Automatic Generation of Task-oriented Help

S.Pangoli, F.Paterno'
CNUCE - CNR
Via S.Maria 36
56126 Pisa, Italy
Tel: +39 50 593289
E-mail:F.Paterno@cnuce.cnr.it

ABSTRACT
This work presents an approach to the design of the software component of an Interactive System, which supports the generation of automatic task-oriented help. Help can easily be generated from the abstract formal specification of the associated system without any further effort. The architectural description is obtained in a task-driven way, where tasks are specified by indicating temporal ordering constraints using operators of a concurrent formal notation. The association of user tasks with software interaction objects, which inherit constraints of related tasks, gives the information to structure task-oriented help in an immediate way. The help given is thus more expressive with a consequent improvement in the usability of an Interactive System.

KEYWORDS: Automatic help, task-driven design of architectures, development process for Interactive Systems software, formal notations.

INTRODUCTION

Automatic help generation is widely recognised as an important feature in order to provide usable environments. However, the design of help suffers from the same limitation as the design of user interfaces: a poor semantic support which implies a lack of semantic indications for users. In fact information which is related to perceivable features and low level actions is usually provided. Users may thus find difficult to associate this type of information with the tasks that they want to perform. This is because the design of most of current user interfaces and related toolkits focuses on appearance and layout, rather than on more important semantic design issues [4]. Task specification plays a key role in solving this kind of problem. It extends the traditional functional specification of a system so as to include the user's view of system functionalities and it is not constrained by aspects related to internal system components.

We propose an approach for obtaining automatic task-oriented help from the user interface specification. No great implementation effort is required, as we follow a structured and systematic methodology based on four concepts:

- An abstract model for software interaction objects (interactors) which captures their relevant features;
- A task-driven modelling of the architecture of Interactive Systems;
- The use of operators provided by a formal concurrent notation (LOTOS [3]) for describing temporal constraints among the possible actions;
- An object-oriented implementation language which provides good support for extendibility and reusability.

The use of operators derived from a formal notation is important as they guarantee a precise semantics which has been mathematically defined and this allows us to implement reliable algorithms to provide information related to tasks. We will use task specification, to produce both the design of the software user interface and the design of help. This approach has several advantages:

- We do not need to duplicate efforts in designing both aspects;
- The generation of help is not involved in detailed implementation issues but is automatically derived from the task specification;
- If tasks that the system should support are modified then the parts of the software architecture and help specification which have to be modified can easily be identified;
- Likewise, if temporal constraints among tasks are modified, the help system gives the consequent answer simply by updating the task specification without further effort.

The paper is organised as follows: we first describe related works, followed by an introduction to the general environment of our Interactive System Workbench. We then describe how support for a task-oriented help is given. Finally, by using a specific example, we show how help messages are structured.

RELATED WORKS

There are several works which use task specification to derive information for the design of Interactive Systems. Here we mention some of them. UAN [2] is a well-known notation which allows tasks to be specified by task decomposition and by indicating temporal constraints. In UAN basic tasks are described in terms of user action and system feedback while in our approach they are associated with software interaction objects (interactors) which support their performance. BOZ [1] is a tool where task specification is associated with perceptual operators to obtain more suitable graphical presentations. In [5] there is yet another approach, using task specification for driving dialogue specification, but it does not use any specific architectural model.

The first proposal for the attachment of help to points in a dialogue model was in the Syngraph system [8]. Then systematic work on automatic generation of help was in [15] which describes how it is obtained in UIDE. In this environment the logical structure of the user interface is described as a set of application objects which have pre and post conditions. The former indicate what is required to interact with one object, the latter indicate the effects after the interaction. This work was then extended in order to generate textual, audio and animated help.

In [9] there is an environment where the dialogue of the user interface is controlled by the implementation of Petri Nets. The user interactions modify the number and the location of tokens in the nets. By analysing the net it is possible to answer questions such as: What can I do now? Why is this interaction not available? How can I make that action available again?

Another approach is in [6] which describes hypertext-based help about data presented in application displays, commands to manipulate data, and interaction techniques to invoke commands. The authors propose a model-based design where the model is very abstract. In fact, it consists of the commands and objects of an application, the methods for presenting these commands and objects, and the behaviour of these objects in response to input events.

Our contribution to these useful works is the possibility to automatically generate help whose information is structured in terms of user tasks, thus making its contents more immediate to understand and use. This is a direct result of our task-driven methodology in modelling the software architecture of Interactive Systems. In fact, we noted that those who work on task specifications use their results for designing other aspects of their Interactive Systems, while those who work on help generation and design did not consider task-related structures and often explicitly mention this as one limitation of their work (see for example [6]). Besides we believe that the set of task relationships operator which we introduce can express richer relationships than the pre-condition/post-condition approach of some of the previous proposals.

THE INTERACTIVE SYSTEMS WORKBENCH

This work on the automatic generation of task-oriented help is part of a more general effort to produce an environment to support the various phases of the design and development of Interactive Systems. In our approach we first perform task decomposition. Abstract tasks are described in terms of basic tasks. This task decomposition is graphically represented in a tree-like organisation. The root of the tree is a very abstract task which gives an indication of the set of tasks that the system should support. Our tool allows designers to interactively edit the task specification.

Operators for temporal ordering are used to link subtasks at the same abstraction level. These operators are: *interleaving* (T1 ||| T2), where actions among two tasks can be performed in any order; *choice* (T1 [] T2), where it is possible to make a choice from a set of tasks, and once the choice has been made it has to be terminated and other tasks are not available; *synchronization* (T1 |[a1, ..., an]| T2), which indicates the actions (a1, ... , an) where two tasks have to synchronise; *deactivation* (T1 [> T2), where the first task is deactivated once the first action of the second task is performed; *enabling* (T1 >> T2), where one task enables a second one when it terminates. These operators are taken from notations developed for specifying concurrent systems (LOTOS). In the tasks specification we can have some tasks with an * next their name. This means that the tasks are performed recursively: when they terminate, their actions automatically start to be executed again from the beginning. This continues until the task is deactivated or an explicit exit action is reached.

This part is similar to the UAN approach. The main difference is that in our case basic tasks are associated with software interaction objects, while in UAN basic tasks are associated with the specification of possible user actions and system feedback.

Once we have obtained a satisfactory task specification with the graphical editor (see next section for an example), we can either automatically translate it into a LOTOS specification or derive the corresponding software architecture. Obtaining the LOTOS specification can be useful in order to perform further processing such as automatic simulation of the possible actions, verification of properties expressed in Action-based temporal logic, and so on.

Also, most of the transformation into software objects can be automatically performed. This translation is easy since to make a prototype we use a toolkit which we have developed following a semantics-based approach [11]. In fact, the toolkit was implemented incorporating a 3D design space for interaction objects. It has been implemented by a object-oriented language and the resulting hierarchy of the software interactors incorporates the design space in such a way that the first layers are driven by the most important aspects (semantic aspects). The first dimension defines its semantics, and how it modifies the state of the application side of a user interface; it can be associated with the basic task that it supports. The second dimension is how the presentation of the interaction object is performed, the third dimension concerns how feedback of the user-generated input is provided.

In this approach the resulting architecture of an Interactive System is a network of interaction objects sharing the same model (interactor model [10]) where single interaction objects can be mapped onto basic tasks, and the composition of interaction objects can be associated with abstract tasks. The software interaction objects inherit the same temporal constraints as the corresponding tasks. Software interactors can easily be reused by identifying the presence of the same task in different points of the task tree, and mapping all these occurrences with instances of the same class of software interactors.

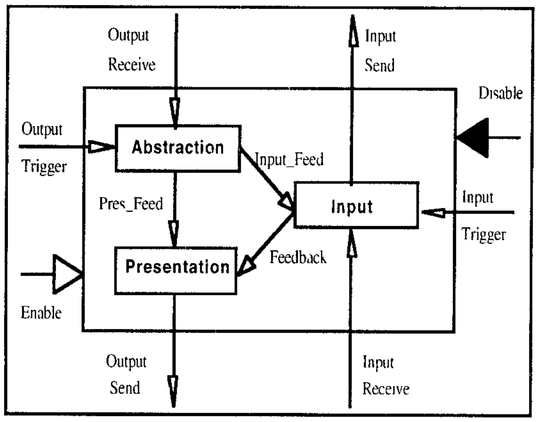


Figure 1: The architectural model of software interactors.

The internal architecture of the software interactors is represented in Figure 1. It is divided into three different components communicating among them. Each one is dedicated to control one of the dimensions mentioned before. The purpose of this structure is to maintain the internal components autonomous, with a well defined interface, in order to obtain a good modularity which allows developers to obtain slightly different interactions just replacing one internal component. For example, by replacing the Presentation component it is possible to modify the type of feedback produced by the interaction considered. Similarly, by replacing the Abastraction component it is possible to modify the definition of the presentation of the application data.

As we said, most of the translation from task specification into interactor-based software architecture is performed automatically. In order to have a complete working user interface the software architecture designer just has to define completely connections among interaction objects and with the Functional Core in order to determine the information flow.

This task-driven approach to the modelling of an Interactive System specification, using LOTOS operators to indicate temporal relationships among tasks, has been applied to several case studies, for example [13] reports its application to Matis, a multimodal user interface for a flights data base; and [14] describes the application to CERD, a user interface for air traffic controllers.

AN EXAMPLE

As an example of our approach let us now look at a user interface for an electronic message handling application. First we decompose the tasks by indicating the temporal ordering among tasks. Each task can be decomposed into two or more subtasks and operators link subtasks at the same abstraction level (see Figure 3).

The global task of message handling can be seen as message management until ([> operator) the session is closed. The message management is performed by checking for new messages and next (>> operator), when the previous task is terminated, by browsing the messages. Browsing can be performed by interleaving (||| operator) actions related to selecting and sending messages. Sending a message means composing it and then transmitting, while selecting means picking one and then handling it. Handling is a choice ([] operator) from replying, saving, deleting and deselecting the message. Finally, replying means composing the answer and then sending it.

SUPPORT FOR TASK-ORIENTED HELP

One advantage of a task-driven modelling of an Interactive System is that consequently also the run time architecture is able to provide information to obtain task-oriented help straightforward.

The fundamental information is the basic task-interactors association, where composition of interactors can be associated with more complex and abstract tasks. In fact, from this association we can know at any time which tasks can and cannot be executed. For those tasks that can be executed we can say by which sequence of actions they can be executed. For the tasks that cannot be executed we can outline the reasons for this and how they can be enabled.

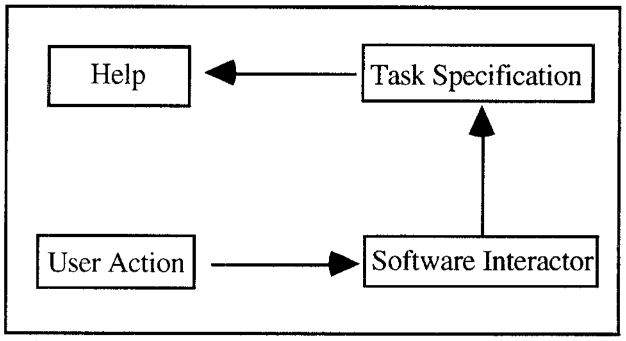


Figure 2: The flow of information for generating task-oriented help.

At run-time we have a Task Manager which receives dynamic information about activated and deactivated interactors from all interactors reacting to events which can be generated by the user, the application and other interactors. It also knows task-interactor associations and the task specification: how tasks are decomposed and their temporal constraints. The state (enabled or disabled and a short textual description) is also stored for each task.

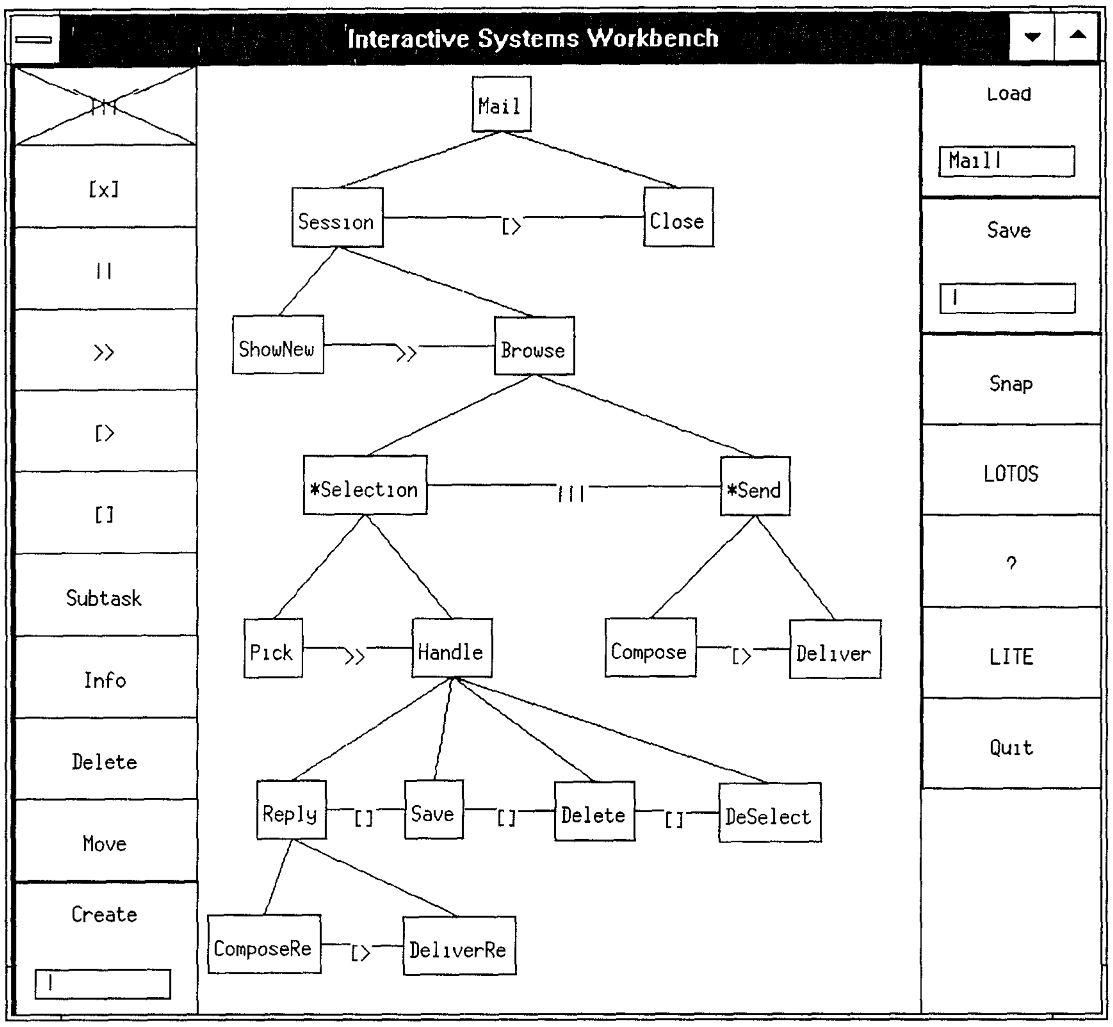


Figure 3: An example of task specification.

Specifically, we consider a basic task to be active when the corresponding interaction object is enabled; for non-basic tasks, we consider them active when at least one of their subtasks is active.

Thus, given a user interaction we can say what the current basic task is and what the current active tasks are. By navigating in the tree of tasks we can find out what the most abstract tasks are including the basic task considered. Thus the help can provide information at each of these abstraction levels.

We can thus answer a set of task-related questions:

- *Why is this task not allowed?*
- *How can I perform this task?*
- *How can I activate this task?*
- *What tasks can I perform now?*

In the case of the last question we can also ask for the actions associated with the performance of the next available tasks. Thus, if the user performs a disabled action, the system can give reasons for refusing to perform it in a task-oriented way saying: you have tried to perform task x which is not enabled because you first have to perform tasks y and z.

There are two ways to activate the help: either by trying to interact with a disabled interaction object or by explicit request.

The answer to the question "*What tasks can I perform now*" is easily found by examining the task graph and looking for active tasks. We then are able to provide information on the available tasks at the different levels of abstraction, down to the lowest level (the actual action the user has to do in order to perform a task).

This kind of information would be best expressed with a hypertext system, where the user could click on a non-basic task, and be given details on its subtasks. This is a future development, for now, the system can only provide a description of the basic tasks currently active, i.e. those which are mapped on actual interaction objects.

When the user asks *"How to perform a task"* the first thing the system does is to check if that task is already active. In this case, a short explanation on how to perform the task is generated. If the task is not active then the system starts to search for a way to enable it. All it needs to do is to go down the tree from the root node of the task tree (which, by definition, is always active, because it represents the whole application), toward the desired task, stepping only on active nodes. As we move down to the desired task, we find an active node N from which we cannot move further down, because the subtask N' of N which includes the desired task is disabled. At this point, we can explain why the task is disabled, by examining the LOTOS relationships among the subtasks of N. For example, suppose that N is decomposed into subtasks T1, T2, N', whose temporal relationship is expressed by the LOTOS behavioural expression T1>>N'>>T2. If we know that T1 is active and N', T2 are disabled, we can then inform the user that task N' is disabled because task T1 must be performed first.

If the user asks *"How to activate a task"*, again, we have to consider the task graph. As we move down to the desired task, we will find an active node N from which we cannot move further down, because the subtree S' of N which contains the desired task is disabled. At this point, there are two possibilities:

1) We can actually activate the subtask, for example by performing another subtask of N. In this case the system generates the required explanation automatically. Then the subtask N' is marked as active, we move to the node N', and the whole process is recursively repeated, until we reach the desired task.

2) We cannot move further down, because the subtask cannot be activated from the current state. In this case the system starts backtracking up to the root, looking for a recursive node *R (marked with an asterisk). This means that we find a task, which once terminated, can immediately be re-executed again in order to perform a different set of tasks which also include the desired task. A short explanation on how to complete task R is then printed. At this point, the system resets the state of the subtree with root *R, as if it had been just activated, and tries again to find a path to the requested subtask.

In order to explain better how this works we suppose (in the example in Figure 3) that while the user is composing an answer to a message, he/she decides to delete a message. While performing task ComposeRe the task is disabled, the system then starts backtracking in the tree of tasks in order to identify how to enable that task. First it finds Reply which is an alternative to Delete. Then it finds Handle, which once terminated, does not enable another choice. Then it finds Selection, which is marked with an * indicating that it is a recursive process. This means that once it is terminated it becomes available again. Thus, in order to delete a message we first have to pick one and then delete it.

Note that there may be no way to find an activation path: for example, in case 2, if no recursive task is found when backtracking. This means that in the current session the task cannot be performed. We consider these situations as design faults: a user interface should not allow for one-way trapdoors.

However, these flaws are easily spotted by generating the LOTOS specification. Then there are tools (e.g. Mauto [7]) which allow us to produce the finite state automaton corresponding to the LOTOS specification and checking for undesirable behaviours.

STRUCTURE OF HELP MESSAGES

From the search on the task graph, we obtain a list of tasks that must be performed in order to achieve the desired goal. Thus we have the information for answering the two typical questions about how to enable and perform a task. The first question has a dynamic answer because it depends on the current state of the session. While the second question has a fixed answer. There may be more than one answer for each question as there may be several ways to perform a task.

To explain how our system works, let us consider again our simple 'mail' application. Suppose that, after checking for new mail, we want to save a message. The output of the graph search is something like:

```
To save a message
1. Perform task Selection
2.        Perform task Pick
3.        Perform task Handle
4.                Perform task Save
```

where the indentation represents the level of subtask nesting.

Now the problem is to present this information to the user in an easy-to-understand way. Experience has shown that a good way to solve this problem is to use pieces of pre-written text which are joined together by the help engine to form a sensible explanation. We choose to consider as basic units of information the description of basic tasks - i.e. tasks that are not decomposed into subtasks but are mapped directly onto interaction objects.

In the answer we put in bold the part which is fixed and predefined for all specific answers. We use the **or** connection when multiple answers are possible. Note that "handle" and "selection" tasks produce no text, as they are not basic tasks.

Going back to our example, the resulting output to the query would be:

How to activate the task: *Saving a message*?
select a message by clicking on the message title, shown in the "pick" interactor.

How to perform the task: *Saving a message*?
type the name of the file you want to save the article in, using the "save" interactor.

Why is the task *Saving a message* **disabled**?
Because first you have to complete the task: *Select message*

What tasks can I do now?
You can perform the tasks:
Pick an existing message
or
Send a new message

To make things a bit more interesting, suppose now that the user, while replying to a message, suddenly needs to save a message. The 'save' and 'pick' tasks are disabled, and so are their interaction objects. The attempted use of a disabled interactor triggers the help engine, which displays a window explaining how to perform the task. The output would be:

How to activate the task: *Saving a message*?
Deliver the message being edited by clicking on the "deliver" button.
and
Select a message by clicking on the message title, shown in the "pick" interactor.

How to perform the task: *Saving a message*?
Type the name of the file you want to save the message in, using the "save" interactor.

Why is the task *Saving a message* **disabled**?
Because first you have to complete the task: *Replying to selected message*

What tasks can I do now?
You can perform the tasks:

Compose the reply to the selected message
and
Deliver the reply to the selected message

or

Compose a new message
and
Deliver the new message

In the answer we put the **and** connection when multiple tasks in a sequential order have to be performed. Here, the system tried to work its way down the task tree, but on the node associated with the "handle" task it found there was no way to activate the "save" task, because the "reply" branch had already been selected. So it printed information on how to complete the "reply" task, and then backtracked up to the "selection" node. Figure 4 shows how the last example was performed by the running system.

In this case, the resulting explanation was quite readable - and useful. Sometimes it might not be so readable, especially when multiple backtrack steps are required. But this is a problem of the underlying interface design rather than the help system itself: if it takes too many steps to get from a to b, then probably it would be wise to reconsider the task decomposition. However, the average number of logical steps (that is, basic tasks) that are needed to perform a task starting from a given state can be considered as a parameter for the evaluation of the usability of a software system.

CONCLUSIONS

We have presented an approach which allows us to obtain highly structured task-oriented help directly from the system specification. This means providing users with more organized and comprehensible information in order to understand the Interactive System considered. This highlights that to obtain more usable environments, design should focus more on semantic aspects of user interactions rather than presentation aspects alone. This goal can be obtained by means of a task-driven approach in modelling the software architecture of the system.

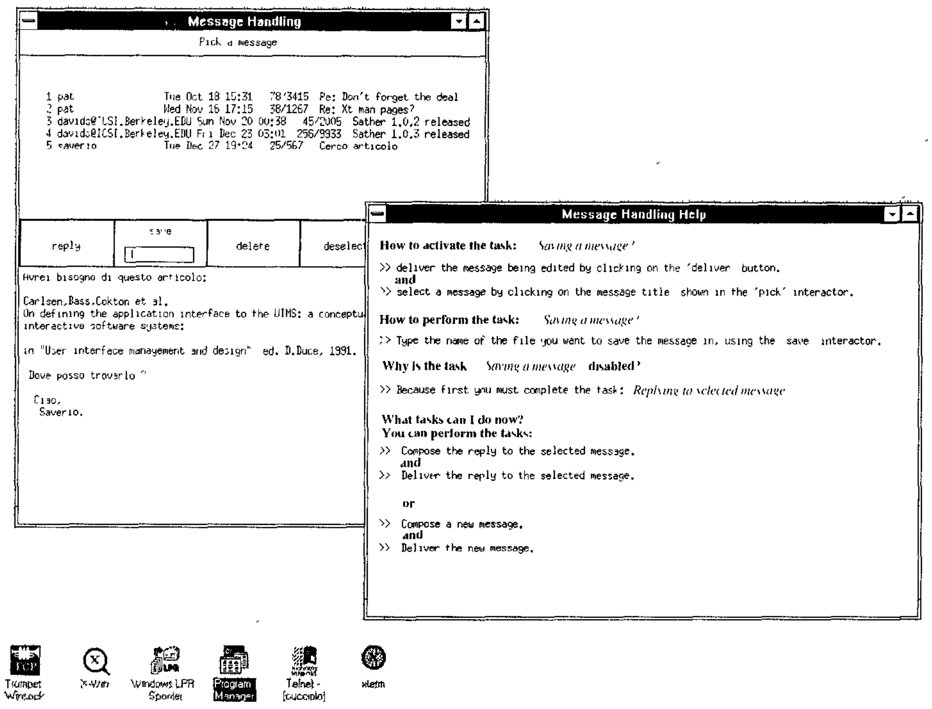
As a future development we plan to design and implement a hypertext organisation for the help system to provide information so as to allow users to navigate in the task hierarchy.

REFERENCES

1 S.M.Casner, "A Task-Analytic Approach to the Automated Design of Graphics Presentations", ACM Transactions on Graphics, Vol.10, N.2, April 1991, pp.111-151.

2 H.R.Hartson, A.C.Siochi, D.Hix. "The UAN: A User-Oriented Representation for Direct Manipulation Interface Designs". ACM Transactions on Information Systems, Vol.8, N.3, pp.181-203.

3 ISO (1988) Information Processing Systems - Open Systems Interconnection - LOTOS - A Formal Description Technique Based on temporal Ordering of Observational Behaviour. ISO/IS 8807, ISO Central Secretariat.



```
1 pat                    Tue Oct 18 15:31   78/3415  Re: Don't forget the deal
2 pat                    Wed Nov 16 17:15   38/1267  Re: Xt man pages?
3 davids@ICSI.Berkeley.EDU Sun Nov 20 00:38   45/2005  Sather 1.0.2 released
4 davids@ICSI.Berkeley.EDU Fri Dec 23 03:01  256/9933  Sather 1.0.3 released
5 saverio                Tue Dec 27 19:24   25/567   Cerco articolo
```

```
Avrei bisogno di questo articolo:

Carlsen,Bass,Cokton et al,
On defining the application interface to the UIMS: a conceptu
interactive software systems:

in "User interface management and design"  ed. D.Duce, 1991.

 Dove posso trovarlo ?

  Ciao,
   Saverio.
```

Message Handling Help

How to activate the task: *Saving a message'*

>> deliver the message being edited by clicking on the 'deliver button.
and
>> select a message by clicking on the message title shown in the 'pick' interactor.

How to perform the task: *Saving a message'*

>> Type the name of the file you want to save the message in, using the save interactor.

Why is the task *Saving a message* **disabled'**

>> Because first you must complete the task: *Replying to selected message*

What tasks can I do now?
You can perform the tasks:

>> Compose the reply to the selected message.
and
>> Deliver the reply to the selected message.

or

>> Compose a new message.
and
>> Deliver the new message.

Figure 4: An example of automatically generated task-oriented help.

4 J.Johnson, "Selectors: Going Beyond User-Interface Widgets", Proceedings ACM CHI'92, pp.273-279.

5 A.Monk, M.Curry, "Discount Dialogue Modelling with Action Simulator". In Cockton, G., Draper, S.W. and Weir, G.R.S. (Eds), Computers and People 9: HCI'94 Proceedings, Cambridge: Cambridge University Press, pp.327-338.

6 R.Moriyon, P.Szekely, R.Neches, "Automatic Generation of Help from Interface Design Models", Proceedings ACM CHI'94, pp.225-231.

7 E. Madelaine, D. Vergamini. "AUTO: a verification tool for distributed systems using reduction of finite automata networks". Proc. FORTE'89 Conference, pp. 61-66, North Holland, Amsterdam, 1990.

8 D.R.Olsen, E.P.Dempsey, "SYNGRAPH: A Graphical User Interface Generator", Proceedings SIGGRAPH'83 ACM Computer Graphics, Vol.17, N.3, pp43-50, July 1983..

9 P.Palanque, R.Bastide, L.Dourte, "Contextual help for free with formal dialogue design", Proceedings HCI'93 International, pp.615-620.

10 F.Paterno', "A Theory of User-Interaction Objects", Journal of Visual Languages and Computing, Vol.5, N.3, pp.227-249.

11 F.Paterno', A.Leonardi. "A Semantics-based Approach to the Design and Implementation of Interaction Objects", Computer Graphics Forum, Blackwell Publisher, Vol.13, N.3, pp.195-204.

12 F.Paterno', A.Leonardi, S.Pangoli, "A Tool Supported Approach to the Refinement of Interactive Systems", in Interactive Systems: Design, Specification, and Verification, pp. 149-160, Springer Verlag, Focus on Computer Graphics Series, ISBN 3-540-59480-9.

13 F.Paterno', M.Mezzanotte, "Analysing Matis through Interactors" and ACTL, Amodeus II BRA Report, sm/wp36.

14 F.Paterno', M.Mezzanotte, "Formal Analysis of User and System Interactions in the CERD Case Study", EHCI'95 IFIP Conference Proceedings, Grand Targhee Resort, August 1995, Chapman & Hall.

15 P.Sukaviriya, J.Foley, "Coupling a UI Framework with Automatic Geneation of Context-Sensitive Animated Help", Proceedings UIST'90, pp.152-165.

16 P.Sukaviriya, J.Muthukumarasamy, A.Spaans, H. de Graaff, "Automatic Generation of Textual, Audio, and Animated Help in UIDE: The User Interface Design Environment", Proceedings AVI'94, pp.44-52, ACM Press.