Natural-language help in the Consul system

by WILLIAM MARK
USC/Information Sciences Institute
Marina del Rey, California

ABSTRACT

If we use the model of asking an expert, it is fairly clear what users want from a help system: a service that tells them how to do something they want to do. But current help systems aren't like this. They can tell the users about system capabilities, but not in relation to what they want to do. Most help systems are simply databases of online documentation—system manuals, not system experts. Like system manuals, using them to figure out how to do something or to figure out what went wrong is a last resort—when no one else is around.

Providing real expert help requires reasoning in terms of models of what the user wants to do and what the system can do. These models also make it possible to provide facilities for natural-language understanding and generation. Thus, users can deal with the system in much the same way that they deal with a human expert: by asking questions and receiving advice in English.

These ideas are being tested in the Consul system, a research prototype currently under development at the USC Information Sciences Institute.
1. WHAT IS HELP?

Users of an interactive system need help whenever they encounter an obstacle that prevents them from performing a task. They may need to find out how to do something (How do I get rid of the messages Smith sent me yesterday?), how to describe an object (What has to be in a message?), or why something unexpected occurred (What happened to the message that was on my screen?). This need for help may be expressed by the system rather than by users, as when a user runs into an error condition (You can't forward a message you have composed). The point is that the need for help is a feature of users' conceptual framework: it expresses the difference between their state of knowledge and their expectation of how to specify a task to be performed by the system.

Users' conceptual framework may be very different from the system's. Even though the question How do I get rid of the messages Smith sent me yesterday? makes perfect sense to a user, it may not relate directly to anything in a mail system whose "messages" always reside in a central database and are never sent to (and can never be deleted by) users (e.g., [SIGMA 79]). The user's problem must be mapped into the system framework before it can be solved. Then, since the user may not understand the solution in system terms, it must be mapped back into an answer in terms of the user's framework.

2. CURRENT HELP MECHANISMS

Let's contrast how this is done in the two currently existing interactive help mechanisms: asking a system expert and using an online help system. When a user asks a system expert for help, the expert mentally translates the user problem into the system framework, decides what the user must do, and then translates back into the user's terms in order to explain it to the user. Given the question How do I get rid of the messages Smith sent me yesterday? the expert (1) realizes that the user actually wants to delete the citations (records) of a particular set of messages from the mailbox; (2) knows that the system has a command that can filter citations in mailboxes by sender and date and a command for deleting lists of message citations; and (3) tells the user how to invoke the two commands in sequence to achieve the desired result.

Current online help systems operate in a very different manner (for a discussion, see Relles and Price). They contain documentation of system features (which can be thought of as precomputed mappings from the system framework to the user framework), usually indexed in terms of what the system can do, not what the user can do with the system. In other words, the user is responsible for translating need for help into retrieval requests (or menu selections) in order to access the system's documentation database. This often makes it difficult for the user to get to the information wanted (he/she might type "? message" and get a lot of irrelevant information; he/she might have to wade through a list of system commands and guess that "delete" is the one wanted; and so forth). Once the user finds a relevant piece of documentation, he/she may not understand how it relates to the problem (he/she may discover that the delete command takes "a list of messages" and not realize that filters can be used to produce that list).

This points up two shortcomings of current online help systems:

- No explicit representation of the user's framework (the user can't express the problem, but can only search for possibly relevant documentation).
- No flexibility in mapping questions into answers (even if the user could express the problem, it may cut across the precomputed mappings stored in the help database).

There is evidence that these shortcomings are sufficient to discourage most users from using online help at all.

In the Consul system we are trying to overcome these problems by building knowledge into the system—models of what the user wants to do and what the system can do—and providing help by reasoning in terms of these models. The goal is to allow the user to be able to deal with the system in much the same way he/she deals with a human expert: by asking questions and receiving advice in English.

3. HELP IN THE CONSUL SYSTEM

The Consul system is based on a representation of user and system knowledge in a central knowledge base (see Figure 1). This knowledge includes a model of what users want to do with interactive systems, a model of what interactive systems can do, and a set of mapping rules for translating between the two frameworks. When a particular interactive service like a mail system is added to Consul, the knowledge base (including the mapping rules) is particularized to take into account the distinctive features of the service (this process is discussed in the next section).
The system's activity consists of mapping descriptions in user terminology into system terminology and, in the case of help requests, back into user terminology. The process begins with a request from the user, expressed in natural language. The request is parsed and represented as user terminology in Consul's knowledge base. It is then mapped into system terminology, allowing Consul to determine whether it is a request for help or a system command. If it is a request for help, the appropriate information about system features (in system terminology) is isolated. This information is then mapped back into user terminology, and finally into English to provide the answer. If a request that Consul originally interprets as a system command cannot be executed, it is automatically reinterpreted as a request for help. This process can be illustrated in terms of the four help interactions mentioned in the first section:

How do I get rid of the messages Smith sent me yesterday?

As mentioned above, responding to this request requires mapping of the notion of messages in the user framework into message citations in the system framework. Consul produces this mapping by first finding that its model of the mail system does not allow the user to delete messages directly, thus preventing the request from being taken at face value. It then finds that it has a mapping rule that translates user statements about doing things to objects into statements about doing things to summaries of those objects (reflecting the fact that users often say things like "show me a list of my files" when they really want a directory listing, a list of summary information about files). This mapping rule translates the original request into a request to delete a list of message summaries. This can then be mapped into a sequence of the mail system's actual operations for filtering and deleting message citations. Examination of these operations reveals that they (collectively) require a user mailbox and a set of filters—in this case, a sender and a date—as arguments. This information is in fact the answer to the user's question, but it must be mapped into user terminology before actually being displayed to the user. The user terminology in the knowledge base is checked to see if there are constructs corresponding to those found in the system framework. In this case there are, and an English response is generated: "Lists of messages for deletion can be specified by any combination of selectors such as sender, range of dates, and message numbers, as in 'Delete the messages from Smith I received yesterday.'"

What has to be in a message? In this example, the term message can be taken at face value. That is, message in the user world is taken to refer to message in the mail system world as well. The mail system's representation of a message is therefore examined for required fields, which are collected, and, as before, mapped back into corresponding user terms to provide a response to the user: "Messages must have a valid addressee and usually have a subject and body of text."

What happened to the message that was on my screen? This request refers to an event (something "happening" to a particular message): Consul must therefore examine its history records. It first determines which message was most recently displayed on the user's screen, then looks for all of the events that involved that message since it was displayed. Many things could have "happened" to that message since it was displayed (e.g., other users could have received pointers to it in the central database), but few of these events would have been noticed by the user. The system must compare each event involving that particular displayed message with the events of the user world. Those that have significance in the user world are collected, mapped into their corresponding user terminology, and presented in the response: "I had to take it off the screen temporarily to show the list of messages you requested."

You can't forward a message you have composed. This help interaction is not requested by the user; it is generated by the system in response to a user request to forward a particular message (e.g., Forward this message to Jones.). Initially, the request is mapped into system terms, and Consul recognizes it as a command for system action. Let us assume that the message involved is one that the user has just composed, not one that he/she has received. The mail system will not forward composed messages. This is reflected in the Consul system by an inability to map the supposed command into an actual mail system execution sequence. Consul therefore knows that it must generate a help response. It compares the request (as mapped into system terms) with the forward function the user was trying to execute, thus finding the differences between the request as stated and the required form of the arguments of forward. These differences are presumably what prevented the command from being invoked in the first place. In this case, Consul discovers that the forwarding function in the mail system requires a "transmitted" message, while the message in the request is of type "draft." It can map this difference back into user terminology to generate the response shown above.

But in many cases (including this one), Consul can go further in responding to errors. If Consul can find system functions similar in intent to the one the user was trying to execute, it can use them as targets and find differences just as before. This allows it to suggest alternatives that accomplish the user's goal. In this case, the system discovers that send as well as forward could accomplish the user's presumed intent of getting a particular message to a particular user. Therefore, when the user makes his/her erroneous forwarding request, Consul actually generates the more complete help response, "You can't forward a message you have composed. You can send it to Jones instead."

4. ACQUIRING AND MAINTAINING THE HELP INFORMATION

To provide all this knowledge-based help, a lot of specific information about mail systems and how they are used must somehow be put into the machine. In fact, a major issue for all interactive help systems is how the necessary help information gets into the machine and how it is kept up to date in the face of system changes. The approach taken in the Consul system is to build in a general model of interactive services and how they are used, including a model of how users specify commands and ask for help. This general model is then used to solicit specific help information from the programmer of each actual service.

A programmer of a new service (e.g., the mail system) enters each program into Consul through a dialogue-oriented acquisition aid. The acquirer uses Consul's model of inter-
active systems to ask the programmer how a program fits into the system knowledge base. A dialogue ensues as the acquirer asks the programmer questions in terms of its model and the programmer answers in terms of the program. For example, in acquiring the “forward” program, the acquirer will ask what in the program corresponds to the system model of the “destination” of a forwarding operation. The programmer will reply that it is the “mailbox” whose owner is the user specified as the “receiver” argument of the operation. The acquirer then checks to make sure that mailboxes are legal destinations according to its model, that they can be owned by users, and so on, until it is sure that the programmer’s version of “destination” fits the model in the Consul knowledge base. If it does not, the acquirer pursues the dialogue until it discovers how “mailbox” relates to something that is legal in its model. In this way, Consul comes to understand each new program in terms of its knowledge base. The acquirer will not allow the program to be part of the Consul system until it sees how it fits into both the system and the user framework. Fitting into the user framework usually requires the construction of mapping rules, also done automatically during the acquisition dialogue.

Once acquisition is accomplished, Consul has all the information it needs to construct the necessary help responses. Since it knows about message forwarding in general—what it does, how users ask about it, what can go wrong with their requests—and it has learned how the particular forwarding operation of the new mail system fits into this scheme, it knows how to handle user requests for help and how to generate comprehensible error explanations. Moreover, every time a program or data structure is changed, the acquirer is automatically reinvoked, thus insuring that the knowledge base is always up to date.

5. CONCLUSIONS

The Consul system is an experiment in providing a natural-language interface, including natural language help, to users of interactive systems. It currently handles only a small part of a single interactive service—the mail system described in this paper. Much work remains to demonstrate its feasibility in cost and execution speed in a real working environment consisting of varied users and services.

ACKNOWLEDGMENT

This research is supported by the Defense Advanced Research Projects Agency under Contract No. DAHC15 72 C 0308, ARPA Order No. 2223. Views and conclusions contained in this paper are the author’s and should not be interpreted as representing the official opinion or policy of DARPA, the U.S. Government or any person or agency connected with them.

REFERENCES
