Uniform help facilities for a cooperative user interface

by PHILIP J. HAYES
Carnegie-Mellon University
Pittsburgh, Pennsylvania

ABSTRACT

This paper describes the design of the help and explanation component of a user-friendly operating system command interface called COUSIN. The facility can provide two kinds of information: (1) static descriptions of the various subsystems that can be invoked, their parameters, and the syntax that must be used; (2) dynamically generated descriptions of the state of the current interaction, why that state has arisen, and what the user's options for action are. Both types of information are presented in the same way through a network of small text frames connected by semantically motivated links in the style of the ZOG system. Frames containing static information are generated automatically for each subsystem from a declarative description of the subsystem which is also used by COUSIN for its other services, including spelling and grammar correction and interactive error resolution. Dynamically generated frames are incorporated temporarily into the static network with semantic links appropriate to the current command context.
INTRODUCTION

The COUSIN (COoperative USer INterface) project at Carnegie-Mellon University is engaged in a wide-ranging program of research\(^1\)\(^2\)\(^3\)\(^4\)\(^5\) aimed at producing interactive command interfaces that appear to their users more friendly and cooperative than most presently available interfaces and are thus less frustrating and more productive to use. This research includes work on flexible, error-correcting parsing, interactive error resolution, and the topic of this paper: interactive help facilities. The following paper presents the design and rationale for the help facility that we are currently implementing for use with the COUSIN interface.

The COUSIN help facility provides explanations of two kinds:

- **Static explanations**: Descriptions of the aspects of the system that do not normally change during the course of a terminal session, such as what subsystems are available through the interface, what are their parameters, what is their syntax, what is their purpose, and similar aspects.

- **Dynamic explanations**: Information dependent on the immediate context of the request, such as what the user's current options for action are, what is the last command performed, what the interface expects the user to do next, how the interaction came to be in its present state, and related information.

COUSIN distinguishes these types of explanation because of the way they are generated: the static explanations are pre-stored and displayed on demand, whereas the dynamic explanations must be generated on the fly. However, since this distinction is not likely to be of great concern to the user, the design of COUSIN is that the two types of explanation be presented in as uniform and integrated a way as possible. Integration is particularly important if a dynamic explanation (e.g., what commands are currently available) leads the user to request a static explanation (e.g., the syntax for one of the available commands).

By far the most common way to provide online help is to use canned text messages. Such messages are either written into the system by the system designer specifically for interactive use, as in the SOS editor,\(^6\) or extracted by an indexing scheme from an online version of the system manual, as in the RdMail electronic mail system\(^7\) or the CMULisp system.\(^8\) This approach is susceptible to two problems: first, the blocks of canned text may be too long and ill-structured, so that the user must search through irrelevant material to get to the information actually needed; and second, the blocks of text may be insufficiently linked or cross-referenced, so that the user may be unable to locate the needed information even if he has found a related piece of information.\(^9\) To avoid these problems, a second design goal for the COUSIN help facility is to present information in fine-grained chunks that are heavily interlinked and cross-referenced.

The help and explanation facility for COUSIN satisfies the two design goals mentioned above by following an approach similar to that of the ZOG menu-selection system,\(^10\) also developed at CMU. ZOG allows its user to traverse a network of small (one-display terminal screen) text frames linked together in an arbitrarily complex way by semantically motivated links. Each frame consists of a text part and a menu of links (whose descriptions themselves often convey information). Selecting one of the links causes the display of the frame pointed to by the link. COUSIN's help and explanation facility consists of a similar network of text frames connected by mnemonically named semantic links; the frames are of variable size, but usually contain less than a screenful of information. This arrangement clearly satisfies the requirement that the help information be available in small-grained chunks and allows for a set of interconnections rich enough to make information related to that of the current frame easy to obtain. Moreover, it provides a convenient way of integrating static and dynamic help. The static help can be represented as a pre-stored frame network, and the dynamic help can be expressed through frames that are constructed and linked into this static network on the fly. This makes the presentation of static and dynamic help uniform and allows the dynamic explanations to provide convenient access to related parts of the static network.

A further advantage of the ZOG approach for COUSIN is that the static explanation net can be generated automatically from information already required by COUSIN for other graceful interface functions. Briefly, COUSIN requires a declarative description of each subsystem with which it is used. These **subsystem descriptions** contain, among other things, details of the subsystem's parameters, their types, whether they are optional, and the syntax used to express them. Using this information, COUSIN can accept commands from the user, check them for validity, fill in defaults, correct some errors and ambiguities, interact with the user to correct the others, and finally transmit the corrected command to the underlying system. More important for present purposes, COUSIN can use the same information, supplemented by text strings to explain the purpose of subsystems and their parameters, to construct a fine-grained and highly interconnected static explanation network for the subsystems concerned. The following sections discuss the construction and structure of this static explanation network and examine how dynamic explanation frames can be generated and linked into it on the fly.
AUTOMATIC CONSTRUCTION OF STATIC EXPLANATION NETWORKS

This section describes the structure of the static help offered by the COUSIN interface; i.e., the explanations COUSIN can give about the subsystems it provides an interface to, their invocation syntax, their parameter types and structure, and related matters. We also describe how these static explanations are constructed automatically from the declarative subsystem description databases from which COUSIN also obtains the information it needs to provide its input checking and correction services. Generating the static help frames automatically enables the subsystem designer to provide a structured set of online documentation for the subsystem with little incremental effort. The designer needs to construct a subsystem description in any case to use COUSIN's other services. To produce the help documentation, he merely needs to add some text fields to the subsystem description to indicate the purpose of the subsystem and its various parameters, and the rest can be done by COUSIN. We believe that the effort involved in producing documentation in this form is far less than that involved in producing documentation with the same content by hand. Moreover, the resulting documentation will have a considerably more uniform structure, both within a single subsystem and, more important, across a wide range of subsystems. Glasner and Hayes 11 give details of the way this approach was employed to produce online documentation for an ancestor of the present COUSIN interface, and Fenchel 12 has used similar techniques to generate user-oriented descriptions automatically from a parser's grammar.

Before examining the form of the explanations COUSIN generates, we should first look at the structure of the subsystem descriptions from which they are generated. One of the applications of COUSIN is to provide a command-level interface to the UNIX operating system on a VAX 11/780; i.e., to provide an alternative to the standard Unix shell. As described above, COUSIN requires each subsystem, in this case each Unix command, to have a declarative subsystem description. Figure 1 shows an abbreviated subsystem description (there are actually many more optional parameters) for the “cz” command, used in our department to print files on the Dover, a local hard-copy output device.

The details of this notation are not important for present purposes. It is enough to note that the “Schema” indicator introduces a property list whose indicators (“alignment,” “copies,” etc.) name the parameters, optional and required, of cz, and whose corresponding values give more detailed information about the types, defaults, and related matters concerning those parameters. As mentioned previously, this information, together with the “Syntax” property, is used by COUSIN to provide input checking and correction on the syntax and parameter values of invocations of cz.

The same information in conjunction with the “Explanation,” “Description,” and “Summary” fields can also be used to generate a tree-structured net of explanation frames describing cz at varying levels of detail, corresponding to different levels of the tree. The root of the tree, for instance, is a brief description of the purpose and calling syntax of cz, shown in Figure 2.

If, after reading this text frame, the user decides more information is needed, he can select one of the links, which will result in the display of another frame. The “Unix syntax” link leads to a manually constructed frame explaining the general style of Unix syntax. The tobeprinted link leads to the frame shown in Figure 3. From this frame, the user can obtain the general syntax for files (from another hand-constructed frame), if that is what he needs. Taking the “options” link from the root frame would lead to a list of options, as shown in Figure 4.

Taking the “rotated” link from this frame would lead to the frame shown in Figure 5.
From the collection of the Computer History Museum (www.computerhistory.org)

DYNAMICALLY GENERATED EXPLANATIONS

Along with the specific links mentioned above, each frame has links that allow the user to return to the previous frame, go to the root of the present tree of frames, and obtain explanations of the notation being used or of the organization of the help system in general. There is no space here to give details of the program used to generate the static help frames shown above, but comparison between the subsystem description and the frames should convince the reader that only a relatively straightforward rearrangement of the information in the subsystem description is necessary.

A complete static help facility for COUSIN is obtained by applying the frame-generation process to the subsystem descriptions of each subsystem that can be invoked through COUSIN. A set of trees of text frames results, with the tree nodes linked together in ways illustrated previously. The static help network is completed by hand-generated auxiliary frames that provide general information about syntactic conventions and about the organization and notation of the help system itself.

The user obtains the display of a static help frame by giving the name of the frame as a parameter to the help command. The name of the root frame for a subsystem is the same as the name of the subsystem; therefore “help cz” will result in the display of the first frame shown above. Frames below the roots of the trees are named incrementally according to the links that should be followed to get to them; thus the last frame shown above is called cz.options.alignment and may be accessed directly by this name as well as by following links from-the cz base frame. Of course, when frames go out of a subsystem tree, this results in a frame having more than one name; thus cz.tobeprinted.file is also called file.syntax.

Besides the network of static help frames, the COUSIN help facility can also display dynamically generated help frames to give the user contextually dependent help. The kinds of questions these frames are designed to answer include the following: What is the current state of the interaction, what can I do now, what does the system expect me to do, how did the interaction get into this state? The dynamically generated frames are of exactly the same form as the static frames and will normally have links to the static network. Static and dynamic explanations thus appear uniform and integrated to the user.

Currently, dynamic frames are generated in only one kind of situation: when the user makes a request for help without giving the name of a static help frame. When this happens, COUSIN constructs a frame containing the following information:

1. The current state or mode of the interaction.
2. Why the interaction is in this state.
3. A list of actions the user is likely to want to take.
4. Links to other frames that allow the user to find out about the complete range of options available.
5. Links to frames describing contextually relevant subsystems and aspects of subsystems.

A frame of this sort is put together out of canned text templates, one for each of the various components; variables in the template are filled in by contextually appropriate tokens. An example will make this clearer.

Suppose the user gives the command

cz -c 2 rotated foo.press

to the main command level of COUSIN. The command is to print two copies of foo.press in 90-degree rotated mode. Suppose also that the user does not have read permission on foo.press. COUSIN prints out the message:

no read permission for foo.press
envedit

where the second line is a prompt for the environment editor.

For present purposes, we can regard an environment as a set of parameter/value pairs for a subsystem invocation. When an error in a command line is detected, COUSIN makes up an environment from the command, then enters the environment editor to allow the user to alter the problematic parameters without having to retype the rest. Now suppose that the user in this case is unsure of what is happening and so makes a nonspecific request for help by typing “?” or “help.” Following the recipe listed above, COUSIN then constructs the help frame shown in Figure 6. Note how the LINKS section gives more detail about the command the user is most likely to want to use, as well as pointers to the other commands available.

The other links are to relevant frames in the static network.

You are in environment editing mode through which you can change the value of slots in the current environment, which was constructed from:

cz -c 2 rotated foo.press

You are in this mode because "foo.press" does not satisfy the requirement that the "tobeprinted" parameter of cz be a readable file.

LINKS 1. correct - environment editor command to cycle through and change incorrect slots
2. other environment editing commands
3. environments - general information
4. cz
5. tobeprinted parameter of cz
6. files

Figure 6—A dynamically constructed help frame
CONCLUSION

This paper has described the design of the help and explanation facility that we are currently implementing for the COUSIN operating system command interface. COUSIN will provide online help and explanations through display of text frames connected in a network by semantically motivated links in the style of the ZOG\(^\text{10}\) system. Most of the frames containing information that does not normally change over the course of a terminal session can be generated automatically from declarative representations of the various subsystems to which COUSIN interfaces. These declarative subsystem descriptions are necessary for other aspects of COUSIN's operation but must be supplemented by a number of text fields to provide good help frames. In addition to this static help network, COUSIN will provide dynamically generated, contextually sensitive explanations about the current state of the interaction. These dynamic explanations can be assembled out of predefined templates into the same text frame form as the static help and temporarily linked into the static network so that the help system as a whole appears consistent and uniform to the user.

ACKNOWLEDGMENTS

A program written by Pedro Szekeley demonstrated the feasibility of constructing the static help network automatically from subsystem descriptions.

This research was sponsored by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory under Contract F33615-78-C-1551. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. government.

REFERENCES

6. Carnegie-Mellon University, Computer Science Department. Son of Stop-Gap (SOS). Originally developed at Stanford AI Lab; the help facility was added at CMU. 1978.