

## Talking to UNIX in English: An Overview of UC\*

*Robert Wilensky*

Division of Computer Science  
Department of EECS  
University of California, Berkeley  
Berkeley, Ca. 94720

### 1. Introduction

UC (UNIX Consultant) is an intelligent natural language interface that allows naive users to communicate with the UNIX\*\* operating system in ordinary English. UC is currently capable of handling simple dialogues, of which the following is representative:

#### Dialogue 1

User:How can I delete a file?

UC: typing 'rm filename' will remove the file with name filename from your current directory

User:Move a file?

UC: typing 'mv filename newfilename' will move the file with name filename to a file with name newfilename

User:How might a file be created in UNIX?

UC: the best way to create a new file is to use the editor, i.e.- 'ex filename'.

User:A directory?

UC: typing 'mkdir dirname' will create the directory 'dirname' in the current directory.

User:How can I find out how full the disk is?

UC: typing 'df' will tell you how full the disk is.

Users:How should I find out who is on the system?

UC: typing 'who' will tell you who is connected to UNIX.

UC is able to handle most variations of such requests. For example, asking "What is a way to remove a file?" or "How does one delete a file?" will return the same answer as is given above to "How can I delete a file?"

In addition, UC can handle some requests that require greater awareness of the context generated by a dialogue. For example, consider the following dialogue with UC:

\* This research was sponsored in part by the Office of Naval Research under contract N00014-80-C-0732 and the National Science Foundation under Grant MCS79-06543.

\*\* UNIX is trademark of Bell Laboratories

#### Dialogue 2

User:How do I print the file fetch.l on the line printer?

UC: To print the file fetch.l on the line printer type 'lpr fetch.l'.

User:Has the file fetch.l been printed yet?

UC: The file fetch.l is in the line printer queue.

User:How can I cancel it?

UC: To remove the file fetch.l from the line printer queue you must type 'lprm fetch.l'.

In this example, UC determines that 'it' refers to the request to print a file rather than some more recently issued command, because the immediately preceding dialogue has brought this request into focus.

UC is also being extended to handle requests requiring more complex reasoning to formulate an intelligent response.

### 2. The Problem Domain

The problem of naive user communication with an operating system provides an appealing domain for study and application of a number of Artificial Intelligence techniques. The domain is complex enough to provide substantial sub-problems, but not so unbounded that a useful working system must possess a hopelessly large repertoire of knowledge. The task involves a quite realistic use of natural language, namely, participating in a directed dialogue.

In addition, UC provides an application of natural language processing that people would actually be willing to use. Some common uses for natural language processing, for example, natural language front ends to data bases, probably do not have this property. Once a user has learned a somewhat more cryptic query language, he is apt to prefer it over natural language as it is likely to provide a simpler and less verbose mode of expression for the majority of queries.

In contrast, a naive user of a system would probably be willing to converse in English with a machine, since the alternatives are apt to be worse. Manuals and on-line help facilities are only marginally useful - a user usually needs to be fairly sophisticated in order to issue the right help command, and then these are successful only a fraction of the time. People knowledgeable enough to be helpful are not

always available and usually do not have the time to offer.

In addition, the domain is "soft" in the sense that a system that did not work all the time is still likely to be useful. For example, if UC failed to give the desired answer, the user is no worse off than he would be otherwise, and still has the normal repertoire of options at his disposal. Of course, if UC does provide a useful answer, the user will benefit from it. Thus UC is an AI system that will be useful even as it expands to a larger portion of its domain.

Lastly, the problem integrates a number of areas of concern for artificial intelligence, including natural language understanding, natural language production, planning and problem solving.

### 3. The Structure of UC

UC runs on a VAX11/780 and is programmed in FRANZ LISP and in PEARL, an AI language developed at Berkeley (Deering et al., 1981). Although UC is itself a new system, it is built out of a number of components, some of which have been under development at Berkeley for some time. The natural language understanding component consists of an extended version of PHRAN (PHRasal Analyzer), original programmed by Yigal Arens on a DEC10 and moved to the VAX by Margaret Butler. The natural language production is to be done by PHRED (PHRasal English Diction), PHRAN's sibling language processing program, but as of this writing PHRED has not been moved to the VAX. These programs are described in Wilensky (1981b).

PHRAN reads sentences in English and produces representations that denote their meanings. One of the primary design goals of PHRAN is that it be easily extensible to new linguistic forms and domains. This is particularly important for implementing UC, as we expect to be continually extending the system's scope. Applying PHRAN to the domain of requests about UNIX was as much a test of PHRAN's versatility as it was a useful application. In fact, most of the forms used by UC are new to PHRAN, and were added to PHRAN's knowledge base of English by David Chin. The approximate amount of effort to extend PHRAN to handle one of these new forms, once the representation used to denote the meaning of the form has been agreed upon, is currently on the order of a few minutes. Right now, the PHRAN component of UC can understand requests about 25 substantially different topics, each in many different linguistic formats.

One of the extensions to PHRAN being realized in UC is a mechanism to handle some forms of ellipsis, implemented by Lisa Rau. This mechanism handles both intra-sentential forms, such as "I want to delete the small file and the large", and inter-sentence forms, as in "How do I delete a file? A directory?" Ellipsis is handled by first letting the basic PHRAN analysis mechanism produce what it can from the input. PHRAN is a pattern-based system whose patterns map various levels of linguistic structures into meaning representations. Thus the process of understanding an utterance leaves a history of the patterns used to arrive at that understanding. If the result of this process is incomplete (e. g., generally something that is not a sentence where a sentence is expected),

then previously used PHRAN patterns are examined to see if they match the patterns used to understand the fragment. If so, then the words of the fragment are substituted for the words of the previous sentence that correspond to the common pattern. The resulting linguistic unit is then re-analyzed by PHRAN.

#### 3.1. Context in UC

Another extension to PHRAN involves the implementation of a context mechanism by Yigal Arens. This is essentially an activation-based memory mechanism used by UC to keep track of the focus of the discourse (cf. Grosz 1977), handle some aspects of reference, and help in word disambiguation. When a concept is encountered by PHRAN, it is given a level of activation; concepts stored in memory as being associated with this concept are also given a level of activation. As processing proceeds, the activation levels are manipulated according to an algorithm described in Arens (1982).

For example, in Dialogue 2 above, the question preceding the cancellation command brings back into focus the previous request to print the file. That is, asking this question increases the activation level of the memory of that request. Then when the last request is issued, UC will try to interpret 'it' in this new context. Since the request to print the file is now more highly activated than any intervening request, UC will determine that 'it' refers to the print request even though this request is not the most recent.

#### 3.2. Processing Requests in UC

After a meaning representation for a sentence has been produced by PHRAN, a reasoning process comes into play. For the simplest requests, UC simply uses the meaning representation of the request as a query into an associative knowledge base of facts to see if there is a stored answer. This knowledge base, and in fact, much of UC, is implemented in PEARL (Package for Efficient Access to Representations in LISP). PEARL is an AI language developed with flexibility and time and space efficiency in mind. By using PEARL, it is possible to assert new facts into the knowledge base and have UC use these facts quickly in its various response formation processes. PEARL is described in more detail in Deering, Faletti, and Wilensky (1981, 1982).

For example, the following is a simplified version of the PEARL representation of the fact that using the 'rm' command is a way to delete a file:

```
(planfor (concept
  (causation (antecedent (do (actor ?X)))
    (consequent
      (state-change (actor ?F/is-file)
        (state-name physical-state)
          (from 10)
          (to -10))))))
  (is (use (actor ?X) (command (name rm) (arg ?F))))))
```

Questions of the form "How can I delete a file?" are analyzed into a form similar to this, but with the "is" slot filled with a variable. PEARL's retrieval facility finds the stored fact based on this similar structure; it can then be used to supply an answer (right now, canned text associated with these representations is used because the PHRED generator does not yet work on the VAX).

#### 4. Extending UC to Process More Complex Requests

Most requests require more complex processing, however. For these situations, UC uses a reasoning component based on the PANDORA planning mechanism (Wilensky 1981a). PANDORA, implemented by Joe Faletti, is based on a model of planning in which goal detection and goal interactions play a prominent role. For example, consider the indirect request: "I need some more disk space." A literal response to this remark might be "Type 'rm \*'", which is most likely not what the user had in mind.

The problem with this response, of course, is that it violates an unstated user goal, namely, that the user wants to preserve what he has already put on the disk. An intelligent consultant must be able to infer such goals, and reason about the interactions of such goals with those explicit in the user's request. In this example, an implicit goal (preserving a file) may conflict with the stated goal (getting some more space), and this possibility must be explored and dealt with.

Although it was originally constructed to be an autonomous planner, PANDORA's architecture is well suited for this sort of reasoning. PANDORA first tries to apply a stored plan to a given goal. It then simulates the situation that may result from the current state of the world using a mechanism called a *Projector*. In the above example, the simulation will reveal, among other things, that some files will get destroyed, as this is a consequence of the 'rm' command.

Another of PANDORA's basic components is called the *Goal Detector*. This mechanism determines the goals the planner should have in a given situation. The goal detector is essentially a collection of demons that respond to changes in the environment, including the simulated environment created by the projector. In this example, when the simulated future reveals the possible destruction of a file, the goal detector will react to this possibility by inferring the goal of preserving this file.

Since this preservation goal arises from a plan of the user's, PANDORA also infers that there may be a goal conflict between this goal and the goal underlying the user's original request. PANDORA makes this inference by considering a goal giving rise to a preservation goal as another situation in which to detect a goal (namely, the goal of resolving the goal conflict). Then a plan for this "resolve-goal-conflict" goal can be sought by successive application of the whole planning process.

This algorithm makes use of a *meta-planning* representation for planning strategies. The goal of resolving a goal conflict is actually a meta-goal, that

is, a goal whose successful execution will result in a better plan for other goals. This formulation allows the goal detection mechanism to be used to solve the problem of goal conflict detection, and the normal planning process to find a resolution to such a problem. More detail on meta-planning and the associated algorithms is given in Wilensky (1981a).

In the example at hand, the presence of a goal conflict is only a possibility, as the user may well have some files that he doesn't need. A general strategy in such situations is to determine whether the possibility actually exists. This would lead to the generation of the question "Do you have any files that you do not need?" If the user's response is negative, then the conflict does in fact exist, and a conflict resolution strategy must be employed.

A strategy that is applicable to all conflicts based on a shortage of resources is to try obtaining more of the scarce resource. In the example above, the scarce resource is disk space. PANDORA would then create a new goal of obtaining more disk space. Since a stored plan for this goal is to ask the systems manager for more room, UC can respond with the advice that the user request more room from the systems manager. An implementation of PANDORA and a fuller explanation of its role in UC is found in Faletti (1982).

Of course, it is possible to store requesting more room as a plan for the original goal, and by-pass all this complex reasoning. The problem with such a solution is that it would preclude the possibility of informing the user that he could delete a file that he doesn't need. Alternatively, we could make a more complex canned plan that checks for just this situation. That is, the stored plan would be to delete a file one doesn't want, and, if this fails, then to request more space. The problem here is that it would be necessary to include all possible interactions in the canned plan. While it may be desirable to include some of these some of the time, to deal with unanticipated interactions, a more general reasoning mechanism is required. For example, if the systems manager is unavailable and the user's desire for the disk space is great enough, deleting a file may still be a reasonable solution; however, it is unlikely that such a possibility would be anticipated by the canned plan.

#### References

- Arens, Y. 1982. The Context Model: Language Understanding in Context. In the *Proceedings of the Fourth Annual Conference of the Cognitive Science Society*. Ann Arbor, MI. August 1982.
- Deering, M., Faletti, J., and Wilensky, R. 1981. PEARL: An Efficient Language for Artificial Intelligence Programming. In the *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*. Vancouver, British Columbia. August, 1981.

Deering, M., Faletti, J., and Wilensky, R. 1982. The PEARL Users Manual. Berkeley Electronic Research Laboratory Memorandum No. UCB/ERL/M82/19. March, 1982.

Faletti, J. 1982. PANDORA -- A Program for Doing Commonsense Planning in Complex Situations. In the *Proceedings of the National Conference on Artificial Intelligence*. Pittsburgh, PA. August, 1982.

Grosz, B. J. 1977. The Representation and Use of Focus in a System for Understanding Dialogs. *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*. Carnegie-Mellon University, Pittsburgh, PA.

Wilensky, R. 1981a. Meta-planning: Representing and using knowledge about planning in problem solving and natural language understanding. *Cognitive Science*, Vol. 5, No. 3. 1981.

Wilensky, R. 1981(b). A Knowledge-based Approach to Natural Language Processing: A Progress Report. In the *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*. Vancouver, British Columbia. August, 1981.