Exploiting User Expertise in Answer Expression*

David N. Chin
Department of Information and Computer Sciences
University of Hawaii at Manoa
2565 The Mall
Honolulu, HI 96822

Abstract

Previous natural language help systems have not taken into account the user's knowledge when formulating answers. Such pragmatic information is needed to formulate more concise and helpful answers. By not repeating things that the user already knows, a system can provide more succinct answers that, because they focus on pertinent new facts, are easier to understand. A users's prior knowledge also allows a system to utilize special teaching formats such as similes. This process of refining answers using pragmatic information is called *answer expression*. It has been implemented in the UCExpress component of UC (UNIX Consultant), a natural language system that helps users solve problems in using UNIX. UCExpress separates answer expression into two phases: *pruning* and *formatting*. During pruning, subconcepts of the answer are marked as not needing generation when they are already known by the user, or marked as candidates for generating anaphora or ellipsis when they are part of the conversational context. During formatting, UCExpress uses information about the user's prior domain knowledge to select among specialized *expository formats*, such as similes and examples, for expressing information to the user. These formats allow UCExpress to present different types of information clearly and concisely.

1. Introduction

Previous natural language help systems have not modeled their user's domain knowledge, so they have not been able to take advantage of this pragmatic information when formulating answers. The first help system to use such information is UC (UNIX Consultant) [Wilensky, Arens, Chin, 1984] [Wilensky et al., 1986], a natural language consultation program that answers the user's questions on how to use the UNIX[1] operating system. After other components of UC have identified an answer to the user, the UCExpress component is called to express this answer to the user. In deciding how to do this, UCExpress must take into account the user's prior domain knowledge. To see why this is important, consider the actual UC sessions shown in Figures 1 and 2.

In session 1, UC gives an example of how to use the rm command whereas in session 2, UC does not give an example of how to use ls -i. This is because in the first session, the user is a novice, so UC cannot assume that the user knows the format of the rm command. However, in session 2, the user is an intermediate, so UC can assume that the user would know how to use ls -i. Also, in session 2, UC uses a simile to explain what ruptime does in terms of what uptime does. This simile is shorter and clearer than the full answer given by UC in session 1. However, this simile is only useful if the user already knows what uptime does. UC can assume this for the intermediate user of session 2, but cannot do so for the novice user of session 1.

```
# How can I delete a file?
Use rm.
For example, to delete the file named foo,
type 'rm foo'.
# What does ruptime do?
ruptime is used to list the uptime of all machines on
the network, list the number of all users on them and
list their load average.
```

Figure 1. UC session 1 with a novice user.

```
# How can I find out the inode of a file?
Use ls -i.
# What does ruptime do?
ruptime is like uptime, except ruptime is for all
machines on the network.
```

Figure 2. UC session 2 with an intermediate user.

Differences in answers such as those shown in Figures 1 and 2 can only be achieved through the interaction of answer expression and user modeling. This paper will show how UCExpress is able to exploit a model of the user's expertise to improve the quality of UC's responses to the user.

2. KNOME

KNOME (KNOledge Model of Expertise) is the component of UC that models what the user knows about UNIX. More details can be found in [Chin, 1986, 1987, 1988], so this section will only give enough information so that the reader can understand how KNOME is used by UCExpress.

KNOME uses a *stereotype* approach [Rich, 1979] where the characteristics of classes of users are organized under stereotypes. KNOME separates users into four levels of expertise (stereotypes): *novice*, *beginner*, *intermediate*, and *expert*. Individual users are classified as belonging to one of the above stereotype levels and inherit the characteristics of the stereotype. However, particular facts about the particular user override inheritance, so individual users differ from their stereotypes, which serve as reference points [Rosch, 1978].

Besides stereotypes for users, KNOME also has stereotype levels for UNIX facts. This feature is termed a *double stereo-*

* This research was sponsored in part by the Defense Advanced Research Projects Agency (DoD), under Arpa Order No. 4871, monitored by Space and Naval Warfare Systems Command under Contract N00039-84-C-0089; the Office of Naval Research, under grant N0014-48-C-0732; the National Science Foundation under grant 85-14890.

[1] UNIX is a trademark of Bell Laboratories.

type system [Chin, 1986]. Stereotype levels for UNIX facts include *simple, mundane, complex,* and *esoteric.* Examples of simple information are the rm, ls, and cat commands, the technical term "file," and the simple file command format (the name of the command followed by the name of the file to be operated upon). The mundane category includes the vi, diff and spell commands, the technical term "working directory," and the -l option of ls, while the complex category includes the grep, chmod, and tset commands, the term "inode," and the fact that write permission on the containing directory is a precondition for using the rm command for deleting a file. The esoteric category consists of information which is not in the mainstream usage of UNIX, but instead serves special needs. A good example is the spice program, that is useful only for people interested in semiconductor circuit simulations.

Thanks to the additional stereotype classification of UNIX information, it becomes extremely easy and space efficient to encode the relation between user stereotypes and their knowledge of UNIX. The core of this knowledge is shown in Table 1.

User	Knowledge Difficulty Level			
Stereotype	simple	mundane	complex	esoteric
expert	ALL	ALL	MOST	—
intermediate	ALL	MOST	AFEW	—
beginner	MOST	AFEW	NONE	—
novice	AFEW	NONE	NONE	NONE

Table 1. Relation between user stereotypes and knowledge difficulty levels.

Table 1 indicates that the novice user in session 1 (see Figure 1) likely does not know the format for the rm command, which is a simple fact, and definitely does not know the uptime command, which is a mundane fact. On the other hand, the intermediate user in session 2 (see Figure 2) definitely knows the format for the ls -i command, which is a simple fact, and is likely to know the uptime command.

3. UCExpress

After other components of UC have identified a response to the user, this is passed to UCExpress, which decides how much of the response to present to the user and how to format it. The separation of this process of deciding how much of the answer to express from the process of figuring out the answer was first suggested by [Luria, 1982] who applied this distinction to a question answering system for story understanding. His system first found the causal chain that represented the answer, then used answer expression to decide how much of the causal chain to express to the user.

The response passed to UCExpress is in the form of a conceptual network in the KODIAK representation language [Wilensky, 1987]. UCExpress, operates on this input in two phases, *pruning* and *formatting*. During pruning, UCExpress prunes common knowledge from the answer using information about what the user knows based on the conversational context and a model of the user's knowledge. Next the answer is formatted using specialized expository formats for clarity and brevity. The final result is an augmented KODIAK conceptual network that is ready for direct generation into natural language using a tactical level generator such as KING [Jacobs, 1985].

4. Pruning

When UCExpress is passed a set of concepts to communicate to the user, the first stage of processing prunes them by marking any extraneous concepts, so that the generator will not generate them. Pruning is done by marking rather than actual modification of the conceptual network, because information about the node may be needed to generate appropriate anaphora for the pruned concept.

The guiding principle in pruning is to not tell the user anything that the user already knows. Currently UC models two classes of information that the user may already know. The first class of information is episodic knowledge from a model of the conversational context. The current conversational context is tracked by marking those concepts that have been communicated in the current session. The second class of information concerns the user's knowledge of UNIX related facts. Such user knowledge is modeled by KNOME. Thus any concept that is already present in the conversational context or that KNOME indicates is likely to be known to the user is marked and is not communicated to the user.

4.1. An Example Trace

To see how pruning works in detail, consider the trace of a UC session shown in Figure 3.

```
# How can I print a file on the laser printer?
The parser produces:
(ASK10 (listener10 = UC)
       (speaker10 = *USER*)
       (asked-for10 =
        (QUESTION10
         (what-is10 = (ACTION14?
                        (actor14 = *USER*))))))
(PRINT-ACTION0? (pr-effect0 = PRINT-EFFECT0?)
                (actor0-1 = *USER*)
                (cause0-0 = (ACTION14? &)))
(HAS-PRINT-DEST0 (pr-dest0 = LASER-PRINTER0)
                 (pr-dest-obj0 = PRINT-EFFECT0?))
(HAS-PRINT-OBJECT1 (pr-object1 = FILE3?)
                   (pr-obj-obj1 = PRINT-EFFECT0?))
The planner is passed:
(PRINT-EFFECT0?)
The planner produces:
(PLANFOR260
 (goals260 = PRINT-EFFECT0?)
 (plan260 = (UNIX-LPR-Plz-COMMAND0
             (lpr-plz-file0 = FILE3?)
             (UNIX-LPR-Plz-COMMAND-effect0 =
              PRINT-EFFECT0?))))
(HAS-FILE-NAME18 (named-file18 = FILE3?)
                 (file-name18 = (lisp= nil)))
(LPR-Plz-HAS-FORMAT0
 (LPR-Plz-HAS-FORMAT-command0 =
  (UNIX-LPR-Plz-COMMAND0 &))
 (LPR-Plz-HAS-FORMAT-format0 =
  (LPR-Plz-FORMAT1
   (lpr-plz-file-arg1 =
    (file-name18 = aspectual-of (HAS-FILE-NAME18 &)))
   (LPR-Plz-FORMAT-step1 =
    (SEQUENCE10 (step10 = lpr)
                (next10 = (CONCAT00
                           (concat-step00 = -P)
                           (concat-next00 = lz))))))))
(HAS-COMMAND-NAME30
 (HAS-COMMAND-NAME-named-obj30 =
  (UNIX-LPR-Plz-COMMAND0 &))
 (HAS-COMMAND-NAME-name30 = (SEQUENCE10 &)))
UCExpress:  now expressing the PLANFOR:
```

```
(PLANFOR260 &)
UCExpress:  not expressing the format of the command,
UNIX-LPR-Plz-COMMAND0, since the user already knows it.
UCExpress:  not expressing PRINT-EFFECT0?,
since it is already in the context.
The generator is passed:
(TELL7 (listener7-0 = *USER*)
       (speaker7-0 = UC)
       (proposition7 = (PLANFOR260 &))
       (effect7 = (STATE-CHANGE1
                     (final-state1 = (KNOW-ga0? &)))))
Use lpr -Plz.
```

Figure 3. UC session with an intermediate user showing trace of UCExpress.

The above example traces UCExpress' processing of the question, "How can I print a file on the laser printer?" The answer given by UC is, "Use lpr -Plz." The actual KODIAK conceptual network that is passed to UCExpress, shown in Figure 4, is not nearly as succinct, because it contains all of the details of the command that are needed for planning.

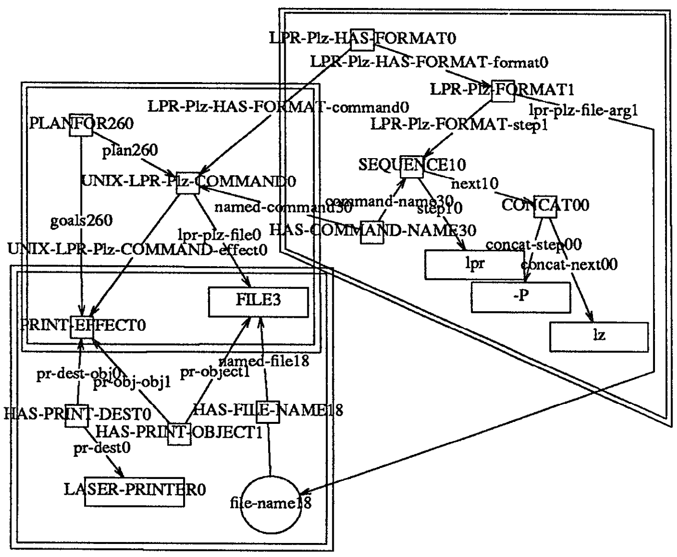

Figure 4. KODIAK representation of the lpr -Plz plan for printing.

If the KODIAK network passed to UCExpress were to be generated directly into English, it might look like the following:

> To print a file on the laser printer, use the lpr -Plz command. The command-format of the lpr -Plz command is "lpr" followed by concatenating "-P" with "lz" followed by the name of the file to be printed on the laser printer.

This literal paraphrase is harder to understand than UC's more concise answer. To see how UCExpress prunes the network to arrive at the actual answer, consider the division of the concepts into the following three subnetworks:

PLANFOR260:	A plan for PRINT-EFFECT0 is UNIX-LPR-Plz-COMMAND0
PRINT-EFFECT0:	printing a file on the laser printer
LPR-Plz-HAS-FORMAT0:	the format of the UNIX-LPR-Plz-COMMAND0 is "lpr -Plz <the name of the file to be printed>"

These three subnetworks are depicted in Figure 4 as regions enclosed in double lines. In traversing this network, UCExpress prunes LAS-PRINT-EFFECT0, because "printing a file on the laser printer" is already a part of the context (it is part of the user's question). Also, the command-format (LPR-Plz-HAS-FORMAT0) is pruned from UC's actual answer based on information from KNOME. In this case, KNOME was able to deduce that, since the user was not a novice, the user knows the UNIX-LPR-Plz-FORMAT, that is an instance of the SIMPLE-FILE-FORMAT (the name of the command followed by the name of the file to be operated upon), that all non-novice users know. Finally what is left unpruned is the plan part of PLANFOR260, UNIX-LPR-Plz-COMMAND0, which is generated as "Use lpr -Plz."

Pruning is similar to the "msg-elmt" realization stage of MUMBLE [McDonald, 1984] that was used to generate pronouns when a concept had been previously mentioned by MUMBLE. However, since MUMBLE did not have access to a model of the user, it was not able to avoid expressing those concepts which a user model would indicate that the user already knows. Another approach is used by KAMP [Appelt, 1985] in planning referring expressions. KAMP used mutual knowledge as a criterion for planning pronominal and anaphoric noun phrases. It would be very difficult to adapt such an approach to do pruning since KAMP does not deal with the uncertainty that is inherent in user models like KNOME that reason from stereotypes.

5. Formatting

After pruning, UCExpress enters the formatting phase, during which it tries to apply different *expository formats* to express concepts in a clearer manner. This is similar in spirit to the TAILOR system [Paris, 1988] that used a simulated user model to choose between two strategies for explanation: describing the processes and describing the parts of an object.

Each expository format is used to express different types of information. They are triggered by encountering particular concept types in the answer network. After triggering, the procedural component of the expository format is called to transform the concept into the corresponding format. The formats are not simple templates that can be filled in with readily available information. A fair amount of additional processing is needed to transform the information into the right format.

Due to space limitations, this paper will only describe two of UCExpress' expository formats: the *example* and *simile formats.* These are most interesting from the viewpoint of answer expression using pragmatic information about the user's domain knowledge.

5.1. Example Format

The example format is used in expressing general knowledge about complex (i. e. multi-step) procedures such as UNIX commands. In UC's representation of UNIX commands, every command has an associated command format. When expressing a command, UCExpress checks to see if it should also express the command's format. If KNOME believes that the user already knows the format of the command, then there is no need to express it. Next, UCExpress checks to see if the format of the command is completely specified. If so, UCExpress collapses the command and format into a single statement as shown in the UC dialog of Figure 5.

```
# How can I add general write protection to the file
personal?
Type 'chmod o-w personal'.
```

Figure 5. UC session with an answer that combines the command and format.

An English rendition of the conceptual network passed to UCExpress for the above example might be something like:

> A plan for adding general read protection to the file personal is to use the chmod command with format 'chmod' followed by concatenating 'o' with '-' with 'r' followed by 'personal'.

Since the command is completely specified, the format of the command is combined with the command to form a shorter and more easily understood answer.

If the command is not completely specified, then UCExpress uses an example format to express the format of the command to the user. The key principle in producing examples is to be explicit. UCExpress first steps through a copy of the general procedure to transform any general information into specific instances. In cases where the underspecified part of the procedure has a limited range of options, UCExpress selects an arbitrary member that is compatible with the rest of the procedure and with previous choices. Next, the new, completely specified copy of the format is combined with a copy of the command, much as in the above UC dialog. Finally the new plan is encapsulated in an example shell (which tells the generator to produce "For example,").

To see the algorithm in more detail, consider the UC dialog of Figure 6.

```
# How can I change the read permission of a file?
UCExpress:  now expressing the PLANFOR:
(PLANFOR330 &)
UCExpress:  creating an example for the incomplete
plan, CHMOD-FORMAT0
UCExpress:  choosing a name, foo, for an example file.
UCExpress:  selecting USER-PROT -- print name, u,
to fill in a parameter of the example.
UCExpress:  selecting ADD-STATUS -- print name, +,
to fill in a parameter of the example.
UCExpress:  not expressing CHANGE-PROT-FILE-EFFECT0?,
        since it is already in the context.
Use chmod.
For example, to add group read permission to the file
named foo, type 'chmod g+r foo'.
```

Figure 6. UC session showing the example format.

The conceptual answer that is passed to UCExpress in the dialog can be paraphrased in English as:

> A plan for changing the read permission of a file is to use the chmod command with format 'chmod' followed by concatenating <the protection-user-type> with <the protection-value-type> with 'r' followed by <the name of the file to be changed>.

In stepping through the above format, <the protection-user-type> is underspecified. In order to give an example, a particular value is needed, so UCExpress arbitrarily chooses a value from the list of possible fillers (user, group, other, or all). The same is done for <the protection-value-type>. In the case of 'r', this is already a fully specified value for protection-access-type, so UCExpress maintains the selection. However, with <the name of the file to be changed>, there is no list of possible fillers. Instead, UCExpress calls a special procedure for selecting names. This naming procedure chooses names for files starting with 'foo' and continuing in each session with 'foo2', 'foo3', etc. Other types of names are selected in order from lists of those name types (e. g. machine names are chosen from a list of local machine names). By selecting the names in order, name conflicts (e. g. two different files with the same name) can be avoided.

Another consideration in creating examples is that new names must be introduced before their use. Thus 'foo' should be introduced as a file before it appears in 'chmod g+r foo'. This is done implicitly by passing the entire PLANFOR as the example, so that the generator will produce 'to add group read permission to the file named foo' as well as the actual plan.

5.2. Simile Format

The simile format is used by UCExpress to provide explanations of what a command does in terms of other commands already known to the user. This format is invoked when UCExpress attempts to explain a command that has a sibling or a parent in the command hierarchy that the user already knows (as modeled in KNOME). An example is explaining what ruptime does in terms of uptime. A trace of UC's processing is shown in Figure 7.

```
# What does ruptime do?
UCExpress: Found a related command, so comparing
UNIX-RUPTIME-COMMAND2 and UNIX-UPTIME-COMMAND0
ruptime is like uptime, except ruptime is for all
machines on the network.
```

Figure 7. UC session showing the simile format.

The processing involves comparing the effects of the two commands and noting where they differ. In the above example, the effects of uptime are to list the uptime of the user's machine, list the number of all users on it, and list its load average. The effects of ruptime are similar except it is for all machines on the user's network. The comparison algorithm does a network comparison of the effects of the two commands. A collection of differences is generated, and the cost of expressing these differences (measured in number of concepts) is compared with the cost of simply stating the effects of the command. If expressing the differences is more costly, then the simile format is not used. On the other hand, if expressing the differences is less costly, then the differences are combined into a shell of the form "<CommandA> is like <CommandB>, except [<CommandA> also ...] [and] [<CommandA> does not ...] [and] ..."

5.3. A Comparison

The TEXT system [McKeown, 1985] is perhaps the closest in spirit to UCExpress. TEXT used a *compare and contrast schema* to answer questions about the differences between objects in a database. This is similar to UCExpress' simile format except that the compare and contrast schema was not used for giving descriptions of an object in terms of another that the user already knew. Since TEXT did not have a complete model of the user, it was unable to determine if the user already knew another object that could be contrasted with the requested object. This lack of a user model was also evident in the fact that TEXT did not provide anything like the pruning phase of UCExpress. Pruning is probably more relevant in a conversational context such as UC as contrasted with a paragraph generation context such as TEXT.

Other related research include work on using examples for explanation and for legal argumentation [Rissland et al., 1984]. The difference between those examples and the examples created by UCExpress is that Rissland's examples are preformed and stored in a database of examples whereas UCExpress creates examples interactively, taking into account user provided parameters. Rissland's HELP system dealt only with help about particular subjects or commands rather than arbitrary English questions like UC, so HELP did not have to deal with questions such as how to print on a particular printer. Also by using prestored text, HELP was not concerned with the problem of transforming knowledge useful for internal computation in a planner to a format usable by a generator.

6. Conclusion

UC separates the realization of speech acts into two processes: deciding how to express the speech act in UCExpress, and deciding which phrases and words to use in UC's tactical level generator. Through this separation, the pragmatic knowledge needed by expression is separated from the grammatical knowledge needed by generation. UCExpress makes decisions on pragmatic grounds such as the conversational context, the user's knowledge, and the ease of understand of various expository formats. These decisions serve to constrain the generator's choice of words and grammatical constructions.

Of course, it is sometimes impossible to realize all pragmatic constraints. For example, UCExpress may specify that a pronoun should be used to refer to some concept since this concept is part of the conversational context, but this may not be realizable in a particular language because using a pronoun in that case may interfere with a previous pronoun (in another language with stronger typed pronouns, there may not be any interference). In such cases, the generator needs to be able relax the constraints. By passing the generator all of the conceptual network along with addition pragmatic markings on the network UCExpress allows the generator to relax constraints as needed. This way, the generator has access to any information needed to relax the constraints added by UCExpress.

Acknowledgements

The work described in this paper was done at the University of California, Berkeley as part of my Ph.D. thesis. I wish to thank Robert Wilensky who supervised this work. I also wish to thank the members of BAIR (Berkeley Artificial Intelligence Research) who have contributed to the UC project.

References

Appelt, D. E. (1981). *Planning Natural Language Utterances to Satisfy Multiple Goals.* Doctoral dissertation, Computer Science Department, Stanford University. Also available as SRI International AI Center Technical Note 259.

Chin, D. N. (1986). User modeling in UC, the UNIX consultant. In *Proceedings of the CHI-86 Conference,* Boston, MA, April 1986.

Chin, D. N. (1987). *Intelligent Agents as a Basis for Natural Language Interfaces.* Doctoral dissertation, Computer Science Division, University of California, Berkeley.

Chin, D. N. (1988). KNOME: Modeling What the User Knows in UC. To appear in A. Kobsa and W. Wahlster (Eds.), *User Models in Dialog Systems,* Berlin: Springer.

Jacobs, P. S. (1985). *A Knowledge-Based Approach to Language Production.* Doctoral dissertation, University of California, Berkeley. Also available as Computer Science Division, University of California, Berkeley, Report No. UCB/SCD 86/254.

Luria, M. (1982). Dividing up the Question Answering Process. In *Proceedings of the National Conference on Artificial Intelligence,* pp. 71-74. Pittsburgh, PA, August.

McDonald, D. D. 1984. Natural Language Generation as a Computational Problem: an Introduction. In *Computational Models of Discourse,* edited by M. Brady and R. C. Berwick. MIT Press. Cambridge, MA. 1984.

McKeown, K. R. (1985). Discourse Strategies for Generating Natural-Language Text. In *Artificial Intelligence,* 27, pp. 1-41.

Paris, C. L. (1988). Tailoring Object Descriptions to a User's Level of Expertise. To appear in Kobsa, A. and Wahlster, W. (Eds.), *User Models in Dialog Systems.* Berlin: Springer.

Rich, E. (1979). User Modeling via Stereotypes. In *Cognitive Science,* 3, pp. 329-354.

Rissland, E. L., Valcarce, E. M., and Ashley, K. D. (1984). Explaining and Arguing with Examples. In *Proceedings of the National Conference on Artificial Intelligence,* pp. 288-294. Austin, TX, August.

Rosch, E. (1978). Principles of Categorization. In Rosch, E. and Lloyd, B. B. (Eds.), *Cognition and Categorization.* Hillsdale, NJ: Lawrence Erlbaum.

Wilensky, R., Arens, Y., and Chin, D. N. (1984). Talking to UNIX in English: An Overview of UC. In *Communications of the ACM,* 27 (6), pp. 574-593. June.

Wilensky, R., Mayfield, J., Albert, A., Chin, D. N., Cox, C., Luria, M., Martin, J., and Wu, D. (1986). *UC—A Progress Report.* Computer Science Division, University of California, Berkeley, Report No. UCB/CSD 87/303.

Wilensky, R. (1987). *Some Problems and Proposals for Knowledge Representation.* Computer Science Division, University of California, Berkeley, Report No. UCB/CSD 87/351.