on a first session. Exploratory and Elaborative questions tended to be used towards the end of sessions, when users were beginning to form a model of the possibilities which the different commands suggested, and began to think in terms of combining commands, or about unknown commands needed to accomplish larger tasks.

Overall, Instruct was the most frequently adopted method, reflecting the frequency of enablement questions. Instruct dominates answers to these questions and Inform is the main form of support. Instruct and Inform methods were also used in Repair work, although here, their role was often subordinate to that of the Explain-evaluate method.

The tutor's main goal is to answer the user's question, but in doing so he attempts to describe and explain aspects of the system's functioning. The Inform method (despite the paucity of elaboration questions) was particularly well used. This is also reflected in the frequency and type of rhetorical predicates in responses.

Moreover, from Figure 10 it can be seen that in about half the dialogue episodes, a successful outcome is not achieved after a single exchange, instead the tutor must focus in on the knowledge, needed by the user, over a series of dialogue turns. In about 12% of response instances, it is the expert
A PROTOTYPE QUESTION-ANSWERER

The focus of this article has been the need to develop ways of generating help responses on-line as an essential first step towards a more genuine interaction between systems and their users. The experimental work indicates the need for dialogue-checking moves to resolve ambiguity in users’ goals and in repair work. In this section a question-answering system is described that was formally specified and implemented based upon the experimental findings.

The PORSCHE system (see Tattersall, 1990, in press) incorporates many of the features of expert responding that were identified in the experimental study.
work. The system uses the Eurohelp Application Model (AM), which models the domain of UNIX mail, and was constructed in collaboration with ICL Knowledge Engineering. The Eurohelp Intelligent Help System Shell (IHSS) was designed so that alternative AMs could be slotted into it without requiring a detailed knowledge of its functioning. For this reason, the knowledge representation could not be influenced by text-generation considerations (Reeder, Smith, & Cole, 1989). Thus, in contrast to TEXT (McKeown, 1985), no knowledge is built into the knowledge base to guide the linking of predicates to the information they must express. That the predicates are generic is further supported by the fact that they were developed to describe the experimental domain of UNIX vi but have been successfully applied to the domain of UNIX mail. It is therefore hoped that the predicate-based schemata are suitable for a wide range of information-processing system applications.

PORSCHE uses the Instruct, Inform, and Evaluate answer frames shown in Figure 4. These consist of a conventionally ordered list of predicates that are described further in the Appendix. The method frames can themselves be sequenced (staged) in order to form a response that addresses both the teaching aim and the type of question asked. The final response is a short text of variable length depending on the needs of the user. The process of answer construction is governed by rules to ensure that the ways in which frames are mixed do not violate "normal ordering" or the "cooperative principle."

The PORSCHE formalism adopts object-oriented techniques to combine different information from a variety of sources flexibly. PORSCHE uses the rhetorical predicates described earlier in two ways. First, predicates are divided into content predicates and organizational predicates according to their functional role. Content predicates are responsible for accessing the relational database (AM) to retrieve concepts needed in the answer. Each content predicate has a functional definition representing how it is to find its content.

The AM is made up of three spaces the "task space" which adopts user-centered vocabulary (of a general applicability to information-processing systems), the "system space," which represents the system designer's view and adopts domain-specific vocabulary, and the "interaction space," which links the two. Thus, the system functions used to achieve a user's task are

---

1 The experimental work and much of the implementation of PORSCHE was 50% funded under CEC ESPRIT project P-280, Eurohelp. Partners: Computer Based Learning, University of Leeds, UK; ICL Knowledge Engineering Division (now Decision Support Product Centre), Manchester, UK; Department of Social Science Informatics, University of Amsterdam, The Netherlands.; Courseware Europe BV, The Netherlands.; Dansk Datamatik Centre, Denmark (2 years only); Computer Resources International A/S, Copenhagen, Denmark (now Axion A/S). Eurohelp technical reports are available from: Commission of the European Communities, Rue de la Loi 200, B-1049 Brussels, Belgium.
linked to that task via the interaction space that represents the command sequences required. Behind these spaces is a model of the state changes that result for objects in UNIX mail. The complexity involved in modelling UNIX mail stems from having to separate clearly the effects of commands (what the user thinks of as effects) and effects at the level of system state changes. Thus, to all intents and purposes, if the user types delete <message> then the message has been deleted. However, in terms of system state changes, in fact, all that has happened is that a flag has been set. It is important to model such information, for example, what happens if you want to retrieve the message later? It is equally important to be able to hide such information from the novice. Answers to user's questions typically involve a translation between the task space and the system space and this is why the careful staging of responses is critical. As Tattersall (1990, in press) pointed out, the answer to this problem is not to try to take the AM to the schema (as McKeown, 1985, did in TEXT), by homogenizing knowledge-base relationships and rhetorical predicates at some higher level of abstraction, because this process is idiosyncratic to the particular domain and would reduce the generality of the system. Instead, the scheme steps down to meet the AM (according to Tattersall, in press, without loss of generality. If some loss in explanation "polish," is evident, it is preferable to loss of generality).

The input to PORSCHE is a question type and topic derived from menu and browser selections. Question types correspond to those in the experimental setting and the user may choose from enablement "How do I [topic]?", elaboration "What is [topic]?", and evaluation "What happened [topic]?" For example, if a user asks what is delete? the system is passed (Elaboration C-delete) which is mapped on to the inform frame.

Once the user has received one answer on the named topic, a follow-up question may be asked. The user chooses from Compare-contrast [topic 'topic]? Show more, which expands on each subreferent in the first answer, and By the way also, which describes a new, related command. An image from PORSCHE is shown in Figure 11. The response collection and organization process is shown diagrammatically in Figure 12.

If we take the question, How do I exit mail? then first an answer frame (Inform, Instruct, Explain) is selected to suit the question type. In this case Instruct is selected, because the question in enablement. The activation of Instruct means that the instrument-achievement content predicate searches the AM for an interaction task. The relevant information is placed on the organization board. There are two organization predicates associated with Instruct answers: goal-orientation and setting-trajectory. Goal-orientation retrieves a task from the task space and wraps around the whole structure

---

1 For a fuller description of this image, see Tattersall (in press).
How do I remove messages?

To remove messages in mail mode (where you are now) use:

```
delete (mail specification)
```

A mail specification is an object reference object which refers to a message. It can be used in mail mode. Examples of mail specifications are `message-lists` and `*`

`delete` is a command which has the effect of setting the delete-flag of a message, setting the touched-flag of a message and updating the current message. It takes a mail specification as argument. It is available in mail mode. `delete` can be used by typing `delete message-list`

or

```
delete *
```

What does `dt` do?

`dt` is similar to `delete`. Both are commands, both take a mail specification as argument, both are available in mail mode, both have the effect of setting the delete-flag of a message, both have the effect of setting the touched-flag of a message and both have the effect of updating the current message. However, `dt` also has the effect of displaying a message and `dt` is used to delete a message and read the next messages whereas `delete` is used to remove messages. The syntax of it is

```
dt (mail specification)
dt can be used by typing
dt message-list
or
dt *
```
Figure 12. The PORSCHE question-answering process
giving an initial signal of the task for longer plans. Setting-trajectory wraps around instrument-achievement pairs (and their replacements), to order steps in time with first, then, and next. In this case only a one-step plan is retrieved and so setting-trajectory is not used. "If you want to leave the current application and abandon the changes made, then you should use the exit command by typing exit." This is realized from:

(GOAL ORIENTATION (T leave-abandon-changes SOC-folder/mailbox))
(INSTRUMENT ACHIEVEMENT (HasPlan (T leave-abandon-changes (IT-exit))))

Question-Answerer next consults the user model for the educational aim. After the parsing of an interaction sequence the correct or incorrect use of commands can be updated and used to suggest whether a concept is used, unused, or possibly misunderstood. This can, in turn, be used to update strategy values associated with an answer in PorschE. If exit is not well known to the user then a Consolidate aim is adopted. The method for consolidation is Compare-contrast. Consequently, the differentiating predicates (similarity and difference) are activated and a search is made of the AM for concepts of the same type as exit with the user value strong. An information pool gathers the Inform frame content predicates for both exit and quit.

The similarity predicate then wraps around those attributions that share slot contents whereas the difference predicate wraps around the attribution predicates that remain. Given that the comparison is to be appended to the Instruct part of the response, the replacement propositions are not required because the user has already been told what to type.

(SIMILARITY (ATTRIBUTION (HasEffect C-Exit (SPQuit SOC-Folder/Mailbox))))
(SIMILARITY (ATTRIBUTION (HasEffect C-Quit (SPQuit SOC-Folder/Mailbox))))
(SIMILARITY (ATTRIBUTION (HasEffect C-Exit (P-leave M-mail))))
(SIMILARITY (ATTRIBUTION (HasEffect C-Quit (P-leave M-mail))))
SIMILARITY (ATTRIBUTION (HasPre-Condition C-Exit (M-mail)))
(SIMILARITY (ATTRIBUTION (HasPre-Condition C-Quit (M-mail))))
(DIFFERENCE (ATTRIBUTION (Purpose C-Exit (T-leave-abandon-changes))))
(DIFFERENCE (ATTRIBUTION (Purpose C-Quit (T-leave-save-changes))))
(DIFFERENCE (REPLACEMENT (HasSyntax C-Exit (IT-EXIT))))
(REPLACEMENT (HasSyntax C-Quit (IT-QUIT)))

The comparison is organized by collecting the similarities followed by the differences, beginning with the most generic similarities (effects). The structure would finally pass to the Utterance Generator for lexical selections and the application of grammar and cohesion rules to realize the desired final output:
If you want to leave the current application and abandon the changes made, then you should use the `exit` command by typing `exit`. The `exit` command is like the `quit` command, both have the effect of quitting the folder or mailbox and leaving the mode. However, the `exit` command is used in order to leave the current application, and abandon the changes made whilst `quit` is used to leave the current application saving the changes made. Both are used when in mail mode.

Adopting answer frames containing predicates in a preferred order, the user is led from task space vocabulary towards system vocabulary, ensuring that the necessary cognitive pegs are instantiated before system vocabulary is encountered. However, we noted earlier that the expert mixed and matched his repertoire much more freely than the fixed ordering of the answer frames would seem to imply. This has been argued as a restriction in the schema-based approach. There are two points to make here. The first is that it is not proven that this restriction affects the quality of help (a view compatible with that expressed by Ringle & Halstead-Nussloch, 1989). Context sensitivity and the availability of both user-model and system state information would seem to be much more critical.

Second, because PORSCHE's rhetorical predicates are designed to link clauses (at their finest grain) and cover a range of moods and tenses, it is easily possible to extend PORSCHE's capabilities in this respect, to allow for more dynamic selections or sequences of content predicates. The arguments for when, and how, to do this are less clear. Local features of the dialogue context clearly precipitate such changes in ordering. One variable is, almost certainly, the specificity of the question and another is what the expert thinks the user knows. However, so is changing your mind about what you want to say halfway through answering! The interactions are complex and not yet fully understood.

The preceding examples have shown the range of possible responses that can be produced by combining answer frames together under the control of one of the aims, either Initiate, Consolidate, Remind, or Repair. This process is not unconstrained, because the top-level answer frame will be one that matches the question type, Inform for elaboration, Instruct for enablement, or Explain for evaluation. Follow-up questions of “comparison” and “show more” are supported, and subordinate answer frames in these cases are of the Inform type. The constraints on combining frames within answers are cooperative in the Gricean sense (Grice, 1975). Information is offered so that the answer addresses the question first and moves from the general and abstract toward the specific concrete. The output of PORSCHE is compatible with an utterance generator adapted from McDonald's (1983) MUMBLE. This component ensures predicate labels and their propositional arguments are realized as cohesive text. The coupling of PORSCHE to PENMAN (Penman, 1986) is a further modification, which will improve the output, but does not, in principle, affect the staging of responses.
PORSCHE's significance lies in this ability to stage the content, enabling the generation of answers on-line from a semantic representation. Responses can be given for each of the major types of question asked by users. The staging mechanisms allow answers to remain appropriate in the face of errors, provided that other components within the IHSS can pass the appropriate information. As a component of the Eurohelp shell, the Question-Answerer (Hartley & Pilkington, 1988; Pilkington, 1988, 1992) can also make use of system state change information when generating responses. The planning component of the shell references this information, supplying the content needed for specific enablement and evaluation questions and returns a context-sensitive set of system procedures that forms a plan to change the state of the application. PORSCHE does not demand that other components provide such information, the system can provide an answer without it, but how appropriate the answer can be depends upon the sophistication of emulation and simulation components to formulate efficient suggestions. The flexibility of the response results from the choice of rhetorical predicate-based schema. These schema allow the generation of coherent and genuinely different responses to a varied set of questions, and even allow cooperatively different responses to the same question according to user variables.

The Question-Answerer forming the precursor of PORSCHE was evaluated within the Eurohelp shell, giving some preliminary impressions from six users. Their chief criticism was the system's response time, which caused some users to go ahead without the answer, or plan too far ahead. Users also expressed difficulty in finding relevant topics to ask about in the browser. However, when help was received, it was thought relevant and applied successfully. Some users thought the detailed expression of answers clumsy; this clumsiness reflected deficiencies in the Utterance-Generation component. Since the evaluation, PORSCHE has addressed these problems. PORSCHE responds six times faster than the Eurohelp Question-Answerer and has a much more polished interface and output expression. However, it remains a Question-Answerer and not a dialogue system.

PORSCHE's remaining limitation is an inability to offer an exchange of belief models for diagnostic troubleshooting. The attraction of such dialogue is that it might bypass the problems of user modelling (Self, 1988). Tutors use dialogue to expose the student's mental model of the domain and as a tool for teaching system functionality. AI has not yet produced natural language interfaces of a sophistication to cope with the necessary parsing. At Leeds, dialogue game theory is currently being used to try to address this problem.

TOWARDS SYSTEMS WITH A DIALOGUE CAPABILITY

When dialogue is a medium for learning, interchanges in the form of cooperative dispute serve to uncover inconsistencies (logical or pragmatic),
prompting the learner to examine fresh evidence and construct and test hypotheses. Although dialogue game theory (see, e.g., Carlson, 1983) is not an empirical theory about discourse, but stems from attempts by logicians to characterize what is valid as opposed to fallacious reasoning, nevertheless, as Girle (1986) suggested, it may be possible to exploit its formal properties without forfeiting the advantages of natural language. The idea would be to limit formally permissible forms of expression so that the perlocutionary force of utterances would be clear. This approach is supported by Ringle and Halstead-Nussloch’s (1989) finding that

\[\ldots\text{user input became more tractable to parsing and query analysis as the dialogue style became more formalised, yet the subjective assessment of naturalness and usability remained fairly constant. (p.227)}\]

Moore (1990) conducted a series of studies in which participants engaged in debate, using MacKenzie (1979) style rules to convert the other participants to their view. It seems that game rules can be adopted by participants with little training. As debate proceeds, participants’ commitment stores are added to and subtracted from according to game rules. There are two sets of rules: one governing what must go into or come out of the commitment stores and the other governing when moves may be made. Consider a hypothetical example from an experimental context to illustrate.

**Statements:** The error message says the file has not been written to disk

**Challenges:** Why [The error message says the file has not been written to disk]

**Withdrawals:** I withdraw that [The error message says the file has not been written to disk]

**Resolutions:** Resolve YOUR STATEMENT THAT [The error message says the file has not been written to disk] AND YOUR STATEMENT THAT NOT [The error message says the file has not been written to disk].

Dialogue game rules (based on MacKenzie’s, 1979, DC game), prevent participants from making statements already in the commitment stores of both players, and control when inferences may be withdrawn or challenged. Other game rules control cooperative responding, for example, after a challenge move the other player must withdraw the challenged statement, demand a resolution, or offer a statement in support of the challenged statement. These rules prescribe intention in utterance so that it would be unambiguous to computational systems.

However, dialogue in the intelligent help context is different from that in the strict logical debate: It is cooperative as opposed to combative in style and, there is an imbalance in the relationship between participants. The tutor or expert has relevant knowledge and reasoning skills that he aims to impart to the user or student. To this end, a set of tutoring rules must link to dialogue game rules. Ways are currently being sought to implement a Socratic strategy (as discussed by Collins, 1977) within a DGT framework (Pilkington,
Hartley, Hintze, & Moore, in press). The advantage of the proposed system would be that the interface could adopt menus or similar simple key-press actions, thus avoiding the need for full natural language understanding.

In summary, this article has described a method for analyzing dialogue elicited in the help context. This dialogue was used as an empirical base for the design of PORSCHE, an intelligent Question-Answering module, which is generic enough to be coupled to other Utterance-Generating modules including MUMBLe (McDonald & Pustejovsky, 1985) and PENMAN (Mann, 1983; Matthiessen, 1988) and is capable of using user-model and system-state information to vary its responses if this information is made available. It separates these modifications to the answer in a distinct processing stage so that if this information is not available, it can still produce meaningful and cooperative declarative responses. Experimental work is ongoing and the key to future developments is felt to lie with increasing the interactive properties of the dialogue by allowing both system and user to make statements and to challenge and withdraw statements. This extension will allow for a more genuinely interactive follow-up facility, which would come nearer to emulating the prolonged exchanges required in up to 40% of user questions. The use of dialogue game theory as a means of formalizing this process is being examined.

REFERENCES


**APPENDIX**

The rhetorical predicates are listed with the types of predicate in boldface followed by the type of clause pattern. Actual predicate labels are numbered. Clue words to the named predicate are in brackets.

**Elaborative—Identification**

Matched, generalization-example or preview-detail. Elaborative predicates serve to distinguish between concepts and their classes.

1. **Membership.** For example, "dd is a delete command" defines a concept (is-a, one of).

2. **Attribution, argument/purpose/effect/precondition.** For example:
   - "The yank command takes double quotes" argument has the distinguishing feature (has, takes, form of).
   - "The purpose of the J command is to join two lines" purpose has the distinguishing feature (in order to, use of).
"The effect of using the J command is the joining of two lines" has effect or result as the distinguishing feature (will do, result in, when).

"Before you can use dw you must be in command mode" has precondition as the distinguishing feature (first, before, when).

3. **Constituency.** For example, "Vi is part of the UNIX operating system" is identification by decomposition into subparts (consists of, part).

4. **Representative.** For example, "X is one of the delete commands" is an exemplar following a class description (for example).

5. **Replacement.** For example, "use the yank-to-end-of-line command—"ay$," is a restatement in another more concrete form of specific terms.

**Elaborative—Comparative**

Matched, either compatible or contrasting. Comparative predicates are used to relate a known concept to an unknown or unfamiliar concept or to distinguish between two concepts.

6. **Equivalence.** For example, "< = is the same as BkSp," a difference is arbitrary on the dimension of interest (same as, equivalent to).

7. **Similarity.** For example, "The x command is similar to the backspace command, both delete a character," arguments both have the same attribute (like, as).

8. **Difference.** For example, "The x command is used in command-mode whilst the backspace is used in insert-mode," arguments differ on some critical attribute though otherwise similar (however, unlike, whilst, rather than).

9. **Differentiation.** For example, "Although the x command is similar to the backspace command, both delete a character, they differ in that the x command is used in command-mode, whilst the backspace is used in insert-mode," a predicate operating on both similarity and difference relations.

**Explanatory—Causal**

Logical sequence. Explanatory causal relations explain dependencies.

10. **Evaluation.** For example, "The best way is to use the permanent buffers because if you use the temporary buffers you may lose the copied text," signals a preference based on a causal dependency (best, if . . . then).

11. **Cause-consequence.** For example, "The line was yanked into a buffer because you typed Y," explains result (cause, if . . . then, why).

**Directional—Causal**

Logical sequence. Directional causal predicates link conditions and consequences in a plan or process.
12. **Goal-orientation.** For example, “If you want to delete a line then you should first move into command-mode, then use dd,” an instructional form of evaluation (want/wish to do, should).

13. **Condition-consequence.** For example, “If you move into command-mode then you can use dd,” gives a precondition for another action or an effect (first, when, then).

14. **Instrument-achievement.** For example, “Use dd to delete the line,” gives an action/object necessary to achieve a result (by, use, to do, action-verb).

**Organizational (Independent)**

Undifferentiated. Organizational predicates take other predicates and their propositions as arguments and group or order them in some way.

15. **Collection.** For example, “The 0, a and i commands are all examples of editing commands, the 0 command..., the a command..., the i command..., here three representative statements are collected (all, and).

16. **Additive.** For example, “Moving to the place you want the join and pressing J will join the two lines,” multiple preconditions are joined with and (and).

17. **Alternative.** For example, “There are two ways of moving a line, you can either use the temporary buffers or the permanent buffers,” conjoins a branching choice (either, or).

18. **Adversative.** For example, “X is a delete command not an insert command,” relating what is to what is not the case (but, not, however).

19. **Trajectory-Time-Place.** For example,
   “First delete the unwanted characters then move to where you want the join, next use the J command and then, insert a carriage return” (first, next, then, to where).
   “At the beginning of the paragraph type O, type in the text at the cursor and insert a carriage return at the end of the line” (from...to, at, through, under, over).

20. **Instantiations.** These are statements or questions about the current situation rather than rhetorical predicates. They take as arguments the current goal state, the task state, and the system state and function as premises on which to base an argument, for example: **Instantiate goal-state**, “You want the line moved to...”; **Instantiate task-state**, “You are now in command-mode”; **Comment**, “like we did last time...”; **Elicit goal-state**, “Do you still need to delete the stray characters?”