

KNOME: Modeling What the User Knows in UC*

David N. Chin

Department of Information and Computer Sciences
University of Hawaii at Manoa
2565 The Mall
Honolulu, HI 96822
email: Chin@Hawaii.Edu

ABSTRACT

KNOME is the user modeling component of UC, a natural language consultation system for the UNIX operating system. During the course of an interactive session with a user, KNOME infers the user's level of expertise from the dialog and maintains a model of the user's knowledge of the UNIX domain. KNOME's model of the user makes use of a *double-stereotype* system in which one set of stereotypes represents the user's expertise and another represents the difficulty level of the information. KNOME is used in UC to help: disambiguate the user's statements, avoid telling the user something that the user already knows, take advantage of prior user knowledge in presenting new information, and detect situations where the user lacks pertinent facts or where the user has a misconception. UC also models its own knowledge of UNIX with *meta-knowledge* (explicit facts about the limitations of the system's own knowledge base), which is used to help in correcting user misconceptions.

March 31, 1994

* This paper is a reprint from chapter 4 in A. Kobsa and W. Wahlster (Eds.), *User Models in Dialog Systems*, pp. 74-107, Berlin: Springer-Verlag, 1989.

KNOME: Modeling What the User Knows in UC*

David N. Chin

Department of Information and Computer Sciences
University of Hawaii at Manoa
2565 The Mall
Honolulu, HI 96822
email: Chin@Hawaii.Edu

1. Introduction

UC, the UNIX Consultant [WILE84, WILE86], is a natural language computer consultant system under development at the University of California at Berkeley. UC advises users on using the UNIX¹ operating system. Users can ask UC how to do things in UNIX, get definitions of UNIX terminology, and get help debugging problems in using commands.

To better accommodate a greater range of potential users, UC contains an explicit user modeling component called KNOME (KNOWledge Model of Expertise). Some of the early results of this research were reported in [CHIN86]. KNOME contains information about what users know. Using such information, UC can:

- 1) avoid telling the user something that the user already knows
- 2) tailor explanations for the user's level of understanding
- 3) utilize the user's background knowledge in interpreting what the user says

KNOME is also helpful in detecting cases where the user's knowledge is incorrect (misconceptions) and cases where the user lacks needed information. In such cases, *UCEgo* [CHIN87], the component of UC that implements an intelligent agent for UC, can take the initiative and try to correct the user's misconception, or try to inform the user of the needed information, respectively.

1.1. Some UC Examples

To see how KNOME can infer a user's level of expertise from a conversation with the user, and how this information can be used advantageously by UC in helping the user, consider the two UC sessions shown in Figure 1 and Figure 2.

* This paper is a reprint from chapter 4 in A. Kobsa and W. Wahlster (Eds.), *User Models in Dialog Systems*, pp. 74-107, Berlin: Springer-Verlag, 1989.

¹ UNIX is a trademark of Bell Laboratories.

```
Welcome to UC (Unix Consultant) version 3.23
To a '#' prompt, please type in your questions
about the UNIX file system in English.
To leave, just type '^D' or '(exit)'.

Hi.
How can I help you?

# 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 with User1.

```
Welcome to UC (Unix Consultant) version 3.23
To a '#' prompt, please type in your questions
about the UNIX file system in English.
To leave, just type '^D' or '(exit)'.

Hi.
How can I help you?

# 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 with User2.

The user of the first session, User1, is apparently a novice who knows very little about UNIX. After all, User1 does not even know something as simple as how to delete a file (the subject of User1's first question). On the other hand, the user in the second session, User2, knows what an inode is, a relatively complex UNIX concept. So User2 is probably a much more advanced user of UNIX than User1. KNOOME uses this type of evidence about particular facts

that users know or do not know in order to deduce their level of expertise.

Based on the different levels of expertise of the two users as deduced by KNOOME, UC is able to provide qualitatively different answers to the two users. In the first answer to User1, KNOOME gives an example of the format of the `rm` command (simply “`rm`” followed by the name of the file to be deleted). UC gives an example of the format to User1, since a user as unsophisticated as User1 probably would not know the format of `rm`. On the other hand, User2 is a more advanced user who undoubtedly already knows the format of `ls`. Hence UC is able to give a very concise answer to User2’s first query, “Use `ls -i`,” that does not include an example of the format of `ls`. This illustrates how KNOOME is used to avoid telling the user something that the user already knows.

The second query of both users is the same. For the more advanced user, UC was able to explain the `ruptime` command in terms of a similar command, `uptime`, that the more advanced user presumably already knows. Such a simile would not work for User1, since a novice user probably would not know the `uptime` command, and so could not exploit the simile to understand `ruptime`. KNOOME’s modeling of the user’s knowledge allows UC to choose whether or not to use different forms of presentation like similes based on whether or not the user will understand them.

1.2. Problems in Modeling

There are many difficulties in building and maintaining a model of what the user knows. The system must first determine what the user knows, and then represent this information internally. To determine the user’s knowledge of the domain, the system could quiz the user exhaustively. However for most domains, such a quiz would be extremely time consuming and most users would not consent to such extensive quizzing. Fortunately, users tend to learn domain knowledge in a predictable order, so the system can make additional inferences about what a user is likely to know or not know based on a partial user model. That is, the user modeling system should be able to *predict* a user’s knowledge based on partial information about what the user knows or does not know. Exactly which predictions/inferences are allowable is problematic. This is the key problem for any user modeling system that wishes to take advantage of the order in which users typically gain knowledge in a domain.

Besides the basic question of which inferences are allowable, there is also the problem of uncertainty. Making inferences about the user based on limited prior knowledge about the user is a form of default reasoning. Such default reasoning is not always valid, so the system needs to be able to handle the fact that such inferences are uncertain. Usually this requires that the user modeler be able to judge the certainty of its default inferences. Also, as the system obtains more concrete information, it must be able to update its user model. The new information may contradict previous predictions, so maintenance is not simple.

Another problem concerns the organization of the inferences. Many inferences tend to be grouped together, because people tend to go through distinct stages of learning (see [KAYD85]). People at a particular stage of learning tend to have similar knowledge. Once a system has determined its user’s level, it can assume that this user knows about as much as other users at that level. A good user modeling system needs to group such inferences together in a simple and efficient manner.

Another problem in user modeling is how to represent the model internally. The simplest method is the *overlay* model [CARR77] wherein the user’s knowledge is represented as a subset of the system’s knowledge. However, a simple overlay model cannot predict what a user might know based on partial information. An overlay model does not represent the order in which users typically learn information in a domain. What is needed is a model that represents such ordering

information in a manner useful for prediction of what a user might know based on partial information.

1.3. Other User Models

User models have appeared frequently in Intelligent Tutoring Systems (e. g. [BROW76a, SLEE82c]). [KASS*] provides a survey of user models in ITS. Typically, ITS use some variant of the simple overlay model [CARR77]. Overlay models encode the user's knowledge as a subset of the system's expert knowledge base. One variant is to encode users as a collection of differences from the system's built-in expert. Differences may include a lack of user knowledge relative to an expert. This is frequently augmented by a library of buggy procedures/rules that are common among users (e. g. [BROW78, STEV79, SLEE81, JOHN84, REIS85, ANDJ85b]). These types of user models are not designed to *predict* a user's knowledge based on partial information about what the user knows or does not know. This sort of default inference is not so appropriate for ITS where the system needs to keep an accurate user model. On the other hand, consultation systems usually do not interact with a single user for as much time as tutoring systems, so they do not have the time to build a complete user model. In such cases, default reasoning is needed. Therefore, consultation systems need a different type of user model than is commonly found in ITS.

An exception is the hierarchical overlay model of USCSH [MATT86] which does some limited prediction. USCSH predicts the user's proficiency for parent nodes in the hierarchy by taking a weighted sum of the proficiency ratings of the children nodes. This approach allows prediction of highly related topics, but does not extend to prediction of the user's knowledge of less related topics. For example, such a system could predict that a user has high proficiency in process monitoring when the system knows that the user has high proficiency in using the `ps`, `who` and `kill` commands. However such a system could not predict the proficiency of the user in an unrelated topic, such as using the `cat` command, based on the system's knowledge of the user's proficiency in process monitoring commands.

Other user models have been built to model a number of aspects of the user (see [WAHL*] for a survey). For example, Grundy [RICH79b, RICH83, RICH*] modeled user personality traits. Grundy used user *stereotypes* to group default inferences about user traits, then used these traits for suggesting books to the user for reading. To determine which stereotypes were applicable to a particular user, Grundy asked the user for a self description at the start of a session. Individual phrases (e. g. self descriptions such as athletic or feminist) would *trigger* stereotypes that were likely to apply to that user. Such an approach worked well for modeling personality traits, but does not carry over to modeling domain knowledge. A user's self description as a "beginner" often will not coincide with the system's idea of a "beginner." Also, stereotype triggers cannot be used to make inferences such as inferring that the user does not know something when the user asks about it. Stereotype triggers are not sufficient for inferring what users know.

Another use of user models is for representing user preferences. The hotel reservation system of HAM-ANS [HOEP84b, MORI*] and the Real Estate Agent [MORI85b] chose different hotel rooms or apartments based on users' needs and preferences. These systems expect users to mention at the start of their session their requirements and preferences for a room/apartment. Such an approach works well in domains where users tend to specify their needs and preferences at the start of a session. However this is not the case in consultation domains where users are unlikely to mention their level of expertise in the domain at the start of a session.

Many systems including UC have modeled the user's plans and goals (e. g. [ALLE80, CARB83, CARB*, LITM84, JOHN84, WILE86]) Such systems infer the user's plans and goals

by matching the user's actions to steps of prestored plans. This type of inference works well for detecting the user's plans, but does not extend to determining what the user does or does not know.

2. Internal Representation of Users

KNOME represents what UC believes users know about UNIX. In this sense, KNOME's model of the user is not necessarily meant to accurately model actual users, but is meant to conform to the way human UNIX consultants model their clients. So in developing KNOME's user models, no attempt was made to psychologically profile what different users know. Instead, former UNIX consultants (UC's implementors) were informally surveyed to determine how they viewed users.

2.1. Double-Stereotypes

Humans use categorization to organize inferences (see [ROSC77, ROSC78, ROSC83]). These categories serve as reference points or prototypes for judging objects. Once an object has been identified as belonging to a particular category, default inferences about the object can be made based on the object's membership in the category.

An approach similar to human categorization is used in KNOME to organize inferences concerning users. In KNOME, users are grouped according to their level of expertise in using UNIX. Each group is represented by a prototype-style category. There are four such categories in UC: *novice*, *beginner*, *intermediate*, and *expert*. Each of the categories represents an increasing mastery of UNIX information.

Individual users are represented as members of one of these categories and inherit properties from their category. Specific information about individual users overrides inherited information. These user categories are also termed stereotypes after the stereotypes of Grundy [RICH79b, RICH83, RICH*], which pioneered categorization of users in computer programs. However, unlike the stereotypes in Grundy, which consist of collections of attributes, the stereotypes in KNOME are implemented as categories within the KODIAK representation system [WILE87]. As such, KNOME's stereotypes are integrated into KODIAK's multiple inheritance hierarchy and its default inheritance mechanisms.

Besides categorizing users, KNOME also categorizes information into levels of difficulty. Commands, command-formats, terminology, and other relevant information are categorized according to their level of difficulty. These stereotype categories include *simple*, *mundane*, and *complex*. Information is grouped into categories based on their typical location on the learning curve, in other words, based on when the typical user would learn the information. For example, some simple concepts include 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). These simple concepts are learned early in the experience of a typical user. Somewhat more advanced are mundane level concepts. Some examples are the `vi`, `diff` and `spell` commands, the technical term "working directory," and the `-l` option of `ls`. Examples of complex concepts are 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. Complex concepts are learned relatively late by the typical user.

In addition to the three previously mentioned levels of difficulty, KNOME has the category *esoteric* that includes information that is not typically learned at any particular stage of experience. Such concepts are learned only when users have special needs. Some may be familiar to

beginners, yet unfamiliar to many experts. An example of an esoteric command is `spice`, a program used for electronic circuit simulation. Only people who need to perform circuit simulations would know how to use `spice`. This group would likely include beginning users as well as intermediate and expert users. On the other hand, there are many expert UNIX users who do not know about `spice`, because they have never needed to perform semiconductor circuit simulations.

Once information has been classified into levels of difficulty, inferences based on user stereotypes can easily be represented as relations between the various user stereotypes and difficulty levels. For example, Figure 3 shows a graphical KODIAK representation of one such relation. This Figure shows the KODIAK representation of the fact that intermediate users know most facts that have the difficulty level of mundane. The predicate of MOST is used, because there is uncertainty as to whether any particular intermediate user will know any particular mundane level fact. However, any intermediate level user will know most mundane level facts. Table I summarizes the relation between different user stereotypes and different knowledge difficulty levels.

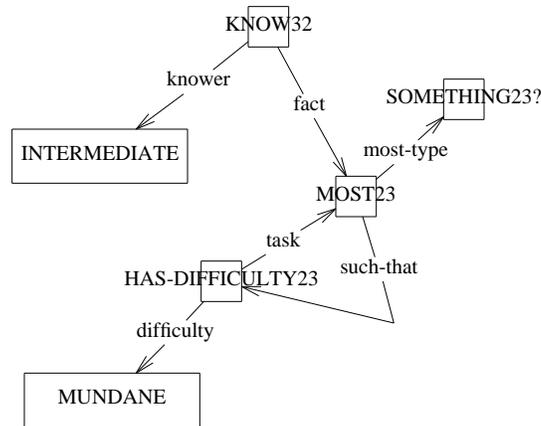


Figure 3. KODIAK representation of: intermediates know most mundane facts.

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

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

The use of stereotype categories for both users and information is an example of a *double-stereotype* system. Single stereotype systems like Grundy used stereotypes for users (e. g., FEMINIST, INTELLECTUAL, SPORTS-PERSON) but not for information (i. e., Grundy did not have stereotypes for books such as GOTHIC-ROMANCE, DETECTIVE-STORY, or SCIENCE-FICTION). KNOME is the first user modeling system to utilize double-stereotypes.

The use of stereotypes allows KNOME to *predict* what users are likely to know or not know based on a partial user model. Once KNOME has accumulated a small number of facts

(typically ~3)² about what users do or do not know during the course of a UC session, it can infer the user's level of expertise (see Section 3). Then based on the user's level of expertise, KNOOME can predict whether or not the user is likely to know other facts. The small set of facts that allows KNOOME to infer the user's level of expertise constitutes a partial user model, and the inference that the user belongs to a particular stereotype represents a set of default inferences that are made on the basis of the partial user model.

The use of stereotypes allows KNOOME to efficiently group these default inferences together. A system that did not use stereotypes would need to represent individually every partial user model and all the default inferences from that partial user model. Since a partial user model is simply a set of facts about a user, there is a combinatorically large number of partial user models. So, representing all such partial user models and their default inferences would be extremely inefficient. However many partial user models have similar default inferences. These can be grouped together efficiently using user stereotypes. A double-stereotype system is even more efficient, since it allows the system to encode for each fact only its level of difficulty rather than having to encode its relationship to every user stereotype. The rest of a double-stereotype system can then be encoded in a small number of statements relating the two levels of stereotypes. This relationship in KNOOME between user stereotypes and knowledge difficulty levels is summarized in Table I.

KNOOME currently has only one double-stereotype range for users. This range represents user's knowledge of UNIX file manipulation commands, UNIX information gathering commands, and UNIX concepts and terminology. This corresponds to UC's domain of expertise within UNIX. If UC were extended to cover more of UNIX or even other operating systems, then KNOOME would need more ranges of double-stereotypes. For example, if UC also covered usage of the `vi` and `emacs` editors, then KNOOME would need separate expertise ranges for each of these domains. KNOOME would also need to encode how the user's level of expertise in one domain might be used to predict the user's level of expertise in another domain. So, an expert in using `vi` would undoubtedly have a high level of expertise in manipulating the UNIX file system. On the other hand, a high level of expertise in using `emacs` does not necessarily indicate a high level of expertise in using UNIX, since the `emacs` editor is commonly found in other operating systems.

2.2. Modeling Individual Users

Individual users usually do not completely fit any one stereotype. As a result, a user model that uses stereotypes must also encode how individual users differ from the stereotypes. KNOOME stores specific information about what individual users know and do not know. Such information is stored as a collection of propositions about what a user knows or does not know. These propositions are represented using KODIAK. In reasoning about what a user knows, this collection of individual facts is checked before resorting to reasoning from the user's stereotype category.

To avoid storing too many individual facts about what users know, only those facts that cannot be inferred with high likelihood from a user's category are stored. For example, the fact that a particular intermediate user knows the `rm` command (a simple command) does not need to be stored explicitly, since it is directly inferable from the fact that intermediate users know *all* simple facts. Also, the fact that a particular beginner knows the `rm` command does not need to be stored. This is because beginners know *most* simple facts and hence it is likely that any

² The actual number varies, because facts may mutually reinforce KNOOME's evaluation of the user's level of expertise, in which case fewer facts will be needed; or facts may provide contradictory evidence, in which case more facts will be needed to evaluate the user's level of expertise.

particular beginner would know `rm`. On the other hand, the fact that a particular novice knows the `rm` command does need to be stored explicitly, since that is not inferable with high likelihood. Because novice users know just *a few* simple facts, it is unlikely that any particular novice would know `rm`.

2.3. Types of Knowing

In UC, two senses of “know” are used. The first is the classical sense of know: the knower believes that some proposition *P* is true, and *P* is true. An example of this sense of knowing is: “the user knows that the `cd` command is used to change the current working directory.” There is also another sense of know that is found in UC. This sense of know is closer to the colloquial usage wherein knowing an object means being familiar with that object. However, in UC this vague usage is made precise. For UC, knowing a command such as the UNIX-RM-COMMAND implies knowing that there exists a command with the name `rm`, knowing the main purposes of this command, knowing the format of the command, knowing the main effects of the command, and knowing the main preconditions of the command.

The main purposes of a command are represented in UC using the *planfor* [CHIN83, WILE86] relation, which relates a goal and the plan that is typically used to satisfy that goal. The planfors of a plan are different from the effects of the plan, since not all of a plan’s effects can be considered goals that the plan is typically used to satisfy. All goals in the planfors of a command are also effects of the command, but some side effects of the command may not be part of its planfors. For instance, calling the game program `rogue` with a file argument has a side effect that is not found in the planfors of `rogue`. When `rogue` is given a file argument that is not a `rogue-format save file`, then `rogue` arbitrarily removes the file. This is a side effect of the `rogue` program, but it is not in any planfors, since users do not typically run `rogue` in order to remove files.

2.4. Dealing with Uncertainty

In any user model, the inferences that the model makes about the user contains some degree of uncertainty. In KNAME, an attempt was made to avoid the use of numeric representations of uncertainty, because their use tends to lead systems to overestimate the accuracy of their ratings of uncertainty. So, UC uses a fixed number of simple rating levels instead: LIKELY, UNLIKELY, VERY-LIKELY, VERY-UNLIKELY, SOMEWHAT-LIKELY, SOMEWHAT-UNLIKELY, TRUE, FALSE, and UNCERTAIN (c. f. fuzzy logic [ZADE65]). KNAME also uses predicates such as AFEW, MOST, ALL, and NONE (c. f. [ZADE82]). Such predicates and ratings are used by KNAME to express the certainty of its inferences. For example, when KNAME is asked whether an particular user knows some particular fact, it will return either TRUE, LIKELY, UNLIKELY, FALSE, or UNCERTAIN. TRUE and FALSE indicate that KNAME has complete confidence. LIKELY and UNLIKELY indicate some uncertainty. An UNCERTAIN result indicates that KNAME does not have enough information to guess whether the user does or does not know that fact.

To deduce whether a particular user knows some particular fact, KNAME uses a multi-step inference process. First, KNAME checks the individual user model for that user. If the individual user model records that this user does or does not know this fact, then the answer is, respectively, TRUE or FALSE. If there is no specific information in the user’s individual user model, KNAME next checks the stereotype category of the user. If users of the user’s stereotype do or do not know that fact, then the answer is again TRUE or FALSE, respectively. If these checks fail, KNAME resorts to inference based on the difficulty level of the fact. If the user’s stereotype knows ALL facts of that difficulty level, then the answer is TRUE; if the stereotype knows

MOST facts of that difficulty then, the answer is LIKELY; and if the stereotype knows AFEW facts of that difficulty, then the answer is UNLIKELY. Finally, if this process fails due to lack of information, then the answer is UNCERTAIN. This algorithm is summarized in the flow chart of Figure 4.

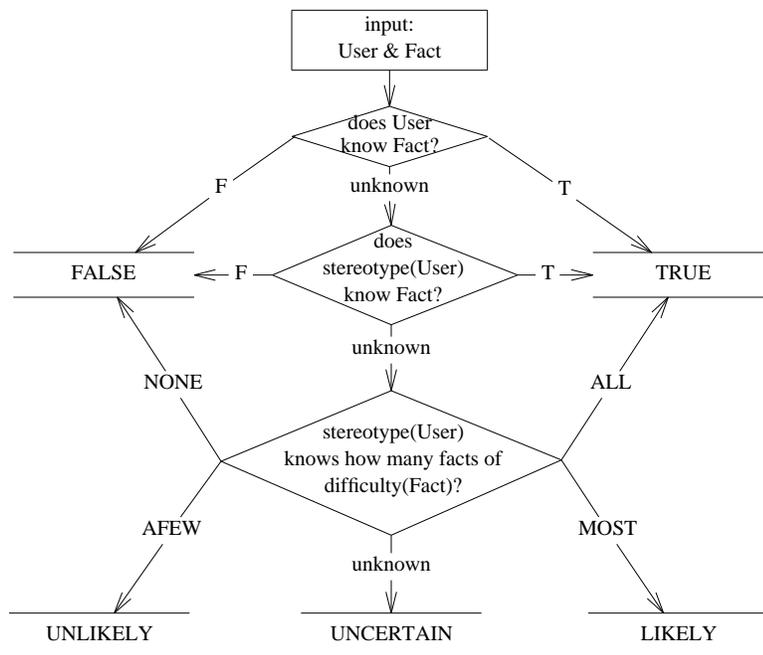


Figure 4. Algorithm for determining whether User knows Fact.

3. Deducing the User's Level of Expertise

One of the main problems in user modeling is how to build up a model of a particular user. In a user modeling system based on stereotypes, this means determining which stereotype(s) best fit the current user and how that user differs from the stereotypes. In UC, this means that KNOME must determine the user's level of expertise in using UNIX.

3.1. Other Model Acquisition Systems

Various approaches to building up models of users have been used. Some systems such as the Real Estate Agent [MORI85b] expect users to mention enough self-characteristics at the start of a session so that the system can build up the user model. This works for situations such as a real estate agent interchange with a client where the client is likely to mention what he/she needs/wants at the beginning of a dialog. However, this does not extend to other consultation situations where the user usually will not mention what he/she knows at the start of a session. In such consultation situations the user typically begins by describing his/her problem to the consultant.

Another approach is to ask the user for a self-description at the start of a session as in Grundy [RICH79b] This is a reasonable approach for judging the user's personality traits, but does not work well for determining what the user knows. For judging the user's level of expertise, this approach is inaccurate, since the user's concept of different levels of expertise like

“intermediate” may differ considerably from the system’s concept or from other users’ concept of an “intermediate.” Also, this requirement is not very practical for occasional users, since this increases the overhead of using the system.

A third approach is to quiz users at the start of a session so that the system may estimate the user’s expertise. For example, the MAP system [MACM83] helps a user to specify a correct user model of how that user interacts with an operating system. Such an approach is extremely time consuming. For tasks like intelligent operating system interfaces for which MAP was designed, such a protracted process is somewhat more acceptable, since the user will realize its benefits over a long period of time. However for interacting with a consultation system like UC where typical sessions are short, such a process would be too time consuming. The casual user that only needed the answer to a single question would be better off without a user model rather than having to put up with such a lengthy process.

The most common approach in ITS (Intelligent Tutoring Systems) is to compare the user’s performance with what a built-in expert would do. Differences in performance can then be attributed to either a lack of user knowledge or differences in user knowledge such as improper procedures or rules. However such an approach is only applicable if the system can “look over the shoulder” of the user. Consultation systems often cannot observe the user at work. Even when this is possible, consultation systems should not be expected to constantly observe the user, since human consultants do not do so.

To properly emulate a human consultant, the system should be able to assess the expertise level of the user while talking to the user. None of the previously mentioned approaches model this ability, which is a difficult problem in natural language understanding. One system that has addressed this problem is the VIE-DPM user modeling system [KOBS85d]. However, VIE-DPM uses only a syntactic interpretation of the user’s statements to derive the user’s beliefs. VIE-DPM makes assumptions about the user’s beliefs based on the form of the user’s input (a yes/no question, a wh-question, or a command) [KOBS85d, KOBS86a] or the presence of linguistic particles (e. g. “not”) in the user’s input [KOBS85b]. Such a syntax-only system would not be able to properly interpret phenomena such as indirect speech acts [SEAR69]. For example, “Can you pass the salt?” is a yes/no question, but one should not assume from this question that the speaker does not know whether the listener can pass the salt. Despite such limitations, VIE-DPM is a step in the right direction.

3.2. KNOME’s Approach

KNOME uses the approach of inferring the user’s level during the course of a session. At the beginning of a session, UC has no information about the user. In such cases, KNOME remains open to the possibility that the user may belong to any stereotype. KNOME maintains a list of candidate stereotypes to which the user may belong. KNOME also rates the likelihood that the user may be a member of any particular stereotype. At the start, the likelihood ratings for membership in any stereotypes is neutral, except for the beginner stereotype. The likelihood rating for beginner is initially slightly higher than neutral, since more of UC’s users tend to be beginners rather than novices, intermediates, or experts. As more information is gathered about the user during the session, the stereotypes may rise or fall in likelihood.

KNOME deduces the user’s level of expertise using a two phase process. First, during the course of the conversation, KNOME deduces individual facts about what the user does or does not know. As such facts are gathered, KNOME combines the evidence to adjust the likelihood ratings for the different levels of expertise. Eventually KNOME identifies the stereotype level of expertise to which the user belongs.

3.3. Collecting Evidence

The first step in the process of deducing the user's level of expertise is to collect evidence on that subject. Typically, such evidence takes the form of individual facts about what the user knows or does not know. Such facts and other forms of evidence can be deduced from what the user says (or does not say) and from an analysis of the user's goals. KNAME distinguishes six classes of inferences that were found to be useful in UC for determining the user's level of expertise. These inference classes are summarized in Table II, which lists the inference class and the kinds of inferences that are commonly found in each inference class.

Inference Class	Kinds of Inferences
Claim	user states that user does (not) know ?x → user does (not) know ?x
Goal	user wants to know ?x → user does not know ?x
Usage	user uses ?x → user knows ?x
Background	user mentions his/her background → user knows as much as the stereotype indicated by the background
Query-Reply	system asks if user knows, user replies → user's reply
No-Clarify	system uses new terminology, user does not ask for clarification → user understands terminology

Table II. Taxonomy of deduction types for inferring what users know.

The first class of inferences, Claim inferences, applies when the user claims to know or not know something. The second class, Goal inferences, applies when the system can infer that the user wants to know something. In such cases, the system can also infer that the user must not know whatever is wanted, since if the user already knew it, then the user would not have the goal of knowing it. The third class, Usage inferences, are made when the user properly uses some command or terminology. KNAME can then deduce that the user must know the command or terminology. The fourth class, Background inferences, are made when the user mentions some relevant information about his/her background. The fifth class of inference, Query-Reply, applies when the system queries the user about the user's knowledge and the user replies. The last type of deduction, No-Clarify, is made when the system uses some new terminology that the user may or may not know. If the user does not ask for a clarification of the terminology, then the system can assume that the user probably already knew the terminology. Each of these classes of inference is described in greater detail below.

Claim inferences are concerned with direct statements by the user about what the user does or does not know. Examples of such statements and the inferences that KNAME makes based on the statements include:

I already know about mkdir.
→ the user knows the mkdir command

I don't know how to move a file.

- the user does not know how to move a file
- the user does not know that `mv` is a plan for moving files
- the user does not know the `mv` command

In the first example, the user claims to already know the `mkdir` command. This statement is not as straightforward as it might seem, since the user and the system may not agree on what it means to know a command. A reasonable interpretation of what a user might mean is that the user knows how to use `mkdir`, knows the main purpose of `mkdir` and knows the main effects and preconditions of `mkdir`. This interpretation is the same as what UC means when it encodes that a user knows a command (see Section 2.3).

Of course, it may be that the system should not believe the user's statement about knowing `mkdir`. It is possible that the user may be confused and does not actually know `mkdir`. But, there is always doubt about what someone might really know, no matter how strong the evidence. For example, even if the system were to watch the user use the `mkdir` command successfully, it is possible that the user may have actually wanted to create a file instead of a directory. However, in none of these cases is there any reason to doubt the user. Unless KNOME obtains evidence to the contrary, it assumes that the user really does (or does not) know something when the user makes such a claim.

The second example requires somewhat more complex processing than the first example. In the second example, the user claims not to know how to move a file. Since UC knows that `mv` is used to move files, KNOME is able to deduce that the user does not know that `mv` is a plan for moving files. Finally, because the user does not know one of the main purposes of the `mv` command, KNOME deduces that the user must not know `mv`.

KNOME only stores information about levels of difficulty for objects and not for descriptions, so only facts about a user knowing (or not knowing) specific objects (e. g. commands, concepts, etc.) can be used in deducing the user's level of expertise. This approach eliminates redundancy, since any object will have many possible descriptions. For example, the user could have referred to `mv` by "how to rename a file" rather than "how to move a file." So, the fact that the user does not know `mv` is useful for deducing the user's level of expertise, but the descriptive fact that the user does not know how to move a file cannot be directly used. UC must first determine that this description applies to the `mv` command before it can be used for deducing the user's level of expertise. In UC, descriptions of the purpose of commands are interpreted by UC's UNIX planner [WILE86]. After UC's UNIX planner has determined the command referred to by the command description, KNOME can use this information in deducing the user's level of expertise.

Goal inferences apply when UC can deduce the user's goals. If UC can infer that the user wants to know something, `?x`, then KNOME can deduce that the user does not know `?x` and also does not know the answer to `?x`. For example, if the user asks "How can I delete a file?" then UC can deduce that the user's goal is to know how to delete a file. From this goal, KNOME can deduce that the user does not know how to delete a file. Then, after UC computes that a plan for deleting a file is to use the `rm` command, KNOME can infer that the user does not know this planfor relation between deleting files and the `rm` command. Finally, since the planfor relation is central to UNIX commands, KNOME can deduce that the user does not know `rm` in the sense that the user is not familiar with `rm` (see Section 2.3).

In UC, the initial deduction about the user's goal is made by UC's goal analysis component, PAGAN [WILE86]. PAGAN computes the user's plans and goals from the dialogue, and can handle phenomena such as indirect speech acts. In some cases, PAGAN utilizes KNOME's

model of the user's knowledge in deducing the user's goals. For example, if the user asks, "Do you know how to compact a file?", then the choice between the direct interpretation (the user wants to know if UC knows how to compact a file) and the indirect interpretation (the user wants to know how to compact a file) depends partly on whether or not KNOME believes the user already knows the **compress** command. If KNOME does believe that the user is likely to know **compress**, then that will bias PAGAN toward the direct interpretation that the user is testing UC (not uncommon), and vice-versa.

This process may seem circular, since PAGAN's analysis of the user's goal depends on KNOME, while KNOME's analysis of the user's knowledge depends on PAGAN. However, KNOME's initial evaluation of whether the user knows is only a prediction from a partial user model. This prediction does not determine PAGAN's interpretation, because PAGAN must also take into account other factors in its analysis. One such factor is the frequency of usage of the direct versus indirect interpretations. PAGAN is biased toward adopting the more frequent usage. Another factor is the context of the dialog. When UC is configured in knowledge acquisition mode as UCTeacher [MARJ85], then the direct interpretation (i. e. the user is testing UC's knowledge), is inherently more likely than the indirect interpretation. KNOME's beliefs about the state of the user's knowledge is only one factor in PAGAN's analysis, which then leads KNOME to its own inference about the state of the user's knowledge.

Besides questions on how to do something, Goal deductions apply to many other kinds of queries such as:

What is a directory?

- the user wants to know what is a directory (goal)
- the user does not know what is a directory
- the user does not know the directory concept

Can you tell me how to find out who is on the system?

- the user wants to know who is on the system (goal)
- the user does not know who is on the system

What does ruptime do?

- the user wants to know the effects of the ruptime command (goal)
- the user does not know the effects of the ruptime command
- the user does not know the ruptime command

All of the above queries represent Goal deductions, since PAGAN can deduce that the user wants to know some information. Based on this initial deduction, KNOME can infer that the user does not know the information that is sought.

Usage deductions are made when the user applies some plan/procedure correctly, or uses some specialized terminology correctly. Examples include:

I tried typing "rm foo," but I got the message, "rm: foo not removed."

- the user knows the rm command

How can I find out the inode of a file?

- the user knows the inode concept

I tried typing “del foo”, but I got the message “del: Command not found.”

→ the user knows the DOS del command

→ the user does not know the UNIX rm command

In the first example, KNOOME can deduce that the user knows the `rm` plan. In the second example, KNOOME deduces that the user knows the meaning of the term “inode.” This is the main type of deduction used in ITS in which the system observes the user at work solving problems. This type of deduction is suggested by [RICH83] for consulting situations.

In general, whenever the user uses specialized terminology correctly, KNOOME infers that the user must know the concept denoted by the terminology. In UC, terms include UNIX command names and operating system terminology such as “file protection”, “directory,” and “inode.” Checking that the user used the terms *correctly* is not very sophisticated in KNOOME. The first check is simply that the term must be used in an English sentence that UC can parse and understand. However, the fact that the user used a term in an understandable sentence does not automatically mean that the user knows the meaning of the term. For example, if the user asks, “What does `rm` do?”, KNOOME should not infer that the user knows the `rm` command. The same principle applies when the user states, “I don’t know the `rm` command.” In these types of sentences, KNOOME actually first infers that the user does know `rm` and then reverses its inference later based on the fact that the user has the goal of knowing the effects of `rm`, which is a Goal type inference. In cases when KNOOME has evidence both that the user knows something and that does not know it, KNOOME always assumes that the user doesn’t know, since it is always possible that the user just seems to know and in fact doesn’t know.

The last example is somewhat more complex than the previous examples. In this case, KNOOME first infers that the user knows the DOS `del` command. Then, since the DOS `del` command is used to delete files and the user mistakenly tried to use the `del` command, KNOOME can deduce that the user does not know the UNIX command for deleting files, that is, the `rm` command. In general, whenever the user tries to use a foreign command in UNIX, then KNOOME assumes that the user does not know the corresponding UNIX command. This correspondence is found by looking at the goal of the foreign command and finding a UNIX command that is a plan for the same goal.

Background deductions are made when the system learns some relevant fact about the user’s background. Examples include:

I am a fourth year computer science graduate student.

→ the user is a member of the 4TH-YEAR-CS-GRAD stereotype

→ the user is LIKELY to be an EXPERT

In my CS2 class, . . .

→ the user is a member of the CS2-STUDENT stereotype

→ the user is LIKELY to be a BEGINNER

In the first example, KNOOME infers that the user is a member of the 4TH-YEAR-CS-GRAD stereotype. This allows KNOOME to further infer that the user is LIKELY to be an expert, because most 4th year CS graduate students are experts. In the second example, UC’s Concretion mechanism infers from “my CS2 class” that the user is a CS2-STUDENT. Based on this, KNOOME then infers that the user is a member of the CS2-STUDENT stereotype, and so is likely to be a beginner.

To make Background deductions, KNOME uses a collection of rules, one for each stereotype related to the user expertise levels. For example, the rule for CS2-STUDENT is: if the user is a CS2-STUDENT, the the user is LIKELY to be a BEGINNER.

Query-Reply deductions are made after the system asks whether or not the user knows some fact. The user's answer usually provides information about whether or not the user knows the fact. For example, if a system were to ask the user, "Do you know the rcp command?" and the user answers "Yes," then the system can infer that the user knows the rcp command. This type of deduction has not been implemented in KNOME, since the current version of UC does not ask the user whether the user knows particular facts.

No-Clarify deductions show how a system might obtain information based on what the user does *not* say. If the system uses some terminology that the system is not sure of the user knows, then the system can infer that the user does know this terminology provided that the user does not ask for a clarification and the meaning of the terminology is not evident from the context. Such deductions are more complex than the previous ones, since they require that the system keep track of the conversation. This type of deduction has not been implemented in KNOME, since UC tries to avoid the use of terminology unfamiliar to the user.

In KNOME, deductions are made using a rule-based system. The actual rules were shown in Table II, with the exception of the Background type of inference. The Background type inference is actually implemented by a collection of rules. For every type of user background that gives some clue about the user's level of expertise in using UNIX, there is a rule. Since there are potentially many types of user backgrounds that are relevant, there will be a corresponding number of such rules. For example, if the user is a CS2 student, then that is a relevant piece of background information which provides some clue about user's level of expertise in using UNIX. On the other hand, the fact that the user is a scuba diver, likes blueberry pies, or is a bachelor are irrelevant. The particular Background rules that have been implemented in KNOME are shown in Table III, which also lists the auxiliary inferences which support the major inferences that were shown in Table II.

user is a fourth year CS graduate student	→	user is likely to be an expert
user is a CS2 student	→	user is likely to be a beginner
user does (not) know ?x, where ?x is a description, and ?y is the referent of ?x	→	user does (not) know ?x
user does (not) know a planfor of command ?x	→	user does (not) know ?x
user does (not) know the effects of command ?x	→	user does (not) know ?x

Table III. Other rules used to infer what users know.

Such rules are implemented using *if-detected daemons* [CHIN87], which are daemons that detect specific configurations of knowledge in UC's KODIAK knowledge bases. For example, the first rule shown above would normally fire after PAGAN has inferred that the user wants to know something. Then the action part of the rule allows KNOME to infer that the user does not know some fact.

These classes of deductions were found to be important in the domain of a natural language consultation system. Other domains may provide other clues as to what a user knows. For example, in a domain where the user provides speech input, a system may be able to make deductions about the user's knowledge when the user mispronounces technical words.

3.4. Combining Evidence

After collecting evidence that the user does or does not know certain facts, the system must combine the evidence to determine the user's level of expertise. Facts are particular pieces of information about what the user does or does not know as described in the previous section. Note that a single user statement or query may produce several facts. Each unique fact is treated equally in determining the user's level of expertise. When KNOME deduces that the user does or does not know something more than once (perhaps from different user statements), only the first such fact is used in deducing the user's level of expertise. The rest only reconfirm the first fact, so they do not contribute to determining the user's level of expertise.

At the start of a session, KNOME has very little idea about the level of expertise of the user. KNOME's beliefs about the user's level is encoded as ratings of the likelihood that the user is a member of a candidate stereotype category. As evidence is gathered during the dialog, KNOME continuously adjusts its ratings concerning the likelihood that the user might be a member of any particular stereotype. Eventually, KNOME gathers enough evidence to peg the expertise level of the user. Typically, this takes about three interchanges. After making this decision, KNOME does not change its estimation of the user's level. This works well in UC where typical sessions are short and it is advantageous to quickly guess the user's level of expertise. A more flexible approach is to keep a running account of likelihoods as in SC-UM [NESS87], a user modeling system that is based on the methodology demonstrated in KNOME. However, a system that does not commit to a particular interpretation of the user's level of expertise cannot avoid the need to store information that can be predicted from the user's stereotype. This becomes inefficient as more is learned about the user.

There are two types of conclusions that may be drawn from any particular fact about what the user knows or does not know. First, the evidence may be enough to eliminate some stereotypes from consideration. An example is when the user does not know a simple fact such as the `rm` or `ls` commands. In these cases, KNOME can rule out the possibility that the user may be an intermediate or an expert, since all intermediates and experts know all simple facts (see Table I). So, if the user does not know a simple command like `rm` or `ls`, then the user could not possibly be more advanced than a beginner.

Even when the evidence is not enough to rule out a category, the system can still infer that the category is either more likely or less likely. KNOME does this by increasing or decreasing the likelihood rating of candidate categories.

Likelihood ratings are combined using the following linear scale: FALSE, VERY-UNLIKELY, UNLIKELY, SOMEWHAT-UNLIKELY, UNCERTAIN, SOMEWHAT-LIKELY, LIKELY, VERY-LIKELY, TRUE. The UNCERTAIN rating is neutral. It implies that the system does not believe that membership in that category is either likely or unlikely. Likelihood ratings combine linearly according to the scale. For example, a rating of LIKELY when combined with another LIKELY rating produces a TRUE rating. Also a SOMEWHAT-LIKELY rating combined with a VERY-UNLIKELY rating produces an UNLIKELY rating.

To see why KNOME might change a likelihood rating, consider what happens when KNOME determines that a user does not know some mundane level fact. In this case, KNOME infers that the user cannot be an expert, since experts know all mundane facts. Also, KNOME considers it somewhat less likely that the user might be an intermediate, since intermediates

know most mundane level facts. Similarly, the user is considered somewhat more likely to be a beginner, since beginners know a few mundane facts. Finally, since the stereotypical novice does not know any mundane facts, the user is considered more likely to be a novice.

Table IV shows the basis for these deductions. Given that a user does or does not know a fact of some difficulty level, and given that a particular user stereotype knows none/afew/most/all of the facts of that difficulty level, then KNOME can deduce that the user belongs to that stereotype with the additional likelihood shown in Table IV.

Stereotype knows Difficulty Level	Likelihood (user \in stereotype)	
	user does know fact	user does not know fact
NONE	FALSE	LIKELY
A FEW	SOMEWHAT-UNLIKELY	SOMEWHAT-LIKELY
MOST	SOMEWHAT-LIKELY	SOMEWHAT-UNLIKELY
ALL	LIKELY	FALSE

Table IV. Deductions when user does (not) know some fact.

Combining Table IV with Table I, yields Tables V and VI. Given the particular stereotypes used in KNOME, these two tables show the deductions that are made when KNOME determines that a user does (Table V) or does not (Table VI) know some fact that has the difficulty level indicated. Deductions consist of eliminating categories (FALSE) or increasing/decreasing the likelihood ratings of categories. Since these two Tables are easily derivable from the previous two, KNOME does not actually store the information of these two Tables internally. Table I is represented declaratively as a KODIAK network that is accessible to the rest of UC. Only Table IV is stored internally in KNOME. This Table is used when needed to derive the likelihood rating differences shown in Tables V and VI.

User Stereotype	Difficulty Level of Fact			
	simple	mundane	complex	esoteric
novice	SOMEWHAT-UNLIKELY	FALSE	FALSE	FALSE
beginner	SOMEWHAT-LIKELY	SOMEWHAT-UNLIKELY	FALSE	—
intermediate	LIKELY	SOMEWHAT-LIKELY	SOMEWHAT-UNLIKELY	—
expert	LIKELY	LIKELY	SOMEWHAT-LIKELY	—

Table V. Deduction when user knows some fact.

User Stereotype	Difficulty Level of Fact			
	simple	mundane	complex	esoteric
novice	SOMEWHAT- LIKELY	LIKELY	LIKELY	SOMEWHAT- LIKELY
beginner	SOMEWHAT- UNLIKELY	SOMEWHAT- LIKELY	LIKELY	—
intermediate	FALSE	SOMEWHAT- UNLIKELY	SOMEWHAT- LIKELY	—
expert	FALSE	FALSE	SOMEWHAT- UNLIKELY	—

Table VI. Deduction when user does not know some fact.

When the likelihood rating of a stereotype reaches TRUE, it is selected as the user’s category. The user’s category can also be deduced by elimination. Candidate stereotypes are eliminated when their likelihood drops to FALSE. When all but one candidate have been eliminated, the remaining candidate is selected. During the period before the final decision, KNOME works under the provisional assumption that the user belongs to the category with the highest current likelihood. In the case of a tie (i. e. when there are two or more candidate categories with the same highest likelihood rating), the lower level of expertise is chosen to represent the user temporarily. Using the lower level helps prevent KNOME from overestimating the user before KNOME is sure about the user’s level of expertise. Underestimating the user’s level may result in more verbose descriptions, but does not lead KNOME to leave out information that the user may not know. Also, inferences that are made based on these provisional stereotype assumptions are marked as having greater uncertainty.

3.5. Examples

To see how KNOME deduces a user’s level of expertise, consider the following traces from the UC sessions shown in Figures 5 and 6. At the start of either session, UC does not have any information about either user. In such cases, UC starts with a slight bias toward believing that the user is a beginner, because more UC users tend to be beginners. Hence the likelihood rating for the user being a beginner is SOMEWHAT-LIKELY, whereas the likelihood ratings for the user being a novice, intermediate or expert are all UNCERTAIN (the neutral likelihood rating).

In Figure 5, KNOME deduces that the user is a novice, because the user does not know either the `more` or `lpr` commands, which are both simple commands, nor the `rwho` command, which is of mundane difficulty. When KNOME first deduces that the user does not know the `more` command, KNOME can draw a number of inferences based on the fact that the user does not know a simple command. Since experts and intermediates know all simple commands (see Table I), KNOME rules out the possibility that the user might be an intermediate or an expert. Also, since beginners know most simple facts, KNOME lowers the likelihood rating for the user being a beginner from SOMEWHAT-LIKELY to UNCERTAIN. Likewise, since novices know only a few simple facts, KNOME raises the likelihood rating for the user being a novice to SOMEWHAT-LIKELY.

When KNOME next determines that the user does not know the simple command, `lpr`, KNOME makes additional inferences. KNOME raises the likelihood rating of the user as a novice to LIKELY and lowers the likelihood rating of the user as a beginner to SOMEWHAT-

UNLIKELY. Finally when KNAME finds that the user does not know the mundane command `rwho`, the likelihood rating for the user being a novice rises from LIKELY to TRUE. TRUE is the threshold (nothing can have a higher likelihood than TRUE), so KNAME concludes that this user is a novice.

Welcome to UC (Unix Consultant) version 3.23
To a '#' prompt, please type in your questions
about the UNIX file system in English.
To leave, just type '^D' or '(exit)'.

Hi.
How can I help you?

What will display a file on my terminal?

KNAME: Asserting *USER* does not know ACTION12?

KNAME: Asserting *USER* does not know PLANFOR40

KNAME: Asserting *USER* does not know UNIX-MORE-COMMAND0
KNAME: Since UNIX-MORE-COMMAND0 is a UNIX-MORE-COMMAND,
asserting *USER* does not know UNIX-MORE-COMMAND
KNAME: UNIX-MORE-COMMAND has difficulty SIMPLE, so deducing:
KNAME: ruling out *USER* = INTERMEDIATE
KNAME: ruling out *USER* = EXPERT
KNAME: *USER* is SOMEWHAT-LIKELY to be NOVICE
=> likelihood(*USER* = NOVICE) = SOMEWHAT-LIKELY
KNAME: *USER* is SOMEWHAT-UNLIKELY to be BEGINNER
=> likelihood(*USER* = BEGINNER) = UNCERTAIN

Use more.
For example, to display the file named foo, type 'more foo'.

How can I print a file on the lineprinter?

KNAME: Asserting *USER* does not know UNIX-DLPR-COMMAND0
KNAME: Since UNIX-DLPR-COMMAND0 is a UNIX-DLPR-COMMAND,
asserting *USER* does not know UNIX-DLPR-COMMAND
KNAME: UNIX-DLPR-COMMAND has difficulty SIMPLE, so deducing:
KNAME: *USER* is SOMEWHAT-LIKELY to be NOVICE
=> likelihood(*USER* = NOVICE) = LIKELY
KNAME: *USER* is SOMEWHAT-UNLIKELY to be BEGINNER
=> likelihood(*USER* = BEGINNER) = SOMEWHAT-UNLIKELY

Use lpr.
For example, to print the file named foo, type 'lpr foo'.

What does rwho do?

KNAME: Asserting *USER* does not know STATE11?

KNAME: Asserting *USER* does not know HAS-EFFECT23

KNAME: Asserting *USER* does not know UNIX-RWHO-COMMAND0
KNAME: Since UNIX-RWHO-COMMAND0 is a UNIX-RWHO-COMMAND,

```
asserting *USER* does not know UNIX-RWHO-COMMAND
KNOME: UNIX-RWHO-COMMAND has difficulty MUNDANE, so deducing:
KNOME: asserting *USER* = NOVICE
```

**rwho is used to list all users on the network, list their tty,
and list their login time.**

Figure 5. UC session with an novice.

In Figure 6, KNOME deduces that the user is an intermediate. First, KNOME deduces that the user knows what an inode is, since the user mentions “the inode of a file.” Since knowing about inodes is a complex fact, KNOME is able to eliminate the novice and beginner categories as possibilities. KNOME also lowers the likelihood rating of the user as an intermediate and raises the likelihood rating of the user as an expert. However, these changes in likelihood ratings are canceled out by KNOME’s deduction from the same user query that the user does not know the `ls -i` command, which is of complex difficulty. Next, KNOME deduces that the user does not know the `rwho` command, which is of mundane difficulty. This allows KNOME to eliminate the possibility that the user might be an expert, leaving KNOME to conclude that the user is an intermediate, since that is the only possibility left.

```
Welcome to UC (Unix Consultant) version 3.23
To a '#' prompt, please type in your questions
about the UNIX file system in English.
To leave, just type '^D' or '(exit)'.
```

```
Hi.
How can I help you?
```

```
# Can you tell me how to find out the inode of a file?
```

```
KNOME: Asserting *USER* knows INODE1
KNOME: Since INODE1 is a INODE,
asserting *USER* knows INODE
KNOME: INODE has difficulty COMPLEX, so deducing:
KNOME: ruling out *USER* = NOVICE
KNOME: ruling out *USER* = BEGINNER
KNOME: *USER* is SOMEWHAT-UNLIKELY to be INTERMEDIATE
=> likelihood(*USER* = INTERMEDIATE) = SOMEWHAT-UNLIKELY
KNOME: *USER* is SOMEWHAT-LIKELY to be EXPERT
=> likelihood(*USER* = EXPERT) = SOMEWHAT-LIKELY
```

```
KNOME: Asserting *USER* does not know ACTION15?
```

```
KNOME: Asserting *USER* does not know PLANFOR390
```

```
KNOME: Asserting *USER* does not know UNIX-LS-i-COMMAND0
KNOME: Since UNIX-LS-i-COMMAND0 is a UNIX-LS-i-COMMAND,
asserting *USER* does not know UNIX-LS-i-COMMAND
KNOME: UNIX-LS-i-COMMAND has difficulty COMPLEX, so deducing:
KNOME: *USER* is SOMEWHAT-LIKELY to be INTERMEDIATE
```

```
=> likelihood(*USER* = INTERMEDIATE) = UNCERTAIN
KNOME: *USER* is SOMEWHAT-UNLIKELY to be EXPERT
=> likelihood(*USER* = EXPERT) = UNCERTAIN

Type 'ls -i'.

# What does rwho do?

KNOME: Asserting *USER* does not know UNIX-RWHO-COMMAND0
KNOME: Since UNIX-RWHO-COMMAND0 is a UNIX-RWHO-COMMAND,
      asserting *USER* does not know UNIX-RWHO-COMMAND
KNOME: UNIX-RWHO-COMMAND has difficulty MUNDANE, so deducing:
KNOME: ruling out *USER* = EXPERT
KNOME: only one candidate left, so asserting *USER* = INTERMEDIATE

rwho is like who, except rwho is for all users on the network.
```

Figure 6. UC session with an intermediate.

Even before KNOME has completely determined the expertise level of the user, KNOME's partial user model is still useful to the other components of UC. In the dialog with the novice, UC chooses to provide the user with examples of command-formats based on KNOME's initial deduction that the user is SOMEWHAT-LIKELY to be a novice. On the other hand, UC does not provide examples of simple command-formats to the intermediate user who presumably would already know such simple formats. Also, UC is able to use a simile to explain to the intermediate user the *rwho* command in terms of the *who* command. The novice user was given a complete explanation of *rwho*, since KNOME believed that the user would not know the *who* command and so would not understand the simile.

4. Modeling UC's Knowledge

Besides modeling what users know, KNOME also models what UC itself knows. Such a model allows KNOME to differentiate between cases where UC lacks knowledge and where the user has a

4.1. Open vs. Closed World Models

An important problem in AI knowledge bases is to define the limitations of the knowledge base. One solution is to use a closed world model, which states that the knowledge base knows everything (see [REIT78]). In a closed world model, anything that is neither in the knowledge base nor deducible from the knowledge base is deemed false. Such a model works only for completely self-contained micro-worlds such as airline reservation systems that know all airline flights. For real world applications, a closed world model is untenable, since real world knowledge bases cannot know everything even in just their own area of expertise. So a system with a closed model would be prone to giving out false information such as claiming that objects that the system did not happen to know about do not exist, claiming that actions that the system could not figure out cannot be done, etc.

The other extreme is an open world model, which states that anything that is neither in the knowledge base nor directly deducible from the knowledge base is not known. This model has a significant problem with ruling out things that do not exist. For example, a knowledge base might list all the participants in any particular relation. Using a closed world model, the system would know that these are the only participants of that relation. However, in an open world model, the system cannot rule out that there might not be other participants unless the knowledge base explicitly encodes for all other possible participants that they do not participate in that relation. Since there are usually many more such negative facts than positive facts, encoding such facts in a knowledge base is extremely inefficient.

Neither model is quite adequate. A closed world model allows a system to rule out spurious hypotheses easily, but is prone to ruling out true facts that were not in the knowledge base. An open world model allows the system to be non-committal about hypotheses that are not covered by its knowledge base, but does not allow it to rule out spurious hypotheses when the system *does* have complete information. The ideal system needs to combine the best of both models. It needs to be assertive in ruling out spurious hypotheses when the system does have complete information, yet remain non-committal when it does not have complete information. To do this, a system needs to know the limitations of its own knowledge.

4.2. Meta-Knowledge

UC uses an open world model augmented by knowledge about the coverage limitations of the knowledge base. By using an open world model, UC is able to remain non-committal about information that is not in its knowledge base. This allows UC to profess ignorance about things outside its area of expertise. On the other hand, when UC does have complete information about an area, this is explicitly encoded in KNAME's model of UC's knowledge. In a sense, this is a model of what UC itself knows. Such knowledge is called *meta-knowledge*.

The term meta-knowledge was previously used for AI knowledge bases by [BARR77, DAVI77]. [BARR77] argues that AI knowledge bases need meta-knowledge, i. e. the system's knowledge about "the extent and limit of available knowledge, what facts are relevant in a given situation, how dependable they are, and how they are to be used." [DAVI77] describes meta-knowledge in the TEIRESIAS system and its use in knowledge acquisition and in selecting which rules to try applying first through the use of *meta-rules*. [SMID83] use meta-level reasoning to find all of the solutions to a problem. Although they use a closed world assumption, they suggest that other systems for which the closed world assumption is not valid can encode limits on the number of solutions and use these limits to cut off inference. Such limits can be considered a type of meta-knowledge.

In KNAME, meta-knowledge is not primarily used for the same purposes as in TEIRESIAS; rather it is primarily used in dealing with user misconceptions. So KNAME's meta-knowledge describes the coverage limitations of UC's knowledge base. Chapter III, Section 4 describes how user misconceptions are detected and how meta-knowledge is used in correcting the user's misconceptions.

4.3. Representation of Meta-Knowledge

Meta-knowledge in KNAME is represented explicitly as a collection of KODIAK statements about what UC knows. Examples of meta-knowledge include the fact that UC knows that it knows all file manipulation commands and the fact that UC knows all possible side effects for all known simple commands, most known mundane commands, and a few known complex commands. The fact that UC does not know all of the side effects for known commands is due to not

enough programming and the knowledge limitations of UC's programmers rather than any inherent limitation of UC.

Figure 7 shows the representation of KNOME's meta-knowledge about simple commands. KNOW1 states that UC knows all of the effects of simple commands, and KNOW2 represents the fact that UC knows all of the options of simple commands.

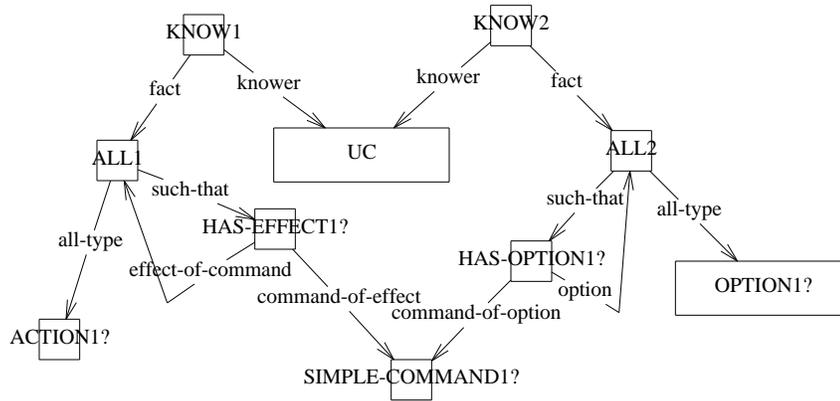


Figure 7. KODIAK representation of some of UC's meta-knowledge.

Besides the explicit meta-knowledge modeled by KNOME, UC also has meta-knowledge that is represented implicitly within the KODIAK representation language. For example, any aspectual of a relation can be constrained to be filled by only certain classes of objects. This effectively constrains the way in which different objects may be related to other objects in KODIAK. Of course, these constraints are sometimes violated when users speak metaphorically.

4.4. Other Approaches to User Misconceptions

Other systems that have dealt with user misconceptions use one of two common approaches. The first approach compiles an a priori list of possible user misconceptions for use in detecting misconceptions. This is common among ITS (Intelligent Tutoring Systems) (e. g. [BROW78, STEV79, SLEE81, JOHN84, REIS85, ANDJ85b]). These systems are not very flexible, since they cannot deal with any user misconceptions that are not listed among their precompiled a priori lists.

Another approach attempts to detect classes of misconceptions. Among the best of these is Kaplan's CO-OP natural language database-query system [KAPS83], which handles a class of faulty user presumptions. An example from CO-OP is shown in Figure 8.

User: Who advises projects in area 36?
CO-OP: I don't know of any area #36.

Figure 8. CO-OP example with hedged response to misconception.

The user in the CO-OP example believes that there is an area 36. In processing the user's query, CO-OP finds that there is no area 36 listed in its database, so it detects a possible user

misconception: the user may mistakenly presume that there is an area 36 when in actuality there might not be an area 36. Note that CO-OP's reply does not actually *correct* the user's misconception, rather it *hedges*, claiming that it does not know of any area #36. By hedging, CO-OP cannot correct the user's misconception, but can only *suggest* to the user that the user might be mistaken. CO-OP is unable to take corrective action, because it does not assume either an open or closed world model for its database. As a result, it cannot tell from the fact that there is no area 36 in its database whether there really is no area 36, or if area 36 was just left out of its database. Unlike UC, in which the KNOME subcomponent models UC's own knowledge with meta-knowledge (see Chapter II, Section 4),

A corrective response is shown in Kaplan's example of a cooperative human, shown in Figure 9.

User: Which students got a grade of F in CS105 in Spring 1980?
Human: CS105 was not given in Spring 1980.

Figure 9. Kaplan's example of a cooperative human respondent.

Because CO-OP does not adopt either an open or closed world model, it would not be able to deny that CS105 was given in Spring 1980 as in Kaplan's example of a cooperative human. CO-OP would only be able to suggest a misconception by saying something like, "I don't know of any CS105 in Spring 1980." To be able to actually correct the user, CO-OP would need to know that the absence of a CS105 among the Spring 1980 courses implies that there was no CS105 offered then. That is, CO-OP would need meta-knowledge. Since CO-OP was designed to be easily transportable among different databases, it did not have such non-transportable meta-knowledge.

The DAIQUIRY module of HAM-ANS [MARB87] extended the misconception approach of CO-OP to include conceptual failures in which the user asks about objects not modeled by the database. Because DAIQUIRY, like CO-OP, did not have meta-knowledge, it too could only hedge its replies to the user. For example, it would reply, "The system has no knowledge about 'SHIP'," when SHIP was not in its conceptual hierarchy.

[MAYS80b, WEBB83] report efforts to deal with user misconceptions that can be found by constraint violations on relations. For example, if the TEACH relation is constrained to relate only FACULTY to COURSES, then the system can detect a misconception if the user asks about UNDERGRADUATES that TEACH. This type of relational constraint can be considered a kind of implicit meta-knowledge. It differs from the explicit meta-knowledge of KNOME in that relational constraints can only be used to detect misconceptions about what can be the case, and not misconceptions about what is the case. For example, relational constraints can be used to deduce that only commands can have command-options, but not whether any particular command has any particular command-option.

After a user misconception has been detected, it must be corrected. [MCCO85a, MCCO*] discuss strategies for correcting object-oriented misconceptions. [QUIL*] discusses correcting plan-oriented misconceptions in AQUA. AQUA assumes a closed world model for its knowledge base of plans, called an "advisor model." As a result of this assumption, AQUA can only model perfect advisors that have complete knowledge of the domain. If AQUA's advisor model were incomplete, then AQUA would tend to mislead the user. For example, if an obscure side effect of a plan were inadvertently left out of AQUA's knowledge base, then AQUA would mislead the

user to believe that the plan did not have this a side-effect. Some form of meta-knowledge would be required to allow AQUA to model an advisor without perfect knowledge of the domain.

4.5. Detecting Misconceptions in UC

User misconceptions are detected by UC during the processing of the user's query. Currently, UC only handles relational misconceptions, that is, misconceptions in which the user believes a relation holds between two objects when in fact, such a relation cannot hold or does not happen to hold between those particular objects. UC does not handle object-oriented misconceptions, such as those for which [MCCO85a] discusses correction strategies.

In processing the user's query, UC checks to see whether all relations mentioned by the user in the user's query have a counterpart in UC's knowledge base. For example, if the user asks, "What does `ls -e` do?", then UC's parser/understander understands part of this as a HAS-OPTION relation relating an instances of the `ls` command and an instance of the `-e` option. While looking for the effects of the command, UC finds that there is no corresponding HAS-OPTION relation between the `ls` command and the `-e` option in UC's knowledge base. At this point, UC suspects that the user may have a misconception.

In order to see whether the user actually has a misconception, or whether UC just lacks information about this particular option of this command, UC checks with the KNOOME subcomponent, which models the limitations of UC's knowledge base using meta-knowledge (see Chapter II, Section 4). If KNOOME has meta-knowledge that UC knows all of the options of this command, then UC knows that the user has a misconception. On the other hand, if the options of this command are not covered by KNOOME's meta-knowledge, then UC cannot assume that the user has a misconception. Instead, UC must assume that it lacks information about this command/option combination.

4.6. Correcting Misconceptions

After UC has determined either that the user has a misconception or that UC lacks knowledge, UCEgo decides how UC should reply to the user. When the user has a misconception, UCEgo tries to correct this misconception by denying that what the user mistakenly believes is the case. On the other hand, if UC determines that it lacks information, then UCEgo apologizes to the user for not knowing the answer to the user's query. These two types of responses are shown in the UC session of Figure 10.

In the first query, UC corrects the user's misconception that `who` has a `-b` option. It does this by first noticing that the user's usage of "`who -b`" translates into a HAS-OPTION relation between a UNIX-WHO-COMMAND and a `-b`-OPTION. There is no equivalent HAS-OPTION relation in UC's knowledge base, so UC suspects a possible user misconception. To see if this is the case or if UC just lacks knowledge about this particular option, UC consults the meta-knowledge stored in KNOOME. The appropriate meta-knowledge in this case is the fact that UC knows all the options of all simple commands. Since `who` is a simple command, and since the `-b`-OPTION is not listed among the options of UNIX-WHO-COMMAND in UC's knowledge base, KNOOME can infer that there is no such option for `who`. Next, UCEgo decides that UC should correct the user's misconception by denying the existence of a `-b` option for `who`.

On the other hand, in the second query, UC professes ignorance about the `-t` option of the `ruptime` command. As in the previous query about `who`, UC detects a possible misconception when it does not find a `-t` option listed for `ruptime` in its knowledge base. However, in this case, `ruptime` is a complex command, so the previous meta-knowledge does not apply. There is no meta-knowledge about the options of complex commands (due to not enough programming by

```
Welcome to UC (Unix Consultant) version 3.23
To a '#' prompt, please type in your questions
about the UNIX file system in English.
To leave, just type '^D' or '(exit)'.
```

```
Hi.
```

```
How can I help you?
```

```
# What does who -b do?
```

```
who does not have a -b option.
```

```
# What does ruptime -t do?
```

```
I'm sorry, I do not know that.
```

Figure 10. UC session showing different UC responses to possible misconceptions.

UC's implementors rather than any inherent limitation of UC), so UC cannot tell if `ruptime` has a `-t` option. In order to be polite, UCExpress apologizes to the user for not knowing.

5. Exploiting KNOOME

KNOOME is used by several different components of UC. This section will provide a survey of these applications, since a detailed analysis is beyond the scope of this paper.

One application of KNOOME is in goal analysis (see [WILE86]). KNOOME can be used to select between different interpretations of the user's statements. For example, consider the user's query "Do you know how to delete a file?" There are two possible interpretations of this query: a direct interpretation where the user wants to know if UC knows how to delete a file; and an indirect speech act interpretation where the user wants to know how to delete a file. If the goal analyzer knows that the speaker already knows how to delete a file, then the system can rule out the indirect interpretation. Similarly, if the system believes that the user does not know how to delete a file, then the indirect interpretation is more likely.

Another use for the user model is to keep UC from telling the user something that the user already knows, and to express information in terms of concepts with which the user is already familiar. For example, in expressing the answer, the UCExpress component of UC prunes those parts of the answer that the user already knows. So in the following UC dialog, UCExpress gives an example of the format of the `vi` command to a novice, since the novice likely does not know the simple-file-format. To beginners and other non-novices, UCExpress does not provide the example, since UC can assume that the user will know that to use `vi`, one types "vi" followed by the name of the file to edit.

User: How can I edit a file?

UC to novice: Use vi.
For example, to edit the file named foo, type 'vi foo'.

UC to beginner: Use vi.

KNOME is also used by UCExpress when determining whether or not to use an analogy in explaining the usage of a command. Consider the following UC dialog. UCExpress needs to know that the user already knows the uptime command before it can choose to use the analogy.

User: What does ruptime do?

UC: ruptime is like uptime, except ruptime is for all machines on the network.

UCEgo [CHIN87], the agent component of UC, uses KNOME when deciding whether or not to provide additional information that the user may need (c. f. [JOSH84a, JOSH84b] for other theoretical approaches). For example, in the UC dialog shown in Figure 11, UCEgo does not actually answer the user's question.

User: Who is on the system?

UC: To find out who is on the system, type 'who'.

Figure 11. UC volunteers information that the user does not know.

UC does not actually tell the user who is on the system (e. g. by saying "John, Mary, and David are on the system"). Instead, UCEgo decides to tell the user how to find out this information, namely use the `who` command. However, if the user already knows the `who` command, then UC's answer would be superfluous. So UCEgo first consults KNOME to see if the user lacks such knowledge before providing the additional information.

6. Conclusion

6.1. Summary

The user modeling system of KNOME is extremely simple, yet it provides enough functionality for the needs of UC. KNOME can infer what the user knows from a consultation dialog with the user. Based on only a few such facts about what a user does or does not know, KNOME can also *predict* many other default facts about what the user might or might not know. KNOME's stereotypes serve to efficiently group those default inferences that commonly occur. Using only a few such stereotypes, KNOME is able to provide enough differentiation among potential users so that UC can use these differences in answer expression, goal analysis,

and detection of situations when the user lacks knowledge or has misconceptions. The double-stereotype system is not only space efficient, but also allows easy addition of knowledge to KNOOME. When a new fact is added to UC, the knowledge engineer needs only to classify the new fact according to its difficulty level.

Although no attempt has been made yet to apply the methodology to other areas, it would seem that many of the techniques should transfer well. Stereotype categories are used in UC in similar fashion to the way humans use categories, that is as reference points for reasoning about individual members (see [ROSC77, ROSC78, ROSC83]). Thus stereotypes should be useful for approximate reasoning wherever human categorization has proven to be useful. The double-stereotype system extends the use of computer stereotype categories to cover more of human categorization, that is to include categories for knowledge as well as categories/stereotypes for people/users. The double-stereotype system can be applied straightforwardly to other domains where a computer system needs to model the relation between different classes of users and different classes of information. In cases where the user population is homogeneous or where the information is homogeneous, then a double-stereotypes system would not be applicable.

The methodology used to deduce the user's stereotype in KNOOME transfers in two ways to other domains. First, the method used to infer individual facts about what the user knows is applicable to any natural language system that needs to infer facts about the user's knowledge based on what the user says. On the other hand, the methodology used to combine such evidence is highly dependent on a double-stereotype system. However it can be used in any double-stereotype system where the system can deduce particular relations between the user's stereotype and the modeled information.

6.2. Current Status

KNOOME is currently implemented in FRANZ LISP and Common LISP in a version of KODIAK [WILE87], which is a network style knowledge representation language featuring objects and relations, multiple inheritance, and other epistemological tools.

6.3. Problems

Currently, KNOOME does not address the problem of whether the user knows something after UC has informed the user. The difficulty is that the user may not understand the system's presentation, in which case the user modeling system should not infer that the user knows the new information. A drawback of systems limited to natural language interactions is that these systems cannot watch the user's reactions to deduce whether or not the user is having trouble understanding the system's presentation. Human consultants are able to switch explanation strategies midway through a presentation when they notice their clients' facial expressions showing confusion. A computer consultant would have to wait for the user to respond with language in order to get feedback. In such cases, the best that a user modeling system can do may be to assume that the user has understood (perhaps with a lower certainty factor), until the modeler gets evidence to the contrary. Then, if the user does not ask for a clarification in the next exchange, the modeler can increase the certainty of its belief that the user has indeed understood.

Another deficiency in KNOOME is that it does not contain very many specific rules for inferring what specific facts a user might know based on other specific facts that the user knows. For example, if a user knows the `rwho` command, then it is very likely that the user will also know the `who` command. KNOOME has avoided these types of explicit rules in favor of the more efficient, though less specific mechanism of double stereotypes. The double stereotypes allow KNOOME to combine a small number of explicit facts to first infer the user's level of expertise and

from this, the user's knowledge of large sets of other facts. Although this powerful general mechanism works well, there are still cases when explicit rules will give more specific information about what a user knows with a higher degree of certainty. The GUMS general user modeling shell [FINI*] makes good use of these explicit rules.

A possible deficiency with KNOME's method of combining evidence is that KNOME marks the user as belonging to a particular stereotype after only a short time (typically ~3 interchanges). After this, KNOME does not alter its conception of the user's level of expertise. This is reasonable in the UC domain, because UC sessions are typically short. However, this would not be acceptable for longer term user models. If a system needs to keep track of a user's level of expertise over periods long enough that the user's expertise may increase appreciably, then the system needs to be able to change its judgment of the user's level of expertise. This process is easier than one might suppose, since a user's level of expertise tends to grow monotonically (it seems reasonable to ignore the phenomenon of forgetting).

Currently, KNOME only has a single range of user expertise and a single range of knowledge difficulty. This is sufficient for UC, since UC only covers commands related to the UNIX file system. In order to extend UC to cover other areas of UNIX such as using the vi or emacs editors, KNOME would need additional ranges of user expertise levels and information difficulty levels. Problems which need to be addressed with multiple ranges include how the levels in one range might relate to the levels of other ranges. For example, a system may be able to predict that an expert in using vi is likely to be an expert in using the UNIX file system. On the other hand, there may be no such relations between familiarity with emacs and familiarity with the UNIX file system, since the emacs editor is common to many other operating systems.

Another area which KNOME does not address completely is how to model users of other operating systems. Such users may be able to transfer some amounts of expertise to UNIX. These users can benefit from analogies between UNIX and the operating system that they know. On the other hand, such users sometimes incorrectly apply analogies and often mix up commands. Adding additional stereotypes for such users would enhance KNOME's ability to provide proper analogies and to detect improper analogies or command usage. [MACM83] has shown that one user modeling system can adequately model different operating systems (UNIX, TOPS-20, and VM). However, no-one has investigated how knowledge of the user's expertise in one domain transfers to another domain (correctly or incorrectly) or how to exploit such knowledge (e. g. to provide analogies).

Other areas not addressed by KNOME include applying the techniques demonstrated in KNOME for other purposes such as selection of terminology in generation (c. f. [LEHM*]). The approach used in KNOME would also be useful in the selection of different explanation strategies as in the TAILOR system [PARI*].

7. References

[ALLE80]

Allen, J. F., Perrault, C. R. (1980). Analyzing Intention in Utterances. *Artificial Intelligence*, 15, pp. 143-178

[ANDJ85b]

Anderson, J. R., Boyle, F., G. Yost. (1985). The Geometry Tutor. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, 1, pp. 1-7, Los Angeles, CA, August

- [BARR77]
Barr, A. (1977). *Meta-knowledge and Memory*. Stanford University Heuristic Programming Project working paper HPP-77-37
- [BROW76a]
Burton, R. R., Brown, J. S. (1976). A Tutoring and Student Modelling Paradigm for Gaming Environments. In *Symposium on Computer Science and Education*, pp. 236-246, Anaheim, CA, February
- [BROW78]
Brown, J. S., Burton, R. R. (1978). Diagnostic Models for Procedural Bugs in Basic Mathematical Skills. *Cognitive Science*, 2, pp. 155-192
- [CARB83]
Carberry, S. (1983). Tracking User Goals in an Information-Seeking Environment. *Proceedings of the National Conference on Artificial Intelligence*, Washington, DC, August
- [CARB*]
Carberry, S. (1989). Plan Recognition and Its Use in Understanding Dialogue. In Kobsa, A. and Wahlster, W. (Eds.), *User Models in Dialog Systems*. Berlin, Heidelberg, New York, Tokyo: Springer-Verlag
- [CARR77]
Carr, B., Goldstein, I. (1977). *Overlays: A Theory of Modeling for Computer Aided Instruction*. Technical Report AI Memo 406, AI Laboratory, Massachusetts Institute of Technology, Cambridge, MA
- [CHIN83]
Chin, D. N. (1983). A Case Study of Knowledge Representation in UC. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*. 1, pp. 388-390. Karlsruhe, West Germany, August
- [CHIN86]
Chin, D. N. (1986). User modeling in UC, the UNIX consultant. *Proceedings of the CHI-86 Conference*, pp. 24-28, Boston, MA, April
- [CHIN87]
Chin, D. N. (1987). *UCEgo: An Intelligent Agent for the Unix Consultant*. Ph. D. thesis, Department of Computer Science, University of California, Berkeley.
- [DAVI77]
Davis, R. Buchanan, B. G. (1977). Meta-level knowledge: Overview and Applications. *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, Cambridge, MA
- [HOEP84b]
Hoepfner, W., Morik, K., Marburger, H. (1984). *Talking it Over: The Natural Dialog System HAM-ANS*. Research Unit for Information Science and Artificial Intelligence,

University of Hamburg, Report ANS-26

[JOHN84]

Johnson, W. L. and Soloway, E. (1984). Intention-Based Diagnosis of Programming Errors. *Proceedings of the National Conference on Artificial Intelligence*, pp. 162-168, Austin, TX, August

[JOSH84a]

Joshi, A., Webber, B. L. (1984). Living up to Expectations: Computing Expert Responses. *Proceedings of the National Conference on Artificial Intelligence*, pp. 169-175, Austin, Texas, August

[JOSH84b]

Joshi, A., Webber B. L., Weischedel, R. M. (1984). Preventing False Inferences. *Proceedings of Coling84*, pp. 134-138, Stanford, CA, July

[KAPS83]

Kaplan, S. J. (1983). Cooperative Responses from a Portable Natural Language Database Query System. In M. Brady and R. C. Berwick (Eds.), *Computational Models of Discourse*. Cambridge, MA: MIT Press

[KASS*]

Kass, R. (1989). Student Modelling in Intelligent Tutoring Systems — Implications for User Modelling. In Kobsa, A. and Wahlster, W. (Eds.), *User Models in Dialog Systems*. Berlin, Heidelberg, New York, Tokyo: Springer-Verlag

[KAYD85]

Kay, D. S., Black, J. B. (1985). The Evolution of Knowledge Representations with Increasing Expertise in Using Systems. *Proceedings of the Seventh Annual Conference of the Cognitive Science Society*, pp. 140-149, Irvine, CA, August

[KOBS85d]

Kobsa, A. (1985). VIE-DPM: A User Model in a Natural-Language Dialogue System. In Laubsch, J. (Ed.), *GWAI-84, 8th German Workshop on Artificial Intelligence*. Berlin: Springer

[KOBS85b]

Kobsa, A. (1985). *User Modelling in Dialogue Systems: Potentials and Hazards*. Report 85-01, University of Vienna, Department of Medical Cybernetics, Vienna, Austria

[KOBS86a]

Kobsa, A. (1986). Generating a User Model from Wh-questions in the VIE-LANG System. In P. Hellwig and H. Lehman (Eds.), *Trends in der Linguistischen Datenverarbeitung*. Hildesheim, West Germany: Olms Verlag

[LEHM*]

Lehman, J. F., Carbonell, J. G. (1989). Learning the User's Language: A Step Towards Automated Creation of User Models. In Kobsa, A. and Wahlster, W. (Eds.), *User Models*

in Dialog Systems. Berlin, Heidelberg, New York, Tokyo: Springer-Verlag

[LITM84]

Litman, D. J., Allen, J. F. (1984). A Plan Recognition Model for Clarification Subdialogues. *Proceedings of the Tenth International Conference on Computational Linguistics*, Palo Alto, CA, July

[MACM83]

Macmillan, S. A. (1983). *User Models to Personalize an Intelligent Agent*. Doctoral Dissertation, School of Education, Stanford University, Stanford, CA

[MATT86]

Matthews, M., Biswas, G. (1986). *USCSH: An Active Assistance Interface for Unix*. University of South Carolina technical report USC-CS TR 86-003, June 12

[MARB87]

Marburger, H. (1987). A Strategy for Producing Cooperative nl Reactions in a Database Interface. In *Proceedings of AIMSA-86*, Wana, Bulgaria

[MARJ85]

Martin, J. H. (1985). Knowledge Acquisition through Natural Language Dialogue. In *Proceedings of the 2nd Conference on Artificial Intelligence Applications*, pp. 582-586. Miami, FL

[MAYS80b]

Mays, E. (1980). Failures in Natural Language Systems: Applications to Data Base Query Systems. *Proceedings of National Conference on Artificial Intelligence*, pp. 327-330

[MCCO85a]

McCoy, K. F. (1985). *Correcting Object-Related Misconceptions*. Doctoral dissertation, University of Pennsylvania, Department of Computer and Information Science, Moore School. Available as report MS-CIS-85-57

[MCCO*]

McCoy, K. F. (1989). Highlighting a User Model to Respond to Misconceptions. In Kobsa, A. and Wahlster, W. (Eds.), *User Modelling in Dialog Systems*. Berlin, Heidelberg, New York, Tokyo: Springer-Verlag

[MORI85b]

Morik, K., Rollinger, C-R. (1985). The Real Estate Agent — Modeling the User by Uncertain Reasoning. *AI Magazine*, 6 (2), pp. 44-52, Summer

[MORI*]

Morik, K. (1989). User Models and Conversational Settings: Modeling the User's Wants. In Kobsa, A. and Wahlster, W. (Eds.), *User Models in Dialog Systems*. Berlin, Heidelberg, New York, Tokyo: Springer-Verlag

[NESS87]

Nessen, E. (1987). *SC-UM — User Modeling in the SINIX Consultant*. Memo, FR.10.2 Informatik IV, University of the Saarland, Saarbrücken, West Germany

[PARI*]

Paris, C. L. (1989). Tailoring Object Descriptions to a User's Level of Expertise. In Kobsa, A. and Wahlster, W. (Eds.), *User Models in Dialog Systems*. Berlin, Heidelberg, New York, Tokyo: Springer-Verlag

[QUIL*]

Quilici, A., Dyer, M., Flowers, M. (1989). Recognizing and Responding to Plan-Oriented Misconceptions. In Kobsa, A. and Wahlster, W. (Eds.), *User Models in Dialog Systems*. Berlin, Heidelberg, New York, Tokyo: Springer-Verlag

[REIS85]

Reiser, B., Anderson, J. R., Farrell, R. G. (1985). Dynamic Student Modelling in an Intelligent Tutor for Lisp Programming. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, 1, pp. 8-14, Los Angeles, CA, August

[REIT78]

Reiter, R. (1978). On Closed World Data Bases. In Gallaire, H. and Minker, J. (Eds.), *Logic and Data Bases*, New York, London: Plenum Press

[RICH79b]

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

[RICH83]

Rich, E. (1983). Users are Individuals: Individualizing User Models. *Int. Journal of Man-Machine Studies*, 18, pp. 199-214

[RICH*]

Rich, E. (1989). Stereotypes and User Modelling. In Kobsa, A. and Wahlster, W. (Eds.), *User Models in Dialog Systems*. Berlin, Heidelberg, New York, Tokyo: Springer-Verlag

[ROSC77] Rosch, E. (1977). Human Categorization. In Warren, N. (Ed.) *Studies in Cross Cultural Psychology*, I. London: Academic Press

[ROSC78]

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

[ROSC83]

Rosch, E. (1983). Prototype Classification and Logical Classification: The Two Systems. In Scholnick, E. (Ed.) *New Trends in Cognitive Representation: Challenges to Piaget's Theory*. Hillsdale, N. J.: Lawrence Erlbaum

- [SEAR75]
Searle, J. R. (1975). Indirect Speech Acts. In Cole, P. and Morgan, J. L. (Eds.), *Syntax and Semantics, Vol. III: Speech Acts* New York: Academic Press.
- [SLEE81]
Sleeman, D., Smith, M. J. (1981). Modelling Student's Problem Solving. *Artificial Intelligence*, 16, pp. 171-187
- [SLEE82c]
Sleeman, D., Brown, J. S. (1982). Editors of *Intelligent Tutoring Systems*. New York, NY: Academic Press
- [SMID83]
Smith, D. E., and Genesereth, M. R. (1983). *Finding All of the Solutions to a Problem: Serious Applications of Meta-Level Reasoning II*. Stanford University Heuristic Programming Project Memo HPP-83-21.
- [STEV79]
Stevens, A., Collins, A. Goldin, S. E. (1979). Misconceptions in Student's Understanding. *Intl. Journal of Man-Machine Studies*, 11, pp. 145-156
- [WAHL*]
Wahlster, W., Kobsa, A. (1989). Dialog-Based User Models. In Kobsa, A. and Wahlster, W. (Eds.), *User Models in Dialog Systems*. Berlin, Heidelberg, New York, Tokyo: Springer-Verlag
- [WEBB83]
Webber, B. L., Mays, E. (1983). Varieties of User Misconceptions: Detection and Correction. *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, 2, pp. 650-652, Karlsruhe, West Germany, August
- [WILE84]
Wilensky, R., Arens, Y., Chin, D. N. (1984). Talking to UNIX in English: An Overview of UC. *Communications of the ACM*, 27 (6), June
- [WILE86]
Wilensky, R., Mayfield, J., Albert, A., Chin, D. N., Cox, C., Luria, M., Martin, J., Wu, D. (1986). *UC — A Progress Report*. Computer Science Division, University of California, Berkeley, Report No. UCB/CSD 89/303.
- [WILE87]
Wilensky, R. (1987). *Some Problems and Proposals for Knowledge Representation*. Computer Science Division, University of California, Berkeley, Report No. UCB/CSD 87/351
- [ZADE65]
Zadeh, L. A. (1965). Fuzzy Sets. *Information and Control* 8, pp. 338-353.

[ZADE82]

Zadeh, L. A. (1982). *A Computational Approach to Fuzzy Quantifiers in Natural Language*. Computer Science Division, University of California, Berkeley, Memorandum No. UCB/ERL M82-36(Revised).

Table of Contents

1. Introduction	1
1.1. Some UC Examples	1
1.2. Problems in Modeling	3
1.3. Other User Models	4
2. Internal Representation of Users	5
2.1. Double-Stereotypes	5
2.2. Modeling Individual Users	7
2.3. Types of Knowing	8
2.4. Dealing with Uncertainty	8
3. Deducing the User's Level of Expertise	9
3.1. Other Model Acquisition Systems	9
3.2. KNOME's Approach	10
3.3. Collecting Evidence	11
3.4. Combining Evidence	16
3.5. Examples	18
4. Modeling UC's Knowledge	21
4.1. Open vs. Closed World Models	21
4.2. Meta-Knowledge	22
4.3. Representation of Meta-Knowledge	22
4.4. Other Approaches to User Misconceptions	23
4.5. Detecting Misconceptions in UC	25
4.6. Correcting Misconceptions	25
5. Exploiting KNOME	26
6. Conclusion	27
6.1. Summary	27
6.2. Current Status	28
6.3. Problems	28
7. References	29