Help generation for task based applications with HATS

García, F.[*], Contreras, J. [*], Rodríguez, P. [*] and Moriyón, R. [*,†]
[*]*Escuela Técnica Superior de Informática. Universidad Autónoma de Madrid*

[†]*Instituto de Ingeniería del Conocimiento. ADIC.*

Abstract: This paper describes HATS, a system for the generation of context sensitive help for interactive applications. HATS delivers help based on a hierarchy of tasks the user can perform. As the user interacts with the application and accomplishes the tasks, the help messages are updated automatically, and give him context sensitive information about the actions needed to accomplish the desired task, including graphical references to the places in the application window where the tasks can be performed. HATS includes a reduced environment that simplifies the design of the help system.

Keywords: Help generation, Task models, Formal notations, Intelligent interfaces

1. INTRODUCTION

During the last years, interactive applications have become increasingly harder to use. This process, that has taken place in despite of a great improvement in the design and evaluation methodologies in the life cycle of these systems, has brought special attention to the key role played by the help components, integrated in those applications.

However, these help components have not followed the rapid development pace of the applications they serve. The most extended paradigm employed at the present moment consists in using hypertext, or hypermedia, to inform the user about the steps that must be undertaken to accomplish the tasks furnished in the application. This could be seen as recipes, the hyperlinks being used to change from one recipe to another.

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35349-4_22
S. Chatty et al. (eds.), *Engineering for Human-Computer Interaction*

The main drawbacks of this approach have been analyzed in detail in (Contreras, 1998). The two most important ones are, in the first place, the fact that all the explanations are given to the user at a very low level, related uniquely with atomic interactions s/he must perform. Secondly, the absence of feedback and interaction between the application and the help system.

These deficiencies observed in current help systems may have important consequences. Bhavnani points the type of help provided to users of such complex systems as CAD applications, as one of the reasons that could explain the small productivity increases in firms using those systems (Bhavnani, 1996).

In the present work, we describe a system that generates high level help, with a minimum development cost for designers. This system, called HATS (Help for ATOMS Task System) is able to automatically generate help for those interactive applications for which the user tasks have been defined according to ATOMS.

ATOMS (Rodriguez, 1997), is a system that permits the specification of hierarchical user tasks for interactive applications. These tasks may include context information, so as to describe under which circumstances the tasks are carried out; they are defined by means of a declarative language. This system is also comprised of a tasks manager, in charge of following the user activity in the application with respect to the tasks defined. It is important to note that the task model is general enough so as to be used by other tools who can reason about, providing services such as help, tutoring, macro and undo facilities, etc. A detailed discussion of the advantages of hierarchical task models can be found in (Zeiliger, 1997).

Using the benefits provided by ATOMS, HATS gives help to the user of an application that is substantially different to the one offered by current help systems as described above, and is mainly characterized by:

- It is task oriented. Explanations are generated by analyzing the task model. Thus, they are more expressive and have greater semantic content.
- Help is provided *just in time*, i.e., at every moment only the pertinent explanations are given according to the state of execution of the task being performed. This is possible thanks to the capabilities found in ATOMS. Since the system is able to detect the state of the application, HATS uses this knowledge to provide the user with relevant information, such as feedback about finished tasks or what steps should be performed next.
- The help system makes direct references to the widgets of the application where the user is supposed to act next, in a graphical manner. By means of highlighting in the application window, HATS indicates in a very clear way where interactions described in the help window should take place.

Finally, the cost for the designer when using HATS to generate help for her/his application is reduced: s/he just has to specify the tasks for the application, along with the corresponding help information for those tasks.

This paper is organized as follows. We first describe previous related work, followed by an example originating from a CAD application, that we will be using to illustrate the main characteristics of our help system. We continue with a brief description of ATOMS, just to introduce HATS in detail. We finish discussing some conclusions and research lines for future work.

2. RELATED WORK

In the last years there has been an important amount of research work in the field of generating help for interactive applications, using different paradigms. In this section we describe previous work that overcomes some of the drawbacks of traditional help systems pointed out in the introduction.

Humanoid Hyper Help, H3 (Moriyón, 1994), is a system that generates hypertext based help about the data presented in application displays, the commands to manipulate the data, and the interaction techniques to invoke the commands, for applications that have been developed according to the Humanoid model based paradigm (Szekely, 1992). It can derive the help automatically from the model used to implement the interface. Although it simplifies in a great deal the work of the designer generating a first version of a help system that can be refined a posteriori, it is not able to produce task oriented help about high level questions like *How do I merge these two files?* or *How can I add an arrow head at the end of this line?*. Moreover, help design in systems like H3 is more complex than it is in HATS, since a set of production rules is needed. This is due to the fact that HATS is task oriented, while H3 is widget oriented.

Cartoonist (Sukaviriya, 1990) is another interesting system that has in common with H3 the fact of generating help automatically from the models used to construct the interface. The originality of this system, though, lies in that it uses animation to convey the information to the user. Cartoonist uses backchaining reasoning in order to show how to satisfy preconditions.

The work by Pangoli and Paternó (Pangoli, 1995) has in common with our system the fact of being specialized in giving task oriented help; hence, the help given is more expressive that the one found in traditional help systems. However, this system does not use the tasks descriptions to follow the user actions, and thus it cannot synchronize help messages and user actions, as it happens with HATS.

Finally, Teach me While I Work, TWIW (Contreras, 1998), is another system centered around a model that describes the tasks that can be performed by the user. Although it can be used as a help system, it goes one step further and it is aimed at being used mainly as a tutoring tool. In this sense, it can not only inform the user about the actions to be performed, but also filter these actions in case they are not the appropriate ones. The absence of parameters in the task model used by TWIW makes it less context sensitive than the system we propose in this paper; the task model is less rich than the one used in HATS, specially in sequencing issues.

3. AN EXAMPLE

A very graphical way of introducing the problem we are interested in, is by making use of a real world based example. In order to do that, we have chosen one of the excellent examples presented in (Bhavnani, 1996). That paper focuses on the unrealized potential of Computer-Aided Drafting, even in the case of experienced users, that still employ suboptimal strategies to perform complex CAD tasks. The examples presented in that paper are quite convenient for our own purposes; that is, to introduce HATS, our ATOMS based help system on user tasks. Concretely, the

example we will use is the one there referenced as Example 1, consisting of Adding Fire Protection to a Column:

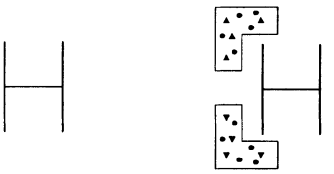

Figure 1. Drawing L-shaped fire protection enclosures for a column.

As described by the authors, the fire protection task consists of drawing 'fire protection enclosures around columns in a floor plan'. The fire protections are L-shaped polygons patterned with dots and triangles symbolizing concrete. So, the picture that represents a column must end up with two identical poched (patterned) L-shapes once the task is finished. This can be seen in Figure 1.

There are two main possibilities of drawing the protections that surround the column. The first one consists of drawing both L-shaped protections, patterning them afterwards. The second one consists of drawing one of the shapes, patterning it and then mirror copying the shape with respect the column axis. Of course, there are some other mixed possibilities. After completing the task in a CAD system, it becomes evident that the second possibility corresponds to the optimal task scheduling, while all the others can be considered as suboptimal.

One of the objectives of the help system we propose, is to provide the users with a global perspective of the task, guiding him/her through the appropriate sequencing of subtasks. This possibility is suggested in Bhavnani's paper as one way of providing users with strategic information that is not part of the application itself; that is, by including knowledge that is not in help manuals.

The inclusion of strategic knowledge in complex systems, secondary or non existent few years ago, is growing in interest within the CAD community, given that commercial CAD products are continuously resulting in more featured systems, often with more than a thousand commands. Of course, all these commands are detailed in manuals or consult books, where several pages are often devoted to the characteristics and parameters of most of them. However, higher level strategies are not usually included, at least as substantial chapters.

Next, we briefly describe ATOMS, the task oriented system HATS is based on.

4. ADVANCED TASK ORIENTED MANAGEMENT SYSTEM

ATOMS, *Advanced Task Oriented Management System*, is a system aimed to deal with complex user tasks. ATOMS tasks include context information that is updated dynamically as the user performs different activities. The task hierarchy is defined

by rules. The definition of tasks and rules is declarative, being similar to the definition of grammar rules in natural language processing systems. ATOMS uses a parser, able to identify the global tasks being accomplished, as well as the context determined from partial information taken from the user actions. More details about the system can be found in (Rodríguez, 1997).

ATOMS based applications incorporate a set of task patterns and rules that constitute its task model. Part of it is common for all the applications, and another part is application dependent. Some tasks are tagged as *application tasks*, in the sense that they are completely meaningful from the user point of view.

On the other hand, task patterns can be *atomic*, supporting the definition of direct user interactions, or *composed*, defining high level actions in terms of simpler ones. Both types of patterns admit the inclusion of parameters, that serve as context information for the tasks. Selecting a graphical object of a window could be an instance of an atomic task pattern, whose parameter is the selected object. Similarly, drawing a polygon could be an example of a composed application task pattern in which the polygon itself becomes the task parameter.

Apart from task patterns, the application task model includes rules. Those rules include information about (a) subtasks that constitute a composed task, (b) constraints over the values of parameters from related tasks, including not only immediate propagation of values, but also methods for assigning new values, (c) task preconditions that indicate the conditions under which the rule holds and (d) task sequencing, that can be sequential, parallel or alternative. Sequential order holds for the consecutive execution of the subtasks, while parallel assimilates to 'any order' and alternative to 'exclusive or'.

Tasks can also be qualified according to the number of times they can be executed. So, for instance, a task can be labeled as *optional*, meaning that its execution is not necessary for the global objectives, and also as *multiple*, meaning that it can be executed several times. In this last case, the number of repetitions can be specified.

Not all the information associated to tasks can be specified declaratively: task preconditions are predicates that must be satisfied whenever the task is accomplished. ATOMS allows the definition of preconditions by means of task patterns, that can be defined in a declarative way, but general preconditions are also possible, and they have to be programmed. The same happens with the methods used for parameter passing.

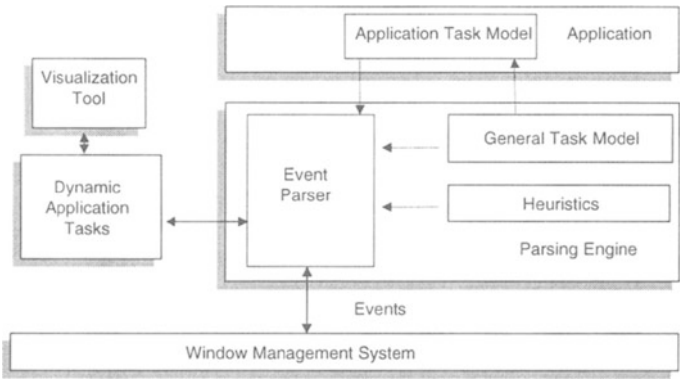

Figure 2. ATOMS architecture

The implementation of ATOMS is based on the AMULET framework (Myers, 1997). AMULET allows the designer to deal with high level events, based on interactors, which represent the basic actions a user performs. This feature, together with the fact that it supports a complete prototype/instance object system, is extensively used in ATOMS. The ATOMS architecture is shown in Figure 2.

At runtime, a set of *dynamic application tasks* reflects which actions the user is performing, including contextual information. The *event parser* analyzes the interactions of the user with the application, on the basis of user events, the task model, and the dynamic application task set. This parsing process allows the event parser to maintain the state of the application tasks in the dynamic application tasks area.

In next section, where we explain how HATS is used to provide help to the user on the task of adding fire protection to a column, most of the above mentioned ATOMS features will reveal.

5. HATS

HATS generates task based help for applications that incorporate an ATOMS task model. HATS provides two kinds of information: on one hand, like in the work of Pangoli and Paternó, (Pangoli, 1995), it explains complex tasks in terms of simpler ones, something that is much easier to comprehend than just a list of elementary interactions like mouse clicks, as it happens in most conventional help systems. On the other hand, somehow as in H3, (Moriyón, 1994), HATS gives context sensitive graphical information about the locations in the window where the user can interact in order to achieve the goals described in the different tasks, and how this interaction should take place. The context sensitivity of HATS is due to the fact that ATOMS is able to distinguish actions that simpler task management systems would consider as being similar.

The user can ask for help in two ways: either by selecting a task from a hierarchically organized list of application tasks, or by selecting a part of the window and asking for actions that can be performed there.

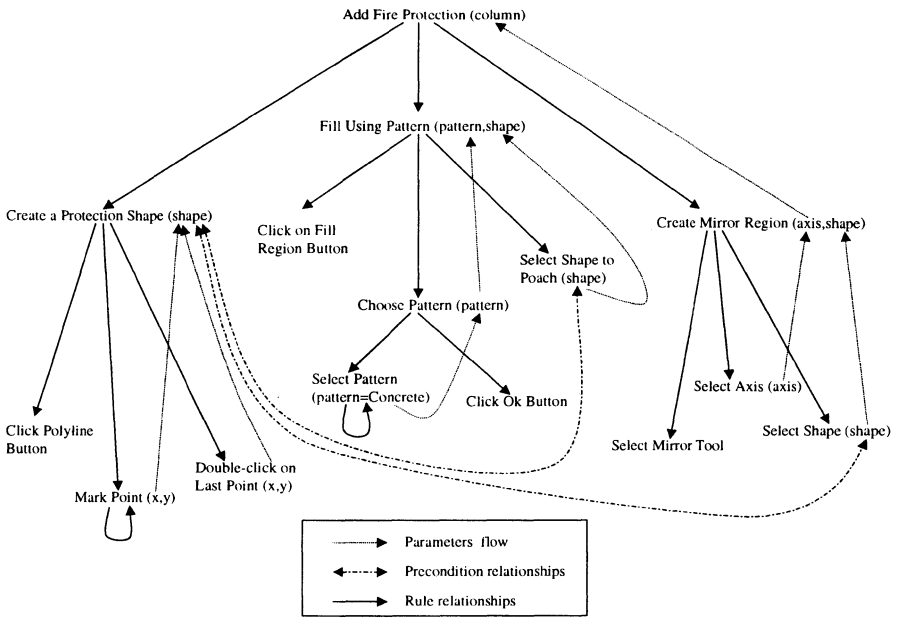

Figure 3. An optimal decomposition of the task: *Add Fire Protection to a Column.*

Once an application task is selected, there are two types of help the user can obtain: on one hand, s/he can navigate through the subtasks of the application task s/he has asked help for, getting descriptive information about them; hence, at each moment there is a *current help task.* On the other hand, the user can get help about the actions that have to be performed, and the relevant information is updated as the user accomplishes the task s/he is getting help for; hence, at each moment there is also a set of *next pending tasks.*

When the current help task is not one of the next pending tasks, HATS is in *descriptive mode.* Then, the messages it generates tell the user about how the current help task can be accomplished. Otherwise, the system is in *active mode,* and the corresponding help messages tell the user about the actions to be done.

By default, when giving help about a subtask, HATS does not give information about the higher level tasks in whose context it is accomplished. This policy is enforced in order to avoid the proliferation of long messages and large windows that make the use of HATS uncomfortable. The user can ask for additional task context information that shows him/her the super-tasks of the current help task.

In the rest of this section we will give first an overview of the kind of help provided by HATS, centered around the help the user can get for the adding fire protection task previously described. The task decomposition we will use is

presented in Figure 3. This example has been implemented on a prototype of a CAD system for which a task model has been specified. Afterwards, more general aspects, like the architecture of the HATS system, and issues about the design of help for applications are treated.

5.1 HATS message window

When the user requests help for a task like *Add Fire Protection to a Column,* the HATS message window appears, as can be seen in Figure 4, showing basic information for the task. In our example, the following *task description message* is shown (both underscored and bold text are emphasized only for explanation purposes later on in the paper):

You have requested help about how to ***Add Fire Protection to a Column.***
This task ***adds L-shape concrete blocks around the outer corners of the column to be protected.***
You can do this task on any of the highlighted objects.

Simultaneously, all the columns that appear in the drawing area of the application window will be highlighted. The HATS message window includes just simple messages, like the one above, and a button, *NextToDo,* that focuses HATS on one of the next pending tasks, hence switching the system to active mode.

The user can also enforce focus based modifications to the help message through the items in the *Focus* menu *(Go To Next Task, In Context,* and *DeContext),* and explore alternative ways to accomplish the current help task through the items in the *Alternatives* menu *(All Alternatives, Next Alternative,* and *Previous Alternative).* The *Show Task Hierarchy* item in the *View* menu opens a window that displays the state of the tree of the subtasks that are being explained. Besides the buttons, the user can also interact with the HATS message window by clicking on the lines of the message being shown, as we will explain later.

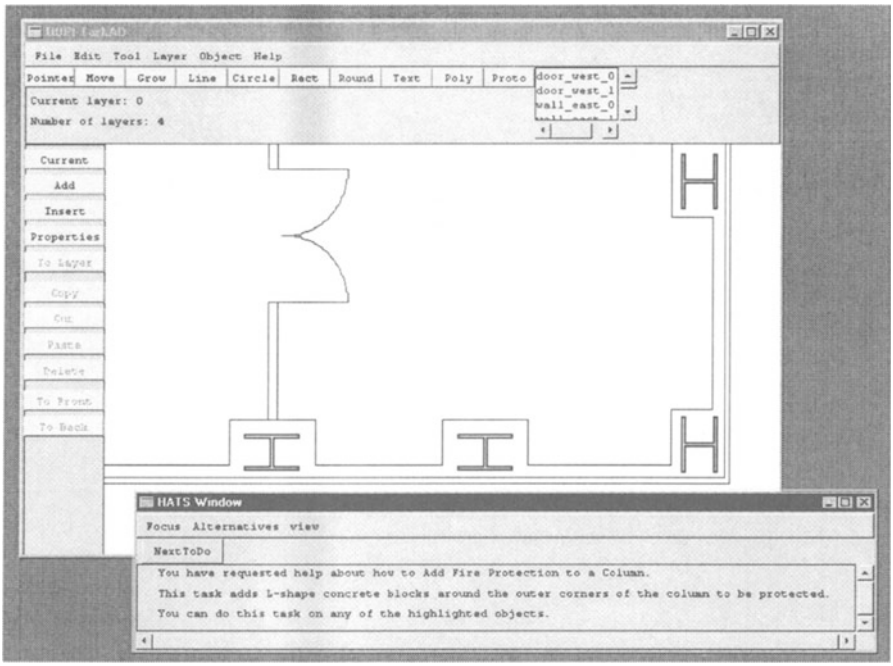

Figure 4. Requesting help for the adding fire protection task.

All the events generated on the HATS message window will be called *HATS* events, while those generated on the application window will be called application events. Application events that match with one of the next pending tasks, as well as the NextToDo button in the HATS message window, turn HATS mode into active, while, in general, HATS events turn the mode into descriptive. Context and alternative commands do not have any effect on the systems mode.

5.2 Descriptive Mode Navigation

When the initial message appears, the current help task is the application task *Add Fire Protection to a Column,* and its first atomic subtask, *Click on the Create Polyline Button* becomes the next pending task. Hence, HATS is in descriptive mode. We shall see now how the user can navigate from here and get information about the different subtasks of the application task.

Double clicking on the initial message will substitute it by a corresponding *task action message* (still a message shown in descriptive mode) obtained by substituting the second part of the previous one by an explanation of the subtasks that correspond to fire protection for columns, as follows:

You have requested help about how to ***Add Fire Protection to a Column.***
In order to do this task you have to fulfill the following subtasks:
Create a Single Protection Shape.

Fill the Region Using a Concrete Pattern.
Create a Mirror Region with respect to the Axis of the Column.

The user can now double click on any of the lines that explain one of the subtasks, and a corresponding *task description message* will appear. For example, if the user double clicks on the first subtask, the message will be:

You have requested help about how to ***Create a Single Protection Shape.***
You can do this task on the part of the window that is highlighted.

At any moment the user can also ask for contextualization by using the *In Context* item in the *Focus* menu, and a higher level of the task hierarchy is shown. The message below shows a contextualized action message for the previous task.

You have requested help about how to ***Add Fire Protection to a Column.***
In order to do this task you have to ***Create a Single Protection Shape.***
In order to do this task you have to fulfill the following tasks:
Click on the Create PolyLine button.
Mark a point *(This has to be done* ***5*** *times).*
Double click on the last point.

The *In Context* item of the *Focus* menu, together with the double clicking action, allows the user complete navigation through the task tree corresponding to the initial application task. Navigation is supplemented through the possibility to *Go To Next Task*. The default behavior of HATS allows the user to see description and action help messages for any subtask, together with a context that consists of a variable amount of its supertasks. The user can also choose the *verbose* behavior, that includes in the HATS message window the messages that correspond to all the subtasks s/he has explored at each moment.

5.3 Automatic Update in Active Mode

If the user starts the execution of the application task s/he is requesting information for, HATS switches from descriptive to active mode. Active messages are similar to task action ones, except for the fact that they speak about the actions that are being done, and the next action or actions to be performed. When switching to active mode, HATS shows the task context formed by the supertasks of the current help task that were not present in the previous context. In our example, once the user has finished the first subtask (creation of a single protection shape), the HATS window will show the following active message, that includes a precondition:

You have finished the task ***Create a Single Protection Shape.***
You have done this task on the highlighted object.
Next you have to ***Poach the Shape Using a Concrete Pattern.***
The following condition must hold: you must poach the shape you just created.
In order to do this task first you have to ***Click on the Fill Region Button.***

*In order to do this task you have to click with the **left** button of the mouse on the highlighted object.*

As the user follows the instructions on the HATS window, HATS updates them, showing her/him constantly the successive actions s/he has to perform. After the user has clicked on the Fill Region button, the help message will be:

*Next you have to **Choose a Pattern for the Protection.***
*In order to do this task first you have to **Select a Filling Pattern**. This can be done one or more times.*
*You can do this task on any of the objects highlighted in green by clicking with the **left** button of the mouse.*
*After this, you have to **Click on the OK Button**.*
*You can do this task on the object highlighted in yellow by clicking with the **left** button of the mouse.*

This message differs from previous ones in that the task being explained can be repeated an undetermined number of times. We have already seen a repetition task with a fixed number of repetitions. Repetition tasks are updated in a different way depending on whether the number of repetitions is fixed a priori or it can be determined by the user. In the first case the number of times the task still has to be performed is updated automatically in the messages generated by HATS; in the second case, the help message doesn't change until the user performs the atomic task that follows the one that can be repeated.

Finally, once the user has chosen the filling pattern, the help window will show:

*You have finished the task **Choose a Pattern for the Protection**.*
*Next you have to **Select the Shape to be Poached**.*
The following condition must hold: the selected shape must be the one you just created.
*You can do this task on the highlighted object by clicking with the **left** button of the mouse.*

In the previous message, the precondition on the next pending task is used to determine automatically that only one of the shapes can be selected, and that shape is the only one that is highlighted. The underlined part of the message is generated according to this situation.

5.4 Sequencing

The example we have studied is representative of a typical situation. The main situations that can arise besides those shown here correspond to different types of sequencing. As previously mentioned, ATOMS admits the specification of sequential, parallel, and alternative sequencing among subtasks of a given task. The previous subsections have been devoted to an example where only sequential subtasks have appeared. Next we shall outline the main features that characterize the help given by HATS in case of other types of sequencing.

In case of alternative sequencing HATS includes a message stating this, followed by a list of the alternative tasks, and, if in active mode, an explanation of how to start them. Double clicking on one of these tasks shows a corresponding description message for it. When in active mode, as soon as the user starts one of the alternative subtasks, HATS treats the situation as if that was the only alternative. The items in the Alternatives menu apply when the current help task is any of the mentioned ones, allowing the user to navigate through the different alternatives.

In case of parallel sequencing, HATS shows the list of subtasks, as in the previous case. Later on, in active mode it shows the task message corresponding to the task that follows the last atomic task that has been accomplished. When asked for context, HATS shows the list of those subtasks that are not finished, together with their corresponding next pending subtasks. HATS never shows information about nested alternative or parallel subtasks, unless it is in verbose behavior.

5.5 Design issues

The design of HATS help for a given application involves two parts: first, the task model has to be defined; after that, the designer must add help information to it. This information is formed by textual descriptions of tasks and preconditions, and the tasks that, when accomplished, will make preconditions to be satisfied.

Naming tasks, although simple, is an essential part of help design. Most parts of the messages previously shown that are highlighted with bold face are task names; they are inserted automatically by HATS in the corresponding help messages.

The descriptions of tasks and preconditions are just character strings that are included automatically by HATS in help messages when the corresponding tasks are being described. Previously we have seen messages that include these descriptions.

Parameter patterns are used to determine the parts of the window to be highlighted as complementary information to the help messages. They are specified in a way similar to task patterns (actually, both of them are instances of *Object Patterns,* a general object prototype that can be matched against other objects in the underlying object system). Moreover, preconditions are used as filters to the list of parts to be highlighted.

The reduced HATS design environment just allows task names and descriptions that appear in the HATS messages window to be edited. This is an essential feature, since usually, when visualizing help messages in context for the first time, the help designer finds out that the names and descriptions of many tasks, as well as those of preconditions, are not appropriate. All the remaining information included in help messages is automatically generated.

Tasks hierarchies and their associated help information can be reused in different applications. Tasks and their components are objects in the AMULET object system. Hence, each task is a potential prototype for instances of it, that inherit its properties, including help information. For example, multiple tasks are instances of a *multiple task* prototype that includes additional information like the number of repetitions.

A higher degree of reusability can be obtained by associating tasks to widgets. HATS based buttons include an instance of the *Click on the $Label button* task prototype. Their name is automatically generated using the label of the corresponding button. Dialog boxes are among the widgets for which defining

corresponding tasks that include help information can be useful. For example, the dialog box for the selection of filling patterns in the example described at the beginning of this section is an instance of a generic widget for pattern selection, that includes a corresponding selection task.

5.6 HATS Architecture

The HATS system works in conjunction with the application it provides help for, and with ATOMS. Figure 5 shows the overall architecture. The system is driven by user events. Application events and the flow of information they generate are represented by solid arrows, while dashed arrows correspond to HATS events.

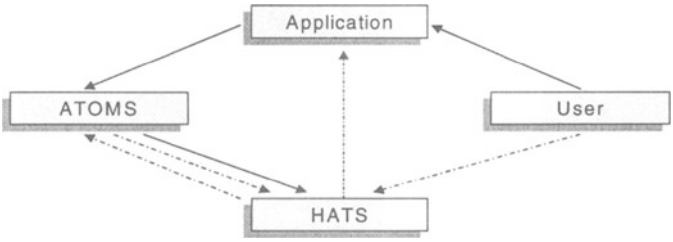

Figure 5. HATS framework.

As the diagram shows, both types of events are sent to HATS after passing through the ATOMS manager. HATS then displays help messages in its own window, and sends highlighting requests to the application window.

The generation of help messages is done as follows: according to what has just been said, HATS receives its input from ATOMS, that indicates the kind of event that has taken place (HATS or application), together with the relevant information at the task level, including any possible changes that affect one of the next pending tasks. HATS uses the information given by ATOMS in order to set a new state, and then a grammar is used for the generation of the messages.

TaskMsg = TaskDescrMsg | TaskActionMsg | TaskActiveMsg
TaskDescrMsg = [Context] [TaskContentsMsg] TaskLocMsg
TaskActionMsg = [DescrContext] ActionMsg
DescrContext = (IntroMsg | TaskContentsMsg | TaskLocMsg) (TaskContentsMsg | TaskLocMsg)*
IntroMsg = You have requested help about how to **TaskName**
TaskContentsMsg = TaskRef **TaskDescr**
TaskRef = The task **TaskName** | IndTaskRef
IndTaskRef = this task | its [first | last] subtask [,] **TaskName** [,]
TaskLocMsg = You can TaskAction on TaskLocDescr
TaskAction = **TaskName** | do IndTaskRef
ActionMsg = In order to TaskAction you have to ActionDescr
ActionDescr = fulfill the following tasks: **TaskName*** | EventDescr on TaskLocDescr
EventDescr = [Double] Click with the **ButtonPosition** button of the mouse on TaskLocDescr

TaskActiveMsg = [ActiveContext] ActiveMsg
ActiveContext = [FinishedMsg] ActiveDescrMsg+
FinishedMsg = You have finished the task **TaskName.** You have done it on
TaskLocDescr
ActiveDescrMsg = [OptPendingDescr] You are accomplishing the task **TaskName**
| [OptPendingDescr] PendingDescr **TaskName.**
OptPendingDescr = In order to do this task
PendingDescr = (First | Next | Finally) you [still] have to
ActiveMsg = PendingDescr EventDescr on TaskLocDescr

Figure 6. Grammar for help message generation (non verbose version).

Figure 6 shows a portion of the grammar, that is relevant for the example described in this section. The parts of the grammar that deal with keyboard events, sequencing, graphical references, and repetitions are omitted. The nonterminal elements of the grammar, encircled in a rectangle, represent single or composite messages; task names and other information accessible from each individual task are typed with bold letters, as it happens in the messages we have seen, and nonterminal elements that correspond to graphical references are underlined.

HATS includes a planner that chooses among the alternatives that appear explicitly in the grammar, and those that are implicit in references to graphical objects. For example, a task can be referred by its name or by an indirect reference like *this task*, depending on whether its name has appeared recently in the message. In the same way, the help message can refer to graphical objects or to a region where they are supposed to be, depending on the information HATS has at generation time. In case there is more than one graphical reference in the message, different colors are used, and corresponding references are included in the text.

6. CONCLUSIONS AND FUTURE WORK

We have presented a system for the generation of task oriented help for interactive applications, that allows the design of the help component in a simple way once a task model for the application has been specified. The type of help provided by the system has many advantages, including the fact that help messages and instructions are updated automatically as the user accomplishes the corresponding tasks, and that they include references to parts of the window where actions can be performed. The system provides support for context sensitive complex tasks; this is achieved by means of a task model that includes task parameters.

Future work is aimed in three main directions: allowing the interactive specification of the task model for the application and the corresponding help information, developing a more sophisticated tool that allows the generation of tutorials about interactive applications that keep track of the student's work and interact with him, and extending our ideas to a distributed environment. The development of these plans also presents challenges about the integration with techniques developed in other user support systems, like Brad Myers' Topaz macro system (Myers, 1998), and Romain Zeiliger and David Kosbie's ACCEL distributed task system (Zeiliger, 1997).

7. ACKNOWLEDGMENT

This work has been partially supported by the Spanish *Plan Nacional de Investigación*, project number TIC96-0723-C02-01/02.

8. REFERENCES

Bhavnani, S.K. and John, B.E. (1996), Exploring the Unrealized Potential of Computer-Aided Drafting, in *Proceedings of CHI 96,* Vancouver, BC Canada, ACM Press.

Contreras, J. (1998) A Framework for the Automatic Generation of Software Tutoring. Phd. Thesis, Université René-Descartes, Paris.

Moriyón, R., Szekely, P. and Neches, R. (1994) Automatic Generation of Help from Interface Design Models, in *Proceedings of CHI'94*, ACM Press.

Myers, B.A., McDaniel, R.G., Miller, R.C., Ferrency, A., Faulring, A., Kyle, B.D., Mickish, A., Klimovitski, A. and Doane, P. (1997) The Amulet Environment: New Models for Effective User Interface Software Development. *IEEE Transactions on Software Engineering*, **23-6**, pp. 347-65.

Myers, B.A. (1998) Scripting Graphical Applications by Demonstration, to appear in *Proceedings CHI'98*, ACM Press.

Pangoli, S. and Paternó, F. (1995) Automatic Generation of Task-oriented Help, in *Proceedings UIST'95*, Pittsburgh, ACM Press.

Rodríguez, P., García, F., Contreras, J. and Moriyón, R. (1997) Parsing Technologies for User-Task Recognition, in *Proceedings of 5th International Workshop on Advances in Functional Modeling of Complex Technical Systems* (ed. M. Modarres), forthcoming.

Sukaviriya, P. and Foley, J.D. (1990) Coupling a UI Framework with Automatic Generation of Context-Sensitive Animated Help, in *Proceedings UIST'90*, ACM Press.

Szekely, P., Luo, P. and Neches, R. (1992) Facilitating de Exploration of Interface Design Alternatives: The HUMANOID Model of Interface Design, in *Proceedings SIGCHI'92*, ACM Press.

Zeiliger, R. and Kosbie, D. (1997) Automating Tasks for Groups of Users: A System-Wide "Epiphyte" Approach, in *INTERACT'97* (ed. S. Howard, J. Hammond and G. Lyndgaard), Chapman & Hall Press, IFIP, Sydney.

9. BIOGRAPHY

Federico García is a Ph.D. student at the Department of Computer Science of the U. Autónoma de Madrid since 1996. He is interested in tools for task model management.

Javier Contreras teaches at the Department of Computer Science of the U. Autónoma de Madrid since 1995. He is a former researcher at the Madrid IBM SC. He obtained his Ph.D. in U. Paris V, developing a tutoring system.

Pilar Rodríguez teaches at the Department of Computer Science of the U. Autónoma de Madrid since 1996. She is a former researcher at the Instituto de Ingeniería del Conocimiento and at the Madrid IBM SC. She holds a PhD in Physics.

Roberto Moriyón is Professor of the Department of Computer Science of the Universidad Autónoma de Madrid since 1980, and a researcher at the Instituto de Ingeniería del Conocimiento since 1989. He holds a PhD in Mathematics from Princeton University.

Discussion

Claus Unger: In your example, the task to be performed by the user is defined in a very precise way: why can't the user simply

- select the columns to be protected
- select the shape of the fire protection
- select the fill pattern

Federico Garcia: Because what the users usually want is to learn how to use a computer program. They usually have advanced domain knowledge from paper drawings, and they want to perform the same tasks with a computer. What we propose is not a tool for automating some common tasks, but a way to help users to perform common tasks in terms of simpler ones within the standard working context, so that later they are able to perform other similar tasks by themselves. If we just automated the execution of specific tasks, they wouldn't be able to learn how to perform similar ones.

Claus Unger: To my understanding, there is a fundamental difference between a tutorial system and an active help system. Your example more or less emphasizes the tutorial aspect of your approach.

Federico Garcia: Yes, there are similarities between our system and tutorial systems. Actually, our work is the basis for a more advanced tool whose focus is to provide designers with an environment for the creation of software courses. So, HATS is the first approach to address the problem of easily generating software courses, and it incorporates a guidance quite similar to tutoring guidance.

Fabio Paterno: I would prefer a task system that when the user performs a disabled action provides task-oriented explanation of why that action is disabled, and how to enable it, rather than a task system that explains which tasks cannot be done

Federico Garcia: The problem with our system is that it cannot detect such input events, only events that the underlying AMULET system sends to the application, so we can not receive that kind of input about disabled widgets.

Philipe Palanque: Couldn't you even get mouse positions?

Federico Garcia: Yes, that's possible, but then we wouldn't get one of AMULET's main benefits, which is portability over different platforms.

Prasun Dewan: Doesn't your task generation facility provide what Claus wanted?

Federico Garcia: I would reformulate the question as follows: Doesn't your HATS system provide automation for a task once the system has realised which task the user is currently performing?

Yes, but our objective is not to auto-execute but to instruct. Moreover, in many cases we cannot realise which task the user is performing until the final steps. For example, you cannot know which column the user wants to apply fire protection to until he chooses it. So, once the system knows the next concrete action, it can offer to perform it. Actually, in the current

implementation, HATS provides a facility for automatic execution of tasks when all their needed parameters are bound.

Jean Scholtz: Do you have any plans to do user studies on your help system

Federico Garcia: Not yet. The system has been tested by some partners, but we agree that is not enough of a representative population.

Jean Scholtz: Oh, well please clarify the difference between descriptive help mode and active help mode.

Federico Garcia: In Descriptive help mode, the system provides meaning and hierarchical task decomposition, while in Active help mode more concrete and widget-oriented explanations are given, explaining only the next action to be performed in order to achieve the task that is currently being explained.

Joelle Coutaz: Do you plan to provide help within the application workspace (as in the MAC OS finder) rather than in a separate window (as in your current system)?

Federico Garcia: We've not considered that. The problem is that it is easier to provide that kind of integrated messages for a widget-oriented help system, while for a task-oriented help system the given explanations are usually more sophisticated and, as a consequence, longer and more difficult to integrate with the application interface. Of course, a separate window wouldn't be needed and an integrated pane could be added to application window.

Joelle Coutaz: Do you plan to work on helping the user to ask for help? The problem is that it is hard to formulate questions to the help system

Federico Garcia: Our aim is not to focus on how a user can ask for help on some topic. Currently, the user selects a task from a hierarchically sorted list of modelled tasks, or by hitting the help-key after beginning to perform a specific task.

Joelle Coutaz: That would constrain the vocabulary

Federico Garcia: Of course, ideally a user would request help by writing a question using natural language, and the system should have to extract the semantic information from that question to provide explanations. But this is out of our scope at this moment.

Laurence Nigay: You spoke of exploring groupware help - how would you specify tasks for groups?

Federico Garcia: The task modelling language we have developed to specify task models should be augmented to incorporate the notion of users and groups. From this notion, things such as constraints about what users (or groups) are allowed to perform some steps, or who has to perform some activities and under which circumstances, should be allowed to be expressed with our task-modelling language. Our current model includes parallelism among tasks, so extending our event parser to a distributed environment would not be a big problem.

Claus Unger: The same task can be performed in many different ways. Have you considered more flexible representations such as precedence graphs?

Federico Garcia: There is the need to reach a trade off between a representation flexible enough to express many different alternatives and a representation easy to provide help from. Task models may have powerful capabilities by using both pre and post conditions and different sequencing in a correct way. Moreover, the hierarchical decomposition of this representation provides a semantic content not present in precedence graphs. An additional advantage of task models is its growing use from the design point of view, being present in many current model-based development environments.