A Framework for the Automatic Generation of Software Tutoring

Javier Contreras and Francisco Saiz

Instituto de Ingeniería del Conocimiento, Universidad Autónoma de Madrid,
Cantoblanco 28049, Madrid, Spain
Phone: +34-1-397.39.73 – Fax: +34-1-397.85.44
E-mail: {contrera, saiz}@lola.iic.uam.es
WWW: http://lola.iic.uam.es/~contrera

Abstract

Interactive Systems present an ever increasing complexity both to their users as well as to their designers. These systems may require a great effort to be mastered by a new user. Therefore, some kind of tutoring for these applications must be provided, in such a way that it does not represent duplicating the work of the designer. This paper describes an approach for automatically generating a tutoring system for the tasks defined in an application, using some particular information on tutoring that these tasks may have. At the same time an editor of these tasks is provided to the designer. The kind of tutoring automatically generated has a variable degree of flexibility in face of user actions, according to the designer's criteria, and it is performed using the real application, not a simulation. This means that the final user can actually work while he is learning how to perform a task. The ideas here presented have been implemented in two small prototypes, Teach me While I Work (TWIW) and Task Models Editor (TME).

Keywords

Software tutoring, task models, user interface design, interactive systems specification, programming by demonstration.

Introduction

As Interactive Systems become more and more complex, more help must be provided to the final user. Standard help systems will soon become obsolete, since they are very rigid and static, not taking into account the dynamic aspects of the Human-Computer Interaction, nor the context in which help is demanded by the user. In this scenario, flexible and powerful tutoring systems seem more appropriate in order to instruct a new user of an Interactive System. But achieving higher flexibility and power for tutoring systems involves an ever higher cost. In this paper we will present an approach that allows the designer of an Interactive System to incorporate a tutoring system for his application in a very easy manner. Our approach has been tested in two prototype tools, TWIW and TME.

TWIW is the tool in charge of generating the tutoring at run-time. To do so, it has to access information that is contained in task objects. As we shall see, these are the appropriate objects for sharing the information between an application and the correspondent tutoring system. These objects have a much richer semantic information than single commands can have.

Moreover, the kind of tutoring generated has great benefits for the final user, as he will receive tutoring on the tasks that can be performed in an application at the same time he is working and actually carrying out these same tasks. This allows him to practice progressively those tasks he has already learned, and also to incorporate its use into new interaction techniques he learns. This is in sharp contrast with current tutoring techniques, as we shall see in the Related Work section.

To work, TWIW needs that the tasks of the application that are going to be tutored have been defined. This must be done by the designer of the application. To avoid increasing the burden of the designer, TME is provided. It is an editor of tasks that incorporates techniques of Programming by Demonstration, that turns the edition of the task models into a very easy commitment.

Regarding the working environment, both TWIW and TME were developed according to the Model-Based Interface Design paradigm, using the HUMANOID tool [Szekely92], although we needed to introduce slight changes in the original language [Saiz95]. The model-based approach in HUMANOID leads to an interface design environment that supports design reuse, delay of design commitments and help to understand design models; benefits not found in current interface building tools [Luo93]

One of the main benefits for working using the Model-Based Paradigm has been already exploited by other systems too [Luo93, Moriyón94]: it is easy to reason about the models. In our case, TWIW reasons about the task models, and TME creates interactively these same models. The techniques used by TME can also be found in [Cypher93].

Although a model-based approach provides the benefits above, it does so at a cost of additional specification effort. To avoid this, both TWIW and TME were built using the KIISS editor [Saiz96], which in turn lies on top of HUMANOID.

The paper is organized as follows: we first describe related work, followed by a general overview of our system and a more detailed section on its architecture. We then present an example, showing the kind of tutoring a user would receive while working. Then we introduce the TME editor, with another example in which the designer specifies tasks for his application. Finally we extract some conclusions and point out relevant ideas for future work.

1 Related Work

We shall start by considering the development of help systems, a related field where a lot of activity has taken place for many years both in the development of commercial applications and in the search of new techniques that enhance the capabilities of previous systems. Help systems can be considered as the most simple software training tools that provide the user information about how to accomplish tasks with the software at hand.

During the last years, with the advent and wide spread of graphical user interfaces, graphical help systems have become usual, and all the main suppliers of such applications have developed their own help systems. Apple, as a pioneer in this direction, has used Balloon Help already for a long time [Macintosh95]. Microsoft has relied more on hypertext style, and its development environments like Visual Basic© or Visual C++© include specific means for the construction of help for applications [Microsoft91].

More recently, Microsoft has also included the possibility to add to an application small rectangles of explanatory text that appear when the mouse stays long enough over a specific region of the screen, or when the user clicks on it after asking for help. All these interactive help systems are characterized by their static nature (they can only give help about a fixed predetermined portion of the screen), and by the fact that the only information they can give for complex tasks is through explicit text explanations. As an example, we consider the following help the user receives when asking how to create a link when using the CorelDRAW© program:

To link an object from CorelDRAW:

Choose Insert Object from the File menu.

1. Select Create from File.
2. Select Link.
3. Type the name, including the path and extension, of the file you want to link. If you don't know the name of the file or it's location, click the **Browse** button to display the Browse dialogue box.
4. If you want the object to appear as an icon, select **Display as Icon**. The icon that's currently associated with the selected application appears. You can choose another icon from the dialogue box displayed by clicking the Change Icon button.
5. Choose OK.

This type of recipe-help is difficult to follow by a new user.

During the last years there has been some research in the HCI community related to the development of high level tools for the construction of help systems. They can be seen in some sense as ancestors of the system we present in this paper, so we shall explain briefly their more relevant aspects.

N. Sukaviriya [Sukaviriya90] has developed a system that is able to give context sensitive help with animations about a user interface. For example, the user can ask how to read mail in a mail tool, and the system will answer with an animation showing how to select a message from the list using the mouse and then click on the Read button. Sukaviriya's system is based both on a backwards chaining inference engine and a facility for the animated simulation of mouse events. The backwards chaining engine uses pre and postconditions associated to interactive actions to search for a chain of events that can accomplish the desired task, and then shows it using the corresponding animations.

R. Moriyón, P. Szekely and R. Neches [Moriyón94] have developed another tool for the generation of help systems, HHH. It produces automatically a help system for any application built using HUMANOID, a high level tool for the development of user interfaces from the University of Southern California. It also allows the designer to change the resulting help system to adapt it better to his specific application. HHH generates automatically help messages in hypertext form that include references to different parts of the interface. To do so it uses a forwards chaining inference engine. HHH messages are context sensitive, and adapt themselves dynamically to the state of the interface.

Finally, Pangoli and Paternó [Pangoli95] have a system that is able to generate help about the achievement of complex tasks. This system is built on top of a tool for the specification of user interfaces that includes the ability to define user tasks. What their work has in common with ours is the identification of the user tasks as being essential in giving high level help (tutoring, in our case) with important semantic content.

Comparing these systems with TWIW, both Sukaviriya's and HHH systems give a very close to atomic interaction help, being the user the one to decide how to split a complex task in a sequence of simpler subtasks. This is not the case with Pangoli's system, that relies on user tasks hierarchies to achieve better help generation. But in contrast, this system does not permit the user to have guided practice, with immediate feedback, as is the case with TWIW.

With respect to software tutoring systems, the state of the art is still far from a situation like the one we have just described for help systems. There are no specific tools that simplify the development of courses on the use of software. Some exceptions are a few projects that are under way for the development of documentation for applications (ARPA is financing such a project at ISI/USC), and general frameworks for course development. All of them ignore completely the specific characteristics and possibilities derived from the fact that the subject of the course is itself a computer program.

Most of the software tutoring systems that exist nowadays include just some kind of "movie" that shows how the system accomplishes determined tasks, including an animated view of the movements of the mouse, etc. The most advanced software tutoring systems allow the student to practice using an emulation of the real software being tutored. But building this emulator is a costly process, that is more costly as more complex the emulator ought to be, and usually the student has the feeling that he is dealing with a canned version of the application. In particular, he can never save the result of his work, and starting a tutoring session is something clearly differentiated and isolated from working with the application. Complex activities like the ones arising in engineering processes (design, emulation, etc.) can be learned better in a more flexible tutoring environment like the one we describe in this paper.

2 TWIW: Teach me While I Work

TWIW is able to do tutoring on applications that incorporate a model of the tasks that can be accomplished with them. The tasks that are part of an application constitute a hierarchy, where complex *root tasks* are decomposed into simpler ones, and so on until *atomic tasks* are reached.

Each task incorporates information about how it will be tutored. TWIW constructs automatically a default tutoring system for each application by adding to it standard information about its tutoring method. The designer of the application can modify and refine the tutoring system automatically generated by modifying the tutoring information corresponding to its tasks.

A special task provided by TWIW, the *Tutoring-Task*, takes care of the generic aspects related to tutoring. TWIW also incorporates a *Task Manager* that is responsible for the execution of the appropriate actions when the user interacts with the application.

The overall structure of the tasks model of an application can be highly complex. For example, several tasks can be accomplished in parallel. Hence, a tutoring system must include a module that allows the user to learn about this complexity by showing him the different tasks he can perform at a given moment and their relationship. TWIW incorporates several standard windows that show the user different views of the task system and its current state.

Figure 1 is just one of these windows, the one that shows a list of all the root tasks defined in a Mailtool application. This is the first window shown when a user expresses his intention to learn about tasks by hitting the appropriate key. By using the menu-bar the user can change this visualization, to obtain only the list of currently *active tasks*. By selecting individual tasks on any of these windows, suitable tutoring actions can be performed on them.

On the other hand, individual root tasks can also show arbitrary big complexity by the nesting and branching of their associated tree. TWIW can help the user to understand these aspects by showing him individual information about specific tasks. The actual information TWIW is able to give right now in this respect is very similar to the one included in [Pangoli95], which in turn is closely related to the information given by Sukaviriya's Help System [Sukaviriya90].

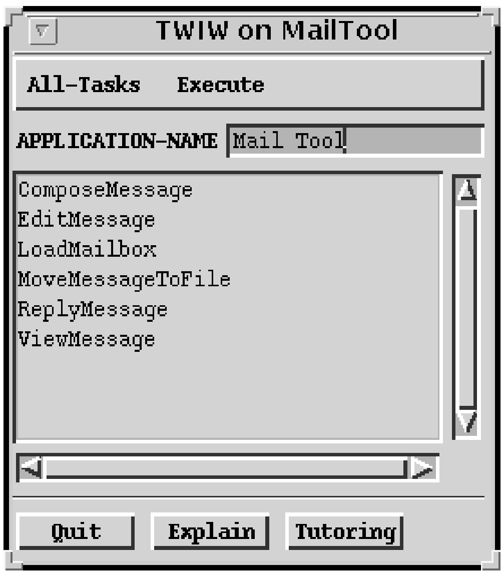


Figure 1. Main window of the TWIW system

It consists of a tree shown in a dedicated window, from which the user can ask for a textual description of individual subtasks, as well as to flash the parts of the screen related to each step of the task under consideration. This kind of help has proven to be much more effective than the static help we can find in many applications, describing all the commands available, but without direct reference to where we can invoke them, and how we can assemble them to perform a complex task.

Taken by itself, the functionality described so far of TWIW does not deserve the name of tutoring functionality. It would be best described as advanced help functionality. It is the features explained in the next paragraph that makes TWIW a powerful tool for the automatic construction of tutoring systems.

The fundamental aspect of TWIW, that allows real tutoring of interactive applications, takes place when the user asks for training on a specific task. The user will then receive complete information about the selected task and how to perform it.

After that, while the user tries to perform the task interacting with the original application, TWIW will watch if the correct steps of the selected task are being taken, and in the right order. According to the tutoring-information associated to this particular task, TWIW will do the following:

- if the action is correct, it will execute it, look for the next step on the task model and perform a preliminary tutoring action associated with it, typically showing an explanation about what should be done next;
- if the action does not correspond to what is expected, two things can happen: if the Tutoring-Task is in *strict mode* (which is the default tutoring behaviour) an error message will pop up in a window, explaining what the user was supposed

to do, and the action will not be executed. On the contrary, if the Tutoring-Task is in *free mode*, a warning message will be displayed, but the action will be executed. Intermediate modes are also available, leaving to the designer the decision about where, how, and when a user can act while learning a certain task.

Of course we can abandon tutoring at any moment, or simply reach the end of the task. This simple tutoring style is complemented by some variations of it where, for example, the system teaches a whole course about the application or it makes the user train randomly different tasks. In this case, it can follow the user's accomplishments and take this into account in the successive training proposals it makes.

3 Architecture

The ideas here presented can be implemented in any environment that supports object-oriented programming and allows the specification of the interactive behavior of graphical objects, encapsulating the low-level events produced by the user activity.

We have chosen HUMANOID [Szekely92] to implement both TWIW and TME. HUMANOID is a model-based interface design and construction tool where interfaces are specified by a declarative description (model) of their presentation and behavior.

The main components of the TWIW tool, necessary to implement the approach described above are:

3.1 Task Models

Our prototype includes a simple task model that has allowed us to concentrate on how to perform tutoring on tasks, and at the same time control the user activity. There are two basic types of tasks: atomic tasks and composite ones. Both are KR objects, the language used to define objects in HUMANOID.

With them the designer builds the task hierarchy of the application, the atomic tasks being the leaves of the tree. These atomic tasks are directly related to atomic interactions coming from the user, via the HUMANOID *behaviors*.

The task objects also have tutoring information. This information can be specified by the designer, or he may choose to use the default behavior. The knowledge contained here is used by the Tutoring-Task, and basically determines:

1. where the user can interact with the application while doing tutoring on some task;
2. pre and post-actions for each node of the tree, used normally to guide him through the task and provide feedback;
3. how the system should behave if the user action was not expected.

3.2 Task Manager

This component incorporates an interpreter of the user's actions with respect to the tasks defined. This is a delicate point due to the asynchronous character of the user's activity.

Although the Task Manager is an essential part of TWIW, it is sufficiently general as to be incorporated in other systems that modify the behavior of applications. As an example of such a modification, one could build a system that helps in the debugging of an application, just by providing a task object specialized on that, or in other words, substituting the Tutoring-Task by a Debugging-Task. The rest of the components of TWIW would work exactly the same with this new tool.

3.3 Tutoring Task

This is an atomic task provided by TWIW. All the behavior described above, that takes place when the user is in tutoring mode, is encapsulated within this atomic task. Besides that, this atomic task is similar to the others, and it is treated in exactly the same way by the Task Manager.

When the user enters tutoring mode, the Tutoring-Task is activated, while all the other application tasks are deactivated. In this way, while in tutoring mode, this task is emulating the others according to the user activity and the states of the other tasks. In figure 2 we can see how these components are assembled in TWIW.

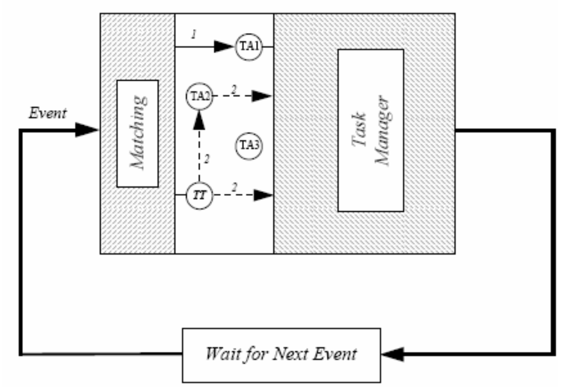


Figure 2. Architecture of TWIW

When the user tries to interact with the application, an event arrives to the system. TWIW will first try to match this event with all the atomic tasks that can be activated. If the matcher can actually associate the input event with an atomic task, indicated in the figure by number 1, this information will be passed to the Task Manager, that will execute the action specified by the atomic task.

If we are in tutoring mode, the single atomic task that can be activated is the Tutoring Task, TT in the figure. This is the situation labeled by number 2. The Tutoring-Task will watch for the state of the task being tutored, and decide if it is an appropriate action from the user. If this is so, it will emulate the atomic task concerned, in this case TA2, and perform some actions to guide the user with the tutoring information contained in TA2.

What we want to stress here is that the tutoring behavior is contained in the Tutoring-Task. The Task Manager and the matcher are absolutely independent of this. The information is distributed in such a way that we could easily modify the behavior of an application putting some knowledge related with our goals in the task objects and creating a new task that knows how to deal with this knowledge. The rest would work exactly in the same way.

4 An Example

In figure 3 we can see two task objects defined for a CAD application. If the user decides to receive tutoring on the "Give Shadow to a 3D Object" task, TWIW will display a window with the message:

"This task permits you to give shadow to a 3D object selected in the design area"

and an OK button to continue. This information is contained in the root task. Then it will display another window with the message:

"First you have to select the object you want to shadow, that must be a 3D object. This can be done by selecting the Pointer in the ToolBar and then marking a zone that completely contains the object desired"

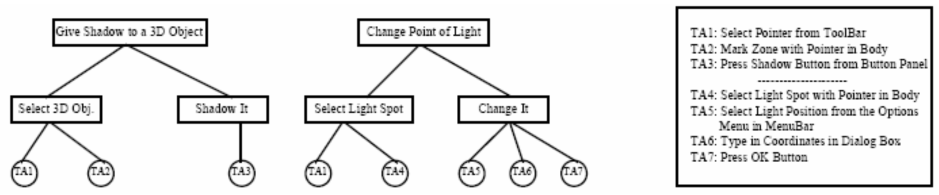

Give Shadow to a 3D Object
Select 3D Obj.
Shadow It
TA1
TA2
TA3
Change Point of Light
Select Light Spot
Change It
TA1
TA4
TA5
TA6
TA7
TA1: Select Pointer from ToolBar
TA2: Mark Zone with Pointer in Body
TA3: Press Shadow Button from Button Panel
TA4: Select Light Spot with Pointer in Body
TA5: Select Light Position from the Options Menu in MenuBar
TA6: Type in Coordinates in Dialog Box
TA7: Press OK Button

Figure 3. Two tasks defined in a CAD application

This second window has two buttons: The OK button and the Flash button, who will highlight the widgets referred to in the explanation, if the user desires. This information is contained in the first subtask. Then it is the moment for the user to act.

If he tries to do a different thing from what he was told, and the system is in strict mode, his action will not be executed and he will receive an error message explaining the situation and the same explanation as before about what he is supposed to do. This is usually the case if it is the first time the user tries to accomplish this task, even if there are related tasks as the one we show in the second tree of figure 3.

Later, if the user has already mastered the first task, or meets any other criteria specified by the designer, the user could be allowed to change position of light as he receives tutoring on the first task. In this case, the system would be in an intermediate mode between free and strict. If the user begins executing the actions specified in the second task, he would receive a warning message, informing him that his action is not directly related to the tutored task, but this action would be performed.

In this case he would be able to change the point of light in the graphics application and then continue with the tutoring of the tutored task, about giving shadows to 3D objects.

5 TME: Task Models Editor

Once the designer has finished to create his application, he must define the tasks that are available to the user and that will be used by TWIW to generate tutoring. The designer can specify the task models using an editor and the HUMANOID language, exactly as he has done previously to define the application.

Proceeding this way there is an easy part, namely the one related to slots which content is textual (e.g., name, description, etc.).

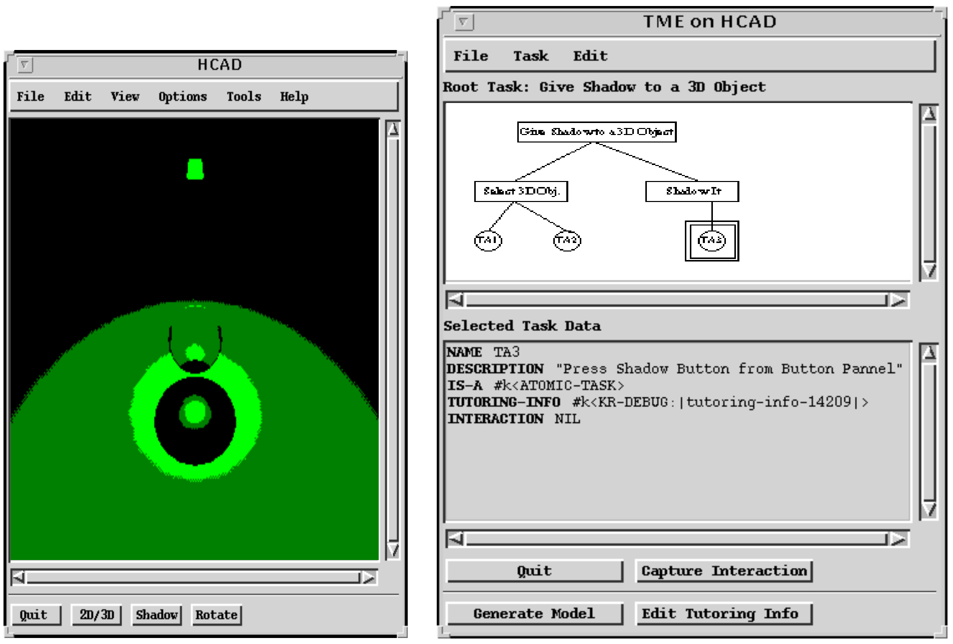
The difficult part arises when the designer is defining the atomic tasks and has to specify what interaction from the user is associated with this task. To do so, the designer would have to know the name of a big number of HUMANOID behaviors used in is application. TME was created to overcome this requirement. It provides the designer of the application with:

- a graphical way to define the tasks' hierarchy, by means of a tree of nodes labeled with the name of the tasks he defines;
- a convenient way to bind user actions with atomic tasks, by means of programming by demonstration techniques. When the designer needs to make such an association, s/he expresses so to TME. Then he can directly interact with the interface of his application (that is present all the time) and do what the final user is supposed to do. TME will capture this event, do the corresponding translation to obtain the appropriate behavior and will introduce it in the atomic task that is being defined.

When the designer has finished editing the task models he can ask TME to generate the corresponding HUMANOID code, homogeneous with the rest of the application.

6 Another Example

In this example we can see a small CAD application created using the KIISS editor on top of HUMANOID. The designer is specifying the task models of his application, using TME. In figure 4 we see both interfaces. The upper part of TME contains the hierarchy or tree of the root task that is being edited.



```
NAME TA3
DESCRIPTION "Press Shadow Button from Button Pannel"
IS-A #k<ATOMIC-TASK>
TUTORING-INFO #k<KR-DEBUG:|tutoring-info-14209|>
INTERACTION NIL
```

Figure 4. Defining tasks in an application using TME

Using the Edit menu, the designer can cut and paste tasks previously defined and introduce them in the current one. When he selects a node from the tree, a list of slot-value pairs corresponding to the selected task appear on the bottom part of the editor. The slots that contain textual information, e.g., name, description, etc. can be edited in this part of the window.

When the task that is being edited is an Atomic Task the Capture Interaction button is activable, otherwise it is dummy, since Composite Tasks do not have the interaction slot. In the figure, the designer is defining the Atomic Task TA3 and is going to set the interaction slot of this task pressing directly the button in the HCAD application he wants to refer to. TME will capture this behavior and introduce the correct value for this slot.

Conclusion

We have seen an approach to the automatic generation of software tutoring systems starting from a description of tasks. This relieves the designer from the burden of the infinitude of small details that these systems must take into account and allows him to concentrate on the most conceptual aspects of the tutoring by specifying the tasks. To avoid programming at this level, a tool called TME is provided.

On the other side, the kind of tutoring provided is essentially of a new type, as learning how to use an application and really working on it can be done at the same time. This represents a great saving in time with respect to those systems that need a previous and separated training phase, without loosing safety, since in our system, in tutoring mode the user's activity is supervised. In current systems the user must receive tutoring with pre-prepared examples, not necessarily related with his own work, and after that he must try to apply what he has learned.

As possible extensions to our system, we consider:

- extending the task models, including a more sophisticated behavior and sequencing. In this work we have concentrated on the tutoring information the tasks of an application must have to be used by a tool similar to TWIW. Real applications have a more complex decomposition of tasks than the one we have used. This includes the possibility of defining alternative tasks to achieve the same goal, specifying tasks that can be accomplished in parallel, etc.
- adapting the tutoring not only to the tasks, as it is the case now, but also to the user. The tutoring could be more guided if our system had a model of the student. This would be possible if we incorporate the work done in [Kobsa90].
- an improvement in the TME usability would consist in the possibility of specifying all the interactions sequentially instead of how it is done now, where the designer must select each time from the tree (the upper part of TME) which task he is editing.

Acknowledgements

We gratefully acknowledge the key role Roberto Moriyón has played in helping us develop these ideas, as well as Ricardo Orosco who helped us with the images of this paper. Finally, we would like to thank the anonymous reviewers who contributed in a great deal in the quality of this presentation.

This work was partially supported by the Plan Nacional de Investigación, Spain, Project Number TIC93-0268.

References

[Cypher93] Cypher, A. (Ed.), *Watch What I Do: Programming by Demonstration*, The MIT Press, Cambridge, 1993.

[Kobsa90] Kobsa, A., *Modeling the User's Conceptual Knowledge in BGP-MS, a User Modeling Shell System*, Computational Intelligence, Vol. 6, 1990.

[Luo93] Luo, P., Szekely, P., Neches, R., *Management of Interface Design in HUMANOID*, in Proceedings of the Conference on Human Factors in Computing Systems INTERCHI'93 « Bridges Between Worlds » (Amsterdam, 24-29 April 1993), S. Ashlund, K. Mullet, A. Henderson, E. Hollnagel, T. White (Eds.), ACM Press, New York, 1993, pp. 107-114. http://www.isi.edu/isd/CHI93-manager.ps

[Macintosh95] *Macintosh System 7*, Apple Computer. 20525 Mariani Ave. Cupertino, CA 95014, 1995.

[Microsoft91] *Microsoft Visual C++*, Microsoft Corporation, One Microsoft Way, Redmond, WA 98052, 1991.

[Moriyón94] Moriyón, R., Szekely, P., Neches, R., *Automatic Generation of Help from Interface Design Models*, in Companion of the Conference on Human Factors in Computing Systems CHI'94 « Celebrating Interdependence » (Boston, 24-28 April 1994), C. Plaisant (Ed.), ACM Press, New York, 1994, pp. 225-231. http://www.isi.edu/isd/CHI94-Help.ps

[Pangoli95] Pangoli, S., Paternó, F., *Automatic Generation of Task-oriented Help*, in Proceedings of the 8th Annual Symposium on User Interface Software and Technology UIST'95 (Pittsburgh, November 1995), G.C. van der Veer, S. Bagnara and G.A.M. Kempen (Eds.), ACM Press, New York, 1995, pp. 181-187.

[Saiz95] Saiz, F., Contreras, J., Moriyón, R., *Virtual Slots: Increasing Power and reusability for User Interface Development Languages*, in Proceedings of the Conference on Human Factors in Computing Systems CHI'95 « Mosaic of Creativity » (Denver, 7-11 May 1995), I.R. Katz, R. Mack, L. Marks, M.B. Rosson, J. Nielen (Eds.), ACM Press, New York, 1995, pp. 236-237.

[Saiz96] Saiz, F., Contreras, J., Moriyón, R., *KIISS: a system for interactive modification of model-based interfaces*, IIC Research Report 06-96, 1996.

[Sukaviriya90] Sukaviriya, P., Foley, J.D., *Coupling a UI Framework with Automatic Generation of Context-Sensitive Animated Help*, in Proceedings of the 3rd Annual Symposium on User Interface Software and Technology UIST'90 (Snowbird, 3-5 October 1990), ACM Press, New York, 1990, pp. 152-166.

[Szekely92] Szekely, P., Luo, P., Neches, R, *Facilitating the Exploration of Interface Design Alternatives: The HUMANOID Model of Interface Design*, in Proceedings of the Conference on Human Factors in Computing Systems CHI'92 « Striking a balance » (Monterey, 3-7 May 1992), P. Bauersfeld, J. Bennett, G. Lynch (Eds.), ACM Press, New York, 1992, pp. 507-514. http://www.isi.edu/isd/CHI92.ps