An Automated Consultant for Interactive Environments

CUCS-391-88

Ursula Wolz*
Gail E. Kaiser+

Columbia University
Department of Computer Science
New York, NY 10027

Abstract

Interactive computing environments provide facilities intended to support and assist the range from novice to expert users, but casual users tend to get trapped in the starter set of commands. We have developed a rule-based technology for providing on-line assistance calibrated to both the task at hand and the user's past experience using the system. Such assistance helps users to progress to more advanced features. We present our automated consultant and describe its application to a practical domain, the Berkeley Unix™ mail system.

*Supported in part by ONR grant N00014-82-K-0256. +Supported in part by grants from AT&T Foundation, IBM, and Siemens Research and Technology Laboratories, in part by the New York State Center of Advanced Technology — Computer and Information Systems, and in part by a Digital Equipment Corporation Faculty Award.

keywords: Automated consulting, help systems, intelligent assistance, interactive computing environments, programming environments, user interfaces.

1. Introduction

Interactive computing environments such as mail systems and programming environments provide resources and facilities intended to support and assist users. A conflict arises between creating an environment simple enough for a novice, yet sophisticated enough to accommodate an expert. A common solution is to expose beginners to a set of starter commands, but provide more comprehensive features they can learn later. Finin [Finin 83] points out that many beginners get trapped in the starter set, since they are not encouraged to progress to more powerful commands. We have developed a solution to this problem: an automated consultant that answers a user's questions about the environment in a manner designed to provide this encouragement.

All interactive environments can be characterized as consisting of a set of functions with which a user can accomplish tasks specific to that environment. The environments themselves might be mail systems, VLSI design tools, word processing systems or programming environments. The means of access to the environment might include command languages, menus with keystroke or pointing devices, or even more sophisticated interfaces. But at the core, a set of functions must be executed as a *plan* (i.e., sequence of steps) to accomplish some *computational goal* of the user.

Although on-line and off-line documentation helps the user learn about the functions themselves, the complaint is often made that such documentation can be inadequate at providing specific 'goal-oriented' help for the task at hand. Furthermore, since most tutorials only scratch the surface of the capabilities of an environment, users tend to rely on the few commands they learn initially and never develop broad expertise with the system. Increasing one's expertise within an environment is often avoided because it tends to cut significantly into a user's short-term productivity. Yet in the long run taking the time to learn something new is likely to increase long-term productivity. Furthermore, with only limited expertise the quality of the resulting job is often diminished since the full potential of the environment is not exploited. Methods for encouraging users to master an environment could certainly be beneficial.

Whether an environment is intended for end users of commercial products or for software

development staff writing such systems, an automated consultant that can give appropriate help for the task at hand can increase user productivity and the quality of the job. The problem then is how to provide the appropriate information that neither swamps the novice (or casual user) with too much complex information nor insults the expert by providing an overly pedantic tutorial. The problem lends itself well to a solution using expert system techniques, namely how to choose and articulate appropriate information from a vast and complex knowledge base.

We are exploring a solution to this problem through the implementation of GECIE (Generated Explanations for Consulting in Interactive Environments — pronounced Jesse), a question answering system for Berkeley Unix Mail. GECIE that generates text based both on what the user is trying to do and what the user already knows how to do. We take a *user's goal-centered approach* in which the help given is a direct function of a user's needs within the current context. In particular, we focus on the content of the answer provided to a user.

First, we provide a small rule base that models a consultant's behavior and a large hierarchical knowledge represention that captures a consultant's domain knowledge. The domain knowledge includes explicit information about the relationships between the computational goals that can be accomplished in the environment, the plans used to accomplish them, and the functions that make up the plans. The rule base allows GECIE to reason about the actions associated with functions, but also allows it to analyze whether plans can satisfy goals and which of many equally good plans is most appropriate in a given context. In a mail system, for example, a goal might be to read a set of messages and forward a subset to a colleague. The plan for executing that goal will be dependent upon the particular functions available within the mail system.

Second, we believe that classifying functions, plans and goals according to level of expertise is inappropriate and global categorization of users as 'novice', 'intermediate' or 'expert' is inadequate. In our work, information on an individual's exposure to goals, plans and functions influences the *pedagogical goals* of the consultant, that is, what specific information it presents following a user's query. Expectations about what the user knows and should be told is based on the computational goals that user has satisfied in the past rather than on broad *ad hoc*

classifications of functions and plans as 'easy' or 'hard'. We exploit the structure of our knowledge base and use a goal-centered representation as a user model. Decisions about how to answer a user's question are based on an analysis of the match between the knowledge base and the user model. Taking another example from a mail system, a user may have extensive experience with sending simple messages to groups of users, and almost none with modifying messages through an editor. Such a user will not fall nicely into a categorization of expertise. A question relating to sending simple messages will require introducing very little new information into the discussion, while a question about modifying messages may require an extensive introduction to editing.

The section that follows describes the problem in more detail and outlines our solution. Section 3 summarizes how we improve on previous work. Section 4 describes GECIE, the initial implementation of our technology as an extension of the Berkeley Unix Mail. We conclude by summarizing our contributions.

2. Consulting in Interactive Environments

In order to use an environment effectively, a user must know the system's capabilities and how to make best use of them. This requires access to information that describes the specific features of the system — the functions, commands or constructs (henceforth functions) available. It also requires access to methods or plans for best accomplishing goals.

We claim there is a large middle ground between a novice who knows only the rudiments of a system and an expert who has gained complete mastery over it. The continuum in between is one in which user expertise may not be optimal for a given task. When the user must take time to find the appropriate function, or develop an efficient technique, productivity decreases. Furthermore, in some environments in which the tasks are primarily 'throw-away', users may rely on inefficient methods that are well-known rather than taking time to develop more sophisticated expertise. A primary reason for the inefficiency of learning is that users bear the burden of deciding what must be learned and how to locate the appropriate information. This is typically done by searching through manuals, asking help of others or simply experimenting with the system. Expert system techniques should be able to provide mechanisms that can relieve

some of this burden.

The objective of our research is to address these issues and offer a theory of how to build an automated consultant that can assist users in extending their expertise. The following insights are the result of informal observations of human consultants giving help in environments that support EMACS, Lisp, Unix, Pascal and Logo, and on an examination of manuals, tutorials and texts for these environments.

---

1. Function specification: What does function F do?
2. Goal satisfaction:
   a. How can goal G be accomplished?
   b. Plan P accomplishes goal G in the context of situation S, but there must be a 'better' way, what is it, and why is it better?
3. Analyze or debug a plan:
   a. What does plan P (learned by rote, for example) do?
   b. Plan P ought to accomplish goal G in the context of situation S, but doesn't, why not?

Figure 2-1: Typical Types of Questions Users Ask

There is rarely a direct correspondence between a precise statement of a user's goal and a plan to satisfy it. It is more often the case that the user's goal is poorly defined. Furthermore, a goal may be satisfied by more than one plan. The problem presents itself as requiring a mapping of many user queries to many possible answers. In order to constrain the potential mappings, user queries can be categorized at least partially as relating goals to plans as summarized in figure 2-1. Although the question itself may not be stated clearly in one of these forms, informal observations indicate that the intention of most utterances falls within one of these question types.

Magers [Magers 83] and Borenstein [Borenstein 85] have drawn a distinction between information that is *definitional* and *instructional*. Figure 2-2 further refines this distinction. Definitional information is more appropriate for reminding someone about something they have previously learned, while instructional information is more appropriate for introducing new information. These types differ not only in their format and level of detail, but also in their

Introduce:	Present functions and plans that the user has not encountered before.
Remind:	Briefly describe functions and plans that the user has been exposed to but may have forgotten.
Clarify:	Explain details and options about functions and plans to which the user has been exposed.
Elucidate:	Clear up misunderstandings that have developed about functions and plans to which the user has been exposed.
Execute:	Perform functions and plans directly for the user.

Figure 2-2: Types of Responses a Consultant Might Provide

emphasis and the degree to which related information is included. Clarifying and elucidating require a careful mixture of reminding and introducing. In this article, we address only the first four types of answers. We have developed a separate system, Marvel [Kaiser & Feiler 87; Kaiser *et al.* 88], that automatically generates and executes plans for the user in the context of software development and maintenance; our next application of GECIE, after mail systems, will be to scale up to Marvel.

Although the categorization in figure 2-1 constrains the question, while the taxonomy in figure 2-2 constrains the answer provided, the requisite knowledge and the processes needed to search that knowledge are still complex. The processes include the abilities to estimate the user's goal, to understand the user's plan, to evaluate the current situation in order to formulate an answer that does not digress from the current task, to analyze the user's plan in terms of the estimate of the goal and within the current situation, and to choose an appropriate answer and explanation depending on the user's current knowledge of the system. This requires knowledge of the functions provided by the system, the possible goals that can be accomplished with the system, the plans that may accomplish those goals, the things that typically go wrong (bugs), and what the user currently does and does not know about the functions, goals, plans and bugs.

Much of this cannot be completely known. For example, it seems unlikely that all possible goals achievable within a given interactive computing environment will be known before the environment is used extensively. It also does not seem possible to predict with certainty what the user's goal is and what the user knows. Thus the processes described above not only must

operate with incomplete information, but ought to be able to do so effectively. Innovative techniques or novel applications ought to be easily and reliably incorporated into the knowledge base.

From an AI Expert Systems perspective, these issues can be encapsulated in two fundamental problems: (1) How can the search through a vast and complex knowledge base be restricted in order to glean the appropriate information for the immediate needs of the user? and (2) What decisions must be made in order to choose the appropriate form in which to present that information? We provide solutions to both of these problems.

2.1. A Goal-Centered Approach

We propose a goal-centered approach in which the help given is a direct function of user's needs within the current context, treated as a *discourse* between the user and the consultant. In particular, we are interested in the content of the answer provided to users. The primary contributions of this research are as follows.

A good consultant has both extensive domain knowledge and expertise in how to explain something. A good consultant does not simply know how to use an environment effectively, but knows what to say, how much to say, and what approach to take depending on what she thinks the user knows. Unlike traditional expert systems, we do not encode both types of knowledge in one large rule base. Instead, we separate the *procedural domain knowledge* about how to do something from the *explanatory knowledge* of how to talk about how to do something. Modularizing consulting knowledge in this manner provides advantages already proposed by Clancey [Clancey 83]. Figure 2-3 illustrates this distinction. Two rule bases capture the explanatory knowledge. The Plan Analyst determines what information about the domain is relevant, while the Explainer determines in what form to phrase the answer.

Procedural knowledge is captured in a frame-based knowledge representation in a hierarchical organization of computational goals. The emphasis on goals is important because it is not enough for a consultant to know about the functions of an environment. Explicit knowledge of how to combine those functions into plans that accomplish computational goals is

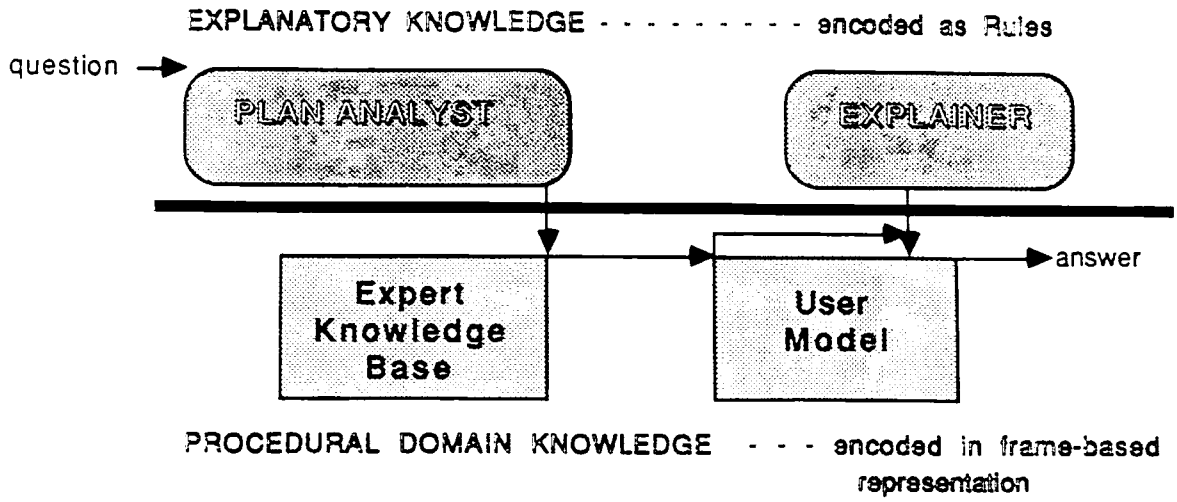

Figure 2-3: GECIE's Division of Consulting Knowledge

equally important. The representation allows the Plan Analyst to reason about the actions associated with functions, but also allows it to analyze whether plans can satisfy goals and which of many equally good plans is most appropriate in a given context. A good consultant does not simply perform a 'core dump' of relevant information, but filters that information to satisfy pedagogical goals. Our knowledge representation also contains explicit discourse information that the Explainer uses to satisfy pedagogical goals.

We exploit the structure of our knowledge base and use a goal-centered representation as a user model. This allows us to abandon a simple categorization of users as novice, intermediate and expert. Similarly we found it insufficient to cluster goals, plans and functions into groups such as simple or hard. Since the structure of the user model and expert knowledge base are the same, decisions about how to answer a user's question are based on an analysis of the match between these representations.

The feasibility of our approach is explored through an automated consulting system called GECIE. We have applied GECIE to the real world problem of the Berkeley Unix mail system [Shoens 86], notorious for the great power it provides experts and the great confusion it creates

for novices and even long-term non-expert users. Our goal is not to replace this mail system, as has been done by others [Stallman 85; Jackson & Barel 86], but instead to augment it with consulting behavior that makes its capabilities accessible to casual users.

For example, a user might ask "what does `type` do?" If the user knows nothing about either the `type` or `print` functions, GECIE provides the standard introduction to the `type` function. But if the user already knows about `print`, then GECIE explains that `type` is a synonym for `print`. If the user has previously used `type`, but has apparently forgotten what it does, GECIE simply remind the user of the `type` function as briefly as possible. This is an example of a "What does function F do?" question. Examples of the other question types from figure 2-1 in the context of Berkeley Unix Mail are presented in section 4.

Since other discourse-based systems focus on determining a user's goals (e.g., Wilensky [Wilensky *et al.* 84], Pollack [Pollack 86]), we assume the output of such an understanding mechanism as input to GECIE, and instead are concerned with how to take advantage of this understanding to generate useful responses to user queries. In order to test GECIE's capabilities with real users, we are building a menu-based front end. This narrows the range of questions that can be asked and requires a more careful articulation of the question by the user. For testing purposes we view this positively, since both the advantages and disadvantages of more sophisticated understanding mechanisms do not obscure our evaluation of the merits of generating an appropriate answer. The front-end is described in more detail in section 4.

Research in user modelling [Carberry 83; Grosz 81; Selker 88] present theories for how to automatically develop and update models of individual users. Again, we assume that appropriate mechanisms can be constructed for maintaining a user model and that output from such a system in the form of a goal-centered knowledge base can be passed to GECIE. For testing purposes, user models will be built by hand based on systematic human evaluations of individual user's knowledge.

One can therefore view the user's question and goal, and a representation of the user's knowledge as hand-coded input to GECIE. In the above example, we assume that GECIE is given the question in symbolic form and that the user model accurately reflects whether the user

already knows about `type`, `print`, or both, and concentrate our efforts on generating the best answer for the context.

3. Related Work

The development of programming environments [Goldberg 87; Habermann & Notkin 86; Stallman 81; Kaiser *et al.* 87; Walker *et al.* 87; Reiss 87]has focused on what the user can do rather than on how the user learns to do it. UC [Wilensky *et al.* 84; Chin 86], WIZARD [Finin 83] and ACRONYM [Borenstein 85] have articulated the need for comprehensive information accessing mechanisms. Evaluations of on-line help using ACRONYM indicated that the information itself is more important than the means for accessing it. UC and WIZARD both assume this, and provide information in the context of the user's goal. Both research groups acknowledge the need for pedagogical goals or 'tutoring strategies', but have not studied them beyond stereotyping functions along a novice/expert spectrum.

Quilici *et al.* [Quilici *et al.* 85] have demonstrated how goal/plan knowledge can be used to answer questions, but they do not describe how the form and content of a response is affected by what the user already knows. Others [Wilensky *et al.* 84; Johnson 86; Waters 86; Finin 83] identify the importance of plans, but they do not include in their knowledge bases the explicit discourse information needed to satisfy pedagogical goals. Much of the recent work on explanation [Kukich 85; Swartout 83] involves determining an appropriate level of detail or developing techniques for making inference chains coherent. McKeown [McKeown *et al.* 85] and Paris [Paris 85; Paris 87] go further to show how the decision of what to present from the knowledge base is dependent on the user's focus of attention and level of expertise.

Our work should be viewed as an extension of McKeown and Paris, but in an environment that is highly procedural. Our emphasis is on *how to do something*, rather than on *what something is*. A second distinction is that in an interactive computing environment there is often not only more than one way to explain something, but more than one way to do something. Therefore the analysis process that determines the most appropriate procedure affects and is affected by the generation process that produces the form and content of the answer.

4. GECIE: A Consultant for Interactive Environments

Consulting can be characterized as a three stage process of question understanding, problem analysis and answer generation. Our understanding component is currently a simple menu-based front end. We concentrate on the latter two stages: analysis, through a rule base called the Plan Analyst, and generation, through a rule base called the Explainer. GECIE attempts to answer a question by doing a two phase search of the knowledge bases. In the first, the Plan Analyst tries to construct a coherent relationship between the user's question, his user model and the capabilities of the system in an attempt to find the most appropriate information. Based on the Plan Analyst's output, the Explainer tries to construct a coherent textual explanation that takes into account what the user already knows. Both rule bases will be discussed extensively in the examples later in this section. In what follows below we describe the structure of the knowledge representation and present details of the understanding and generation components that are not obvious from the examples.

GECIE's 'understanding' component is a simple menu-based interface. Our goal was to develop an interface that would be both easy to implement and rapidly learned by users. Although such an interface does not understand in the Natural Language Processing sense, it does have some intelligence in the way menus are presented. Figure 4-1 shows the top level menu, which is a reformulation of the questions of figure 2-1. The user can select a goal or function by typing the proper word or phase at a command prompt or by browsing a menu of goals or functions. The menus can be arranged alphabetically, or the order of presentation can be based on the goal links of the expert knowledge base. Plans that can be identified by name from the knowledge base can be entered from the command prompt. Otherwise, the user must construct a plan by selecting an ordered list of functions and goals.

When GECIE is invoked within mail, both the expert knowledge base (EKB) and user model (UM) are loaded. The world model (WM) is constructed based on the user's current context in mail. Depending on the question type selected, the user is prompted to provide a function (F), a goal (G), or to construct a plan (P).

EKB is a hierarchy of the computational goals that can be satisfied in the target

Please select a question:

1. What does the function (select a function) do?
2. How can I (select a goal) ?
3. I use this plan (construct a plan) to (select a goal), is there a better way?
4. What does this plan do: (construct a plan)?
5. This plan (construct a plan) ought to accomplish (select a goal), but doesn't, why not?

Figure 4-1: Top Level Menu for Question Selection

environment. Figure 4-2 shows the structure of this frame-based knowledge representation. Computational goals contain links to alternative plans for satisfying the goal. A plan can be linked to a subgoal or an ordered sequence of subgoals that describe how it can be executed, or to a function that executes it directly. Encoded within a computational goal are links that describe the relationship between plans.

Functions describe the operators of the environment. Their representation includes information about the correct syntax of the function, any preconditions and effects, and the actions associated with parameters. Preconditions define a state that must be true before a fucntion can be correctly executed. They may also contain a link to a goal that could satisfy it. Effects encode the actions of functions when applied to the world model. Currently the world model is represented as a simple add/delete list that describes possible states in the mail environment. Therefore effects are encoded as directives to add or delete a state from the world model.

UM has the same representation as EKB. It contains a history of what the user has done in past sessions in terms of what goals have been accomplished and what plans and functions were used to accomplish them. It is currently coded and updated by hand. Problems associated with updating it automatically are discussed in section 5.

Most of GECIE's responses are stereotypical. At the same time, the content of a response must be customized to the user's needs and expertise. Therefore a rule based system that

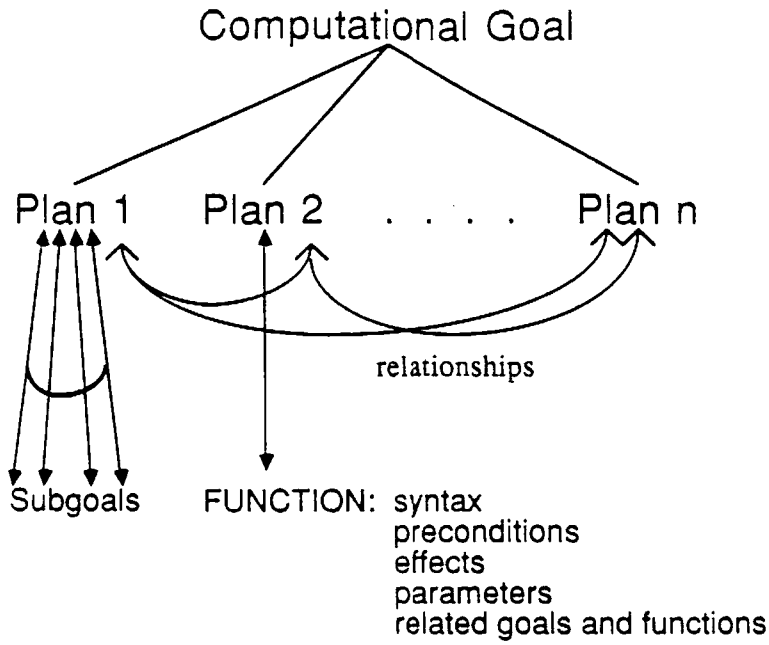

Figure 4-2: GECIE's Frames for Knowledge Representation

ultimately leads to canned text is inappropriate since the canned text is fixed. Similarly, since GECIE's range of discourse is limited, a completely open-ended generation system seems equally inappropriate. We therefore chose template filling as a technique that allows both customization and stereotyped responses. To generate an answer, the Explainer selects an appropriate set of *response agenda* based on the output of the Plan Analyst. The response agenda are directives for filling textual templates. Representative templates are presented in figure 4-3. Operations appear in capital letters; variables are surrounded by braces ("{ }").

We now present five example queries based on the question types in figure 2-1 to demonstrate GECIE's capabilities. The first two include scripts of the entire interaction between the user and GECIE. All five examples describe the rules used by the Plan Analyst to select the appropriate information. They also show typical scenarios of how the content of the user model and the user's question affect the output of both the Plan Analyst and the Explainer.

FUNCTION_INTRODUCE(f)
{f->name} is used to {f->satisfies->description}. It has the form {f->form}, where FOR_EACH (x,f->parameters, "{x} refers to {px->description}"). {f->name} requires that EXPAND_PRECONDS(f->preconds). It causes EXPAND_EFFECTS(f->effects). For example, EXAMPLE(f->form,WM).

FUNCTION_REMIND(f)
{f->name}: {f->form}. It is used to {f->satisfies->description}. For example, EXAMPLE {f->form,WM}.

GOAL_REMIND_SIMPLE(g)
You can {g->description} by using the command {g->function}. For example, EXAMPLE(f->form,WM) would EXPAND_EFFECTS(f->effects).

GOAL_INTRODUCE_SIMPLE(g)
GOAL_REMIND_SIMPLE(g). You must make sure EXPAND_PRECONDS(g->function->preconds).

GOAL_ INTRODUCE_COMPLEX(g,fault)
In order to {g->description}, you must FOR_EACH(gx,g->subgoals,"GOAL_INTRODUCE_COMPLEX(gx)"). IF fault DESCRIBE_FAULT(fault->plan). The commands to {g->description} are FORMAT_PLAN_INSTANTIATION(gx,g->subgoals,gx->function,WM). SHOW_MAPPING(gx,g->subgoals,gx->description,gx->function).

GOAL_REMIND_COMPLEX(g)
In order to {g->description}, use FORMAT_PLAN_INSTANTIATION(gx,g->subgoals,gx->function,WM). SHOW_MAPPING(gx,g->subgoals,gx->description,gx->function).

WM = World Model
Simple goals are satisfied directly by functions.
Complex goals are satisfied by a plan that maps to subgoals.

Figure 4-3: Representative Response Agenda

4.1. Example 1

The first question is: What does `type` do? This is an instantiation of the "What does F do?" category of figure 2-1. In order to ask this question, the user selects question 1 in the menu of figure 4-1. A second menu allows the user to enter a function name, or search functions alphabetically or by goal links. Using one of these methods the user indicates that the desired function is `type`.

Figure 4-4 shows the portion of EKB required to answer this question. The Plan Analyst

uses the following rules to determine what information is relevant to the Explainer:

1. If UM contains F, then report knowledge of F, else report no_knowledge of F.
2. If there exists a function that is directly satisfied by some goal G', which has the least complex relational link to the goal G that satisfies our function F, then F' = that function.
3. If F' exists in EKB, and UM contains F, then report knowledge of F'.

In our example, the Plan Analyst would determine whether the user already knows about `type`, and in this case, since there is a relational link to `print`, whether the user knows about `print`. The outcome of this analysis is passed to the Explainer.

```
G, type.goal,
        G_type: D       /* Satisfied directly by function */
        Satisfied by: F, type;
        Related goals:  RL4
G, print.goal,
        G_type: D
        Satisfied by: F, print;
        Related goals:  RL4
G, display.list.of.messages,
        Description:  display each message in the sequence specified
        G_type: S       /* Satisfied through subgoals */
        Satisfied by: G, type.goal; G, print.goal;

F, print,       Form: print (message_list)
        Preconditions:  P1, P2, P3,
        Effects: E1,
        Satisfies:  print.goal
        Parameters: message-list
F, type,        Form: type (message_list)
        Preconditions:  P1, P2, P3,
        Effects: E1,
        Satisfies:  type.goal
        Parameters: message-list

P, P1,  state: (exists contents_of (*p message_list))
        use: list.message
P, P2,  state: (at read-level)
        use: get.to.read.level
P, P3,  state: (size (*p message-list) > screen-size)
        use: set.window.scroll

E, E1,  Token: A state: (display-contains text-of each (*p message-list))

R, RL1, type.goal, print.goal
        Relation: synonyms
```

Figure 4-4: GECIE's Expert Knowledge for Question 1

Four analyses are possible based on the existence of F and F' in UM. These are illustrated in figure 4-5 along with the corresponding Explainer output. If the user knows nothing about either `type` or `print`, GECIE generates the standard introductory template for `type`, and does not overwhelm the user with the fact that `print` is a synonym. Figure 4-6 shows how the response agenda for FUNCTION_INTRODUCE(type) is filled from the EKB. If the user knows about `print`, GECIE states the fact that `type` is a synonym, reminds the user about `print`, then introduces `type`. If the user knows about `type` but not `print`, GECIE reminds the user about `type` and makes an aside that there is a synonym for `type` called `print`. Finally, in the last case, if the user knows about both, GECIE just reminds him about `type`.

```
/* UM: does not contain either 'type' or 'print' */
Plan Analyst output:  function: type    no_knowledge
Explainer output:     FUNCTION_INTRODUCE(type)

/* UM: contains 'print', but not 'type' */
Plan Analyst output:  function: type no_knowledge
                      function: print knowledge
Explainer output:     DESCRIBE_LINK(type,print)
                      FUNCTION_REMIND(print)
                      FUNCTION_INTRODUCE(type)

/* UM: contains 'type',  but not 'print' */
Plan Analyst output:  function: type knowledge
                      function: print no_knowledge
Explainer output:     FUNCTION_REMIND(type)
                      MAKE_SIDE_COMMENT(DESCRIBE_LINK(type,print))

/* UM: contains both 'type' and 'print' */
Plan Analyst output:  function: type knowledge
Explainer output:     FUNCTION_REMIND(type)
```

Figure 4-5: GECIE's Responses to Question 1

4.2. Example 2a

The second question is: How can I reply to a message? This is an instantiation of the "How can I satisfy G?" category of figure 2-1. To ask this question, the user selects question 2 in the menu of figure 4-1. In a second menu, the user selects the desired goal. Let us assume the user chose "reply.to.message". In this case it might be easier to locate the goal by searching a goal based menu rather than an alphabetized one. Let us further assume that WM contains a message which was sent only to him, not to other group members.

```
type is used to type a sequence of messages on the terminal.  It has
the form:

        type {message_list}

where {message_list} refers to a sequence of messages. type requires
that the contents of the message_list exist, that the user is at read
level and that the messages fit on the screen.  It causes the text of
each message in the message list to be displayed on the screen.  For
example:

        type 1:3

        displays messages 1 through 3.
```

Figure 4-6: Text Generated to Introduce the Function `type`

Figure 4-7 is a graphic representation of the portion of EKB required to answer this question. In this case the Plan Analyst constructs a trace through the goal hierarchy and passes it to the Explainer. The Plan Analyst uses the following rules:

1. If UM contains a plan P for G, then report user_plan = P and user's knowledge of relevant functions.
2. If EKB contains a most efficient plan P' for G, then report best_plan = P' and user's knowledge of any relevant functions.
3. If EKB does not contain P (the user's plan), then report plan_not_known = P.
4. If P = P', then report best_plan = user_plan.
5. If plan_not_known is a valid plan[1] report plan_not_known, else report fault = plan_not_known.

Three possible responses are illustrated in figure 4-8. If the user does not know anything about how to reply to a message, GECIE selects a 'best' plan based on the context and meta-knowledge of relational links. In this case, the context indicates that the response should be to reply only to the sender, and the meta-knowledge indicates that a task should be done now rather than later. Since the user knows about "compose.message", the only relevant function is `Reply`.

Figure 4-9 shows how the response agenda for this case is expanded to produce text. If the user has replied to messages in the past and does it efficiently then GECIE simply reminds the

[1] A discussion of how to determine the validity of a plan can be found in [Wolz 85].

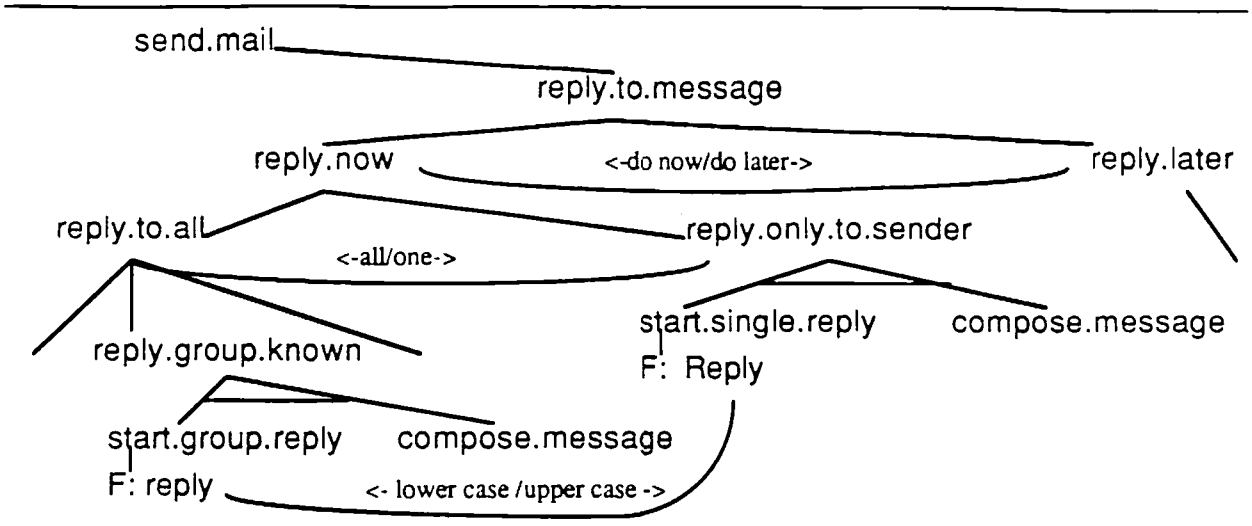

Figure 4-7: GECIE's Expert Knowledge for Question 2a and 3b

user about the command `Reply`. But, if the user seems to know how to reply to messages, but does it awkwardly, then GECIE introduces a better way. GECIE explains why it is better by providing the relational links between goals of the user's plan and the better plan. GECIE considers a plan to be awkward when the user's plan does not match GECIE's plan or when the user's plan is not even in EKB. The latter case is the last case shown in figure 4-8. Here the plan works, and is classified as not known, rather than faulty. Ideally plans that work but aren't known should be incorporated into EKB. This is discussed in section 5. Faulty plans are presented in Example 3b.

4.3. Example 2b

A refinement of the second question is: To reply to a group of users I reply to each individually — is there a better way? This is an instance of the "Given P is there a better P for G?" category of figure 2-1. In this case the user must identify the question type and select a goal and plan. Let us assume the user selected the goal "reply.to.all" and the plan:

```
FOR_EACH (x in group)
send.mail.to.individual
```

In the first case below we will assume that WM contains a message that was sent to the user and others. In the second case, WM contains a message that was sent only to the user. In the third

```
/* UM: contains send.mail compose.message */
Plan Analyst output:  user_plan: nil
                      best_plan :reply.to.message -> reply.now ->
                        reply.only.to.sender
                      function: Reply no_knowledge
Explainer output:     SUMMARIZE.PLAN(best_plan)
                      GOAL_INTRODUCE_SIMPLE(reply.to.message)

/* UM: contains reply.to.message -> reply.now ->reply.only.to.sender */
Plan Analyst output:  best_plan = user_plan
                      best_plan: reply.only.to.sender
                      function: Reply knowledge
Explainer output:     GOAL_REMIND_SIMPLE(reply.only.to.sender)

/* UM: contains reply.to.message -> save.message -> leave.read.level
            -> send.message */
Plan Analyst output:  best_plan: reply.now -> reply.only.to.sender
                      plan_not_known: plan -> reply.now ->save.message ->
                              leave.read.level -> send.message
                      function: Reply no_knowledge

Explainer output:   GOAL_INTRODUCE_COMPLEX(reply.only.to.sender,fault->plan)
```

Figure 4-8: GECIE's Responses to Question 2a

```
In order to reply to a message it is assumed you want to reply right
away and reply only to the sender.  To do this, you must indicate you
wish to reply and compose a message.  You can indicate you wish to
reply by using the command 'Reply'.  For example,

                Reply

would put you in write mode, the receiver of your message would be
identical to the writer of the message you just received.
```

Figure 4-9: Text Generated to Introduce the Goal "reply to a message"

case, WM does not contain any message.

Figure 4-10 is a portion of EKB required to answer this question. This question is analyzed using rules 2 - 5 of the last example. Rule 1 is unnecessary since we assume plan P chosen by the user should be in UM[2].

Three possible responses are illustrated in figure 4-11. In the first, the message to which the

[2]If the plan does not actually exist in UM, one should assume that after the interaction it is indeed inserted by a programmer.

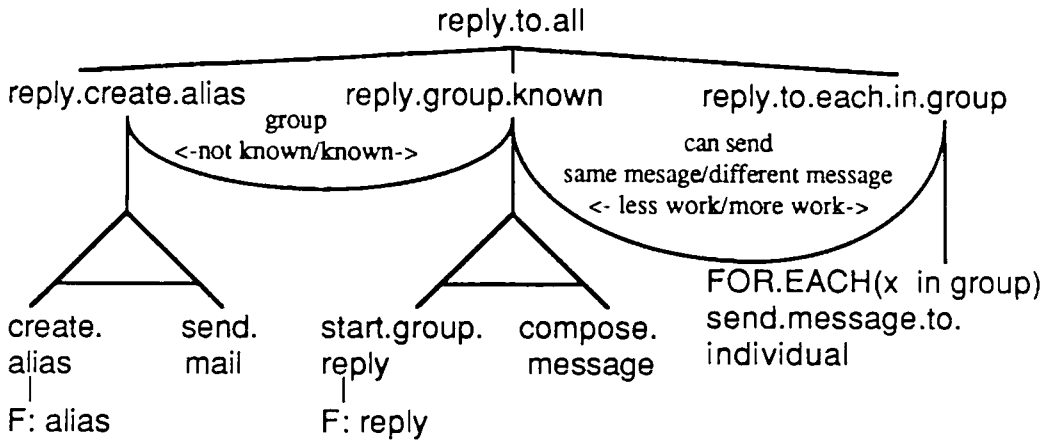

Figure 4-10: GECIE's Expert Knowledge for Question 2b and 3b

user wishes to reply was addressed to a group of users. GECIE chooses to tell the user about the `reply` command since a group exists in WM. In the second case, the message was only addressed to the user. GECIE chooses a plan that requires the user to identify a group of users. In both cases, since the user knows how to send mail, GECIE simply reminds the user about how to send mail and describes the relational links between the suggested plan and the user's. In the third case, the context does not allow a choice between these plans. GECIE presents both options. Both plans are preferred to the user's plan because they require less work on the user's part. In the event that the user's plan is equivalent to the suggested solution, GECIE would inform the user of this and use relational links to justify why the user's plan is best.

4.4. Example 3a

A third question is: My advisor told me to read her mail and look for messages pertaining to her course. These were the instructions:

```
enter MAIL
type h +
{note message numbers of relevant messages}
type save {message numbers} homeworks.txt
```

What's actually going on?

This question is an instance of "What does plan P do?" The user must supply a plan, which

```
/* WM contains message that was sent to user and others */
Plan Analyst output:  user_plan: reply.to.each.in.group
                      best_plan: reply.to.all -> reply.group.known
                      function: reply no_knowledge

Explainer output:       GOAL_REMIND_SIMPLE(reply.to.each.in.group)
                        GOAL_INTRODUCE_COMPLEX(reply.group.known)
                        DESCRIBE_LINK(reply.to.each.in.group,
                                      reply.group.known)

/* WM contains message that was just sent to user */
Plan Analyst output:  user_plan: reply.to.each.in.group
                      best_plan: reply.to.all -> reply.group.create.alias
                      function: alias no_knowledge

Explainer output:       GOAL_REMIND_SIMPLE(reply.to.each.in.group)
                        GOAL_INTRODUCE_COMPLEX(reply.group.create.alias)
                        DESCRIBE_LINK(reply.to.each.in.group,
                                      reply.group.create.alias)

/* WM does not contain explicit reference to a message */
Plan Analyst output:  user_plan: reply.to.each.in.group
                      best_plans:reply.to.all -> reply.group.create.alias
                                :reply.to.all -> reply.group.known
                      function: reply no_knowledge
                      function: alias no_knowledge

Explainer output:       GOAL_REMIND_SIMPLE(reply.to.each.in.group)
                        GOAL_INTRODUCE_COMPLEX(reply.group.create.alias)
                        DESCRIBE_LINK(reply.to.each.in.group,
                                      reply.group.create.alias)
                        GOAL_INTRODUCE_COMPLEX(reply.group.known)
                        DESCRIBE_LINK(reply.to.each.in.group,
                                      reply.group.known)
```

Figure 4-11: GECIE's Response to Question 2b

might be constructed as follows:

```
start.mail
h +
save message_numbers homeworks.txt
```

Figure 4-12 is a portion of EKB required to answer this question. The Plan Analyst searches for the most likely goal in EKB that is satisfied by P. If a goal cannot be found, that is, if there is some problem with the plan, GECIE re-evaluates the plan using the rules presented in the next example for faulty plans. When a goal is found, the Plan Analyst searches UM to see if the user knows the goal or any of its subgoals. It uses the following rules:

1. If EKB contains a goal G that is satisfied by P then best_goal = G, target_plan = P, else report bad_plan = P.

2. If best_goal exists and UM contains best_goal, then report that user knows best_goal.
3. If best_goal exists and for all parts P_i of target_plan, UM contains P_i then report set_of_P_i = all P_i that user knows.

Four possible responses are illustrated in figure 4-13. In the first case the user does not know any steps in the plan. GECIE starts at the highest level goal found, which is to collect a subset of messages, and introduces all of the steps. In the second case the user knows some of the steps. GECIE assumes that the plan is used to collect a subset of messages, introduces those steps that the user does not know and then merely reminds the user of those steps the user already knows. In the third case the user knows all of the steps in the plan and GECIE simply reminds him about how to collect a subset of messages. Finally in the last case the user knows how to collect a subset of messages, by reading the messages themselves rather than the headers. GECIE describes the links between the plan given in the question and the user's plan.

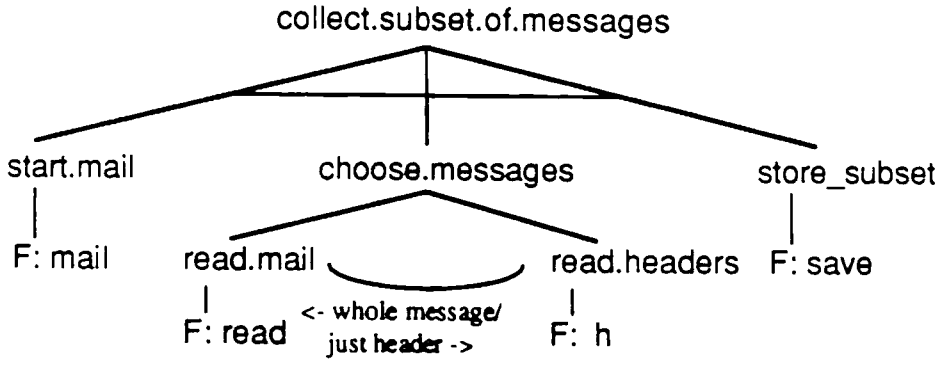

Figure 4-12: GECIE's Expert Knowledge for Question 3a

4.5. Example 3b

The last question we consider is actually two questions. These are modifications of question 3a, but the user actually identifies the plan as bad:

1. I'm trying to reply to a group of users but I only seem to be able to reply to the

```
/* UM: does not contain best goal on any of the steps. */
Plan Analyst output:  set_of_P_i: nil
                      target_plan: collect_subset_of_messages->start.mail->
                                   read_headers->store_subset
                      function: mail no_knowledge
                      function: h no_knowledge
                      function: save no_knowledge
Explainer output:     GOAL_INTRODUCE_COMPLEX(collect_subset_of_messages)

UM: contains start.mail store.subset
Plan Analyst output:  set_of_P-(i): (start.mail,store_subset)
                      target_plan: collect_subset_of_messages->....
                      function: h no_knowledge
Explainer output:     GOAL_SUMMARIZE(collect_subset_of_messages)
                      GOAL_REMIND(start.mail)
                      GOAL_INTRODUCE(read_headers)
                      GOAL_REMIND(store_subset)

/* UM: contains collect_subset_of_messages->start.mail->
read.headers->store.subset*/

Plan Analyst output:  set_of_P-(i): (start.mail,read.headers,store_subset)
                      target_plan: collect_subset_of_messages->....
Explainer output:   GOAL_REMIND_COMPLEX(collect_subset_of_messages)
/* UM: contains collect_subset_of_messages->start.mail->
read.messages->store.subset*/

Plan Analyst output:  set_of_P-(i): (start.mail,store_subset)
                      target_plan: collect_subset_of_messages->....
                      user_plan: collect_subset_of_messages->start.mail->
                        read.messages->store.subset
Explainer output:   GOAL_SUMMARIZE(collect_subset_of_messages)
                    GOAL_REMIND(start.mail)
                    GOAL_REMIND(read.messages)
                    GOAL_INTRODUCE(read.headers)
                    DESCRIBE_LINK(read.messages,read.headers)
                    GOAL_REMIND(store_subset)
```

Figure 4-13: GECIE's Responses to Question 3a

sender of the message. Why?

2. I'm using `reply` to reply to the sender of a message, but seem to send mail to everyone else to whom the message was addressed. Why?

Both are instances of "Plan P ought to accomplish goal G in the context of situation S, but doesn't, why not?" In this case the user must identify both the goal and a plan. The Plan Analyst uses the following rules:

1. If P is a valid plan for G in the EKB, then report valid_plan = P, else report bad_plan = P.
2. If valid_plan exists and EKB contains a most efficient plan P' for G, then report best_plan = P'.

3. If P = P' then report best_plan = valid_plan.
4. If bad_plan exists and the fault is missing preconditions, then report fault_type: missing precondition(s).
5. If bad_plan exists and the fault is missing steps, then report fault_type: missing plan_step(s).
6. If bad_plan exists and the fault is extraneous steps, then report fault_type: extra plan_step(s).
7. If bad_plan exists with missing preconditions and missing step, and missing step satisfies missing preconditions, then report missing preconditions satisfied by missing steps.
8. If bad_plan exists with missing plan_step and extra plan_step, and relationship exists between them, then report relationship.

If the question fires rules 1 - 3 then the scenarios is very similar to examples 3a. Figure 4-10 includes the necessary representation of the EKB to answer the first question above. The user identifies the goal as "reply.to.all", and the plan as:

```
reply
compose.message
```

Assume the cause of the problem is that WM contains a message that was only sent to the user. In this case rule 4 would be fired since a precondition of the reply command is missing, namely that the WM must contain a group to which to send the message and not just a single user. The output of the Plan Analyst and Explainer are shown in the first case of figure 4-14. The Explainer suggests creating an alias to refer to the group of users.

Figure 4-7 includes the necessary representation of the EKB to answer the second question above. The user identifies the goal as "reply.only.to.sender", and the plan as:

```
reply
compose.message
```

The problem here is that the wrong command is being used, namely `reply` rather than `Reply`[3].

The second example in figure 4-14 shows the output of the Plan Analyst and Explainer. The Plan Analyst notices that there is both a missing step — "start.single.reply" and an extra one — "start.group.reply." It checks to see whether there is a relationship between them and finds the

[3]For the Unix uninitiated, character case matters, so `Reply` and `reply` are indeed two different commands. We admit this is grossly user unfriendly and should not occur in the first place, but the example was too good to resist.

relationship one level higher between "reply.to.all" and reply.only.to.sender", and one level lower between `reply` and `Reply`. The Explainer uses this information to generate a response that tells the user that `reply` and `Reply` are two different commands and that the desired one is `Reply`.

```
/* UM: contains compose.message; WM contains message sent only to user */
Plan Analyst output:  bad_plan: user_plan
                      fault_type: missing precond: (user_group)

Explainer output:   GOAL_INTRODUCE_COMPLEX(create.alias)

/* UM: contains compose.message */
Plan Analyst output:  bad_plan: user_plan
                      fault_type: missing plan_step: start.single.reply
                      fault_type: extra plan_step: start.group.reply
                      relations: reply.to.all <-> reply.only.to.sender
                                   <- user_group/single_user ->
                                 F: reply <-> F: Reply
                                   <- small r/ capital R ->
Explainer output:   GOAL_SUMMARIZE(reply.only.to.sender)
                    GOAL_REMIND_SIMPLE(start.single.reply)
                    GOAL_REMIND_SIMPLE(start.group.reply)
                    DESCRIBE_LINK(start.single.reply,start.group.reply)
                    FUNCTION_REMIND(Reply)
                    FUNCTION_REMIND(reply)
                    DESCRIBE_LINK(Reply,reply)
```

Figure 4-14: GECIE's Responses to Question 3b

5. Implementation Status and Discussion

GECIE is implemented in C under Unix 4.3 BSD on a MicroVAX II. We currently have plans to test GECIE on real users and to further develop the system. Work remains to be done on the menu-based front end, on the rules for both the Plan Analyst and the Explainer, on the mechanics of generating text form templates, and on automatically updating both knowledge bases. We also have plans to test the domain independence of GECIE by applying it to two other domains, software development and VLSI design.

In order to evaluate GECIE's capabilities we plan to will conduct a small controlled study with obliging human guinea pigs. These users will be given a limited introduction to Berkeley Unix Mail and will be asked to complete a series of assigned tasks. We will compare GECIE with the standard Unix "man" facility, with a human consultant, and with a scaled down GECIE

that only reasons about goals from the expert knowledge base, but not from a user model.

The current version of the menu interface was developed for expediency rather than for theoretical insights into interfaces or natural language understanding methods. Clearly there are better ways to extract information from a user, but this is not the focus of our work. One enhancement we do intend to make is to give the user the option of constructing a hypothetical world model rather than using the current mail as context. This would allow users to ask "what if" questions rather than being bound by their current task.

The current rule bases in both the Plan Analyst and Explainer are sufficient for answering uncomplicated questions such as those presented in the examples. But often a user's initial question spawns other questions that may be combinations of question types and require a mixture of answering strategies. We plan to expand both rule bases to handle more complex questions.

A problem with the current version is the Explainer's use of templates. In many of the examples one can see that the final text generated by the response agenda is redundant. This demonstrates an inadequacy of template based methods and has led us to consider using a functional unification grammar [Kay 79] that can unify the content of the individual response agenda to produce more graceful text.

Finally, we would like to be able to update both knowledge representations automatically. At first glance this would seem to be fairly straightforward. If a user presents GECIE with a plan that works but isn't currently in EKB, simply insert the plan. The problem lies in determining the right place for the plan in EKB and automatically attaching the right relational links between the plan and those already in EKB. Similarly, when the user demonstrates knowledge of a goal, plan or function, one would assume the appropriate structure could be inserted in the right place in the user model. But here we are faced with developing criteria for evaluating knowledge acquisition. Both of these issues fall within the domain of research on machine learning. Consideration of them at the present time would take us too far afield.

6. Conclusions

This paper describes GECIE, an automated consultant for answering questions within interactive computing environments. We focus on answer generation in the context of extending user expertise in such environments. We separate GECIE's knowledge into two components, a rule base that captures knowledge of how to consult, and a frame-based hierarchical knowledge representation that encodes knowledge of the domain about which to consult. We do not categorize users along a spectrum of expertise, and functions along a spectrum of level of difficulty. Instead we present a goal-centered approach where an answer to a question about the environment is based on knowledge of what the user has done in the past and is doing now.

Acknowledgments

This article is an expansion of "A Discourse-Based Consultant for Interactive Environments", which appeared in the *Fourth IEEE Conference on Artificial Intelligence Applications*, San Diego CA, March 1988, pp. 28-33. We thank Kathy McKeown for her ongoing support of this project. Michael Lebowitz, Cecile Paris and Michael van Biema also deserve thanks for their helpful comments and insights.

References

[Borenstein 85] Borenstein, N.S.
The design and evaluation of on-line help systems.
PhD thesis, Carnegie Mellon University, April, 1985.

[Carberry 83] Carberry, S.
Tracking user goals in an information-seeking environment.
In *Proceedings of AAAI-83*. American Association of Artificial Intelligence, 1983.

[Chin 86] Chin, D.N.
User modeling in UC, the UNIX Consultant.
In *Proceedings of the CHI'86 Conference*, pages 13-17 . Boston, MA, April, 1986.

[Clancey 83] Clancey, W. J.
The epistemology of a rule-based expert system - a Framework for explanation.
Artifical Intelligence, 20:215-251, 1983.

[Finin 83] Finin, T.
Providing help and advice in task oriented system.
In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 176-178. Karlsruhe, West Germany, 1983.

[Goldberg 87] Goldberg, A.
Programmer as reader.
IEEE Software, 9:62-70, September, 1987.

[Grosz 81] Grosz, B.
Focusing and description in natural language dialogues.
In A. Joshi, B. Webber and I. Sag (editor), *Elements of Discourse Understanding*, pages 85-105. Cambridge University Press, Cambridge, England, 1981.

[Habermann & Notkin 86]
Habermann, A.N. and D. Notkin.
Gandalf: Software development environments.
IEEE Transactions on Software Engineering, SE-12(12):1117-1127, December, 1986.

[Jackson & Barel 86]
Jackson, P. and M. Barel.
Introduction to computing facilities training program volume 1 -- Using a MicroVax
Carnegie Mellon University Software Engineering Institute, Pittsburgh, PA, 1986.

[Johnson 86] Johnson, W.L.
Intention-based diagnosis of novice programming errors.
Morgan Kaufmann Inc., Los Altos, CA, 1986.

[Kaiser & Feiler 87]
Kaiser, G.E. and P.H. Feiler.
An architecture for intelligent assistance in software development.
In *9th International Conference on Software Engineering*, pages 180-188. Monterey, CA, March, 1987.

[Kaiser *et al.* 87] Kaiser G. E., S. M. Kaplan and J. Micallef.
Multiuser, istributed language-lased environments.
IEEE Software, 11:58-67, November, 1987.

[Kaiser *et al.* 88] G. E. Kaiser, N. S. Barghouti, P. H. Feiler and R. W. Schwanke.
Database support for knowledge-based engineering environments.
IEEE Expert, May, 1988.
To appear.

[Kay 79] Kay, Martin.
Functional Grammar.
In *Proceedings of the 5th meeting of the Berkeley Linguistics Society.* Berkeley Linguistics Society, 1979.

[Kukich 85] Kukich, K.
Explanation structures in XSEL.
In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*. Chicago, IL, 1985.

[Magers 83] Magers, C. S.
An experiemntal evaluation of On-line HELP for non-programmers.
In *CHI'83 Proceedings*, pages 277-281. 1983.

[McKeown *et al.* 85]
McKeown, K.R., Wish, M. and Matthews, K.
Tailoring explanations for the user.
In *Proceeding of the IJCAI*. 1985.

[Paris 85] Paris, C.
Description strategies for naive and expert users.
In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*. Chicago, IL, 1985.

[Paris 87] Paris, C. L.
The Use of Explicit User Models in Text Generation: Tailoring to a User's Level of Expertise.
PhD thesis, Columbia University, 1987.

[Pollack 86] Pollack, M.
Inferring domain plans in question-answering.
PhD thesis, Moore School, University of Pennsylvania, May, 1986.

[Quilici *et al.* 85] Quilici, A.E., G. Dyer and M. Flowers.
Understanding and advice giving in AQUA.
Technical Report, UCLA Artificial Intelligence Laboratory, Los Angeles, CA, 1985.

[Reiss 87] Reiss.
Working in the garden environment for conceptual programming.
IEEE Software, 11:16-26, November, 1987.

[Selker 88] Selker. T.
Cognitive Adaptive Computer Help - A Research Overview.
Technical Report, T.J. Watson Research Center, IBM, T.J. Watson Research Center, Yorktown Heights, N.Y., 1988.

[Shoens 86] Shoens K.
Mail Reference Manual
Version 5.2 edition, 1986 .
Revised by Craig Leres .

[Stallman 81] Stallman, R.M.
Emacs The extensible, customizable, self-documenting display editor.
In *SIGPLAN SIGOA Symposium on Text Manipulation*, pages 147-156. June, 1981.

[Stallman 85] Stallman, R.
GNU Emacs Manual
3rd edition, MIT Artificial Intelligence Lab, Cambridge, MA, 1985.

[Swartout 83] Swartout, W. R.
xplain: a system for creating and explaining expert consulting systems.
Artificial Intelligence, :285-325, September 1983, 1983.

[Walker *et al.* 87] Walker, J. H., D. A. Moon, D. L. Weinreb & M. McMahon.
The symbolics genera programming environment.
IEEE Software, 20:36-87, November, 1987.

[Waters 86] Waters, R.C.
KBEmacs: Where's the AI?
The AI Magazine, 7(1):47-56, Spring, 1986.

[Wilensky *et al.* 84]
Wilensky, R., Y. Arens, and D. Chin.
Talking to Unix in english:An overview of UC.
Communications of the ACM, 27(6):574-593, June, 1984.

[Wolz 85] Wolz, U.
Analyzing user plans to produce informative responses by a programmer's consultant.
Technical Report CUCS-218-85, Department of Computer Science, Columbia University, New York, NY, 1985.