Towards a UML for Interactive Systems

Fabio Paternò

CNUCE-C.N.R., Via V.Alfieri 1, 56010 Ghezzano, Pisa, Italy
fabio.paterno@cnuce.cnr.it

Abstract. Nowadays, UML is the most successful model-based approach to supporting software development. However, during the evolution of UML little attention has been paid to supporting user interface design and development. In the meantime, the user interface has become a crucial part of most software projects, and the use of models to capture requirements and express solutions for its design, a true necessity. Within the community of researchers investigating model-based approaches for interactive applications, particular attention has been paid to task models. ConcurTaskTrees is one of the most widely used notations for task modelling. This paper discusses a solution for obtaining a UML for interactive systems based on the integration of the two approaches and why this is a desirable goal.

1 Introduction

The success of UML [9] reflects the success of object-oriented approaches in software development. UML is already composed of nine notations. In practice, software engineers rarely use all of them. In this context, it seems difficult to propose the inclusion of another notation. However, successful approaches have to take into account what the new important problems are, if they want to continue to be successful. The rapidly increasing importance of the user interface component of a software system cannot be denied. Since the early work on UIDE [3], various approaches have been developed in the field of model-based design of interactive applications [7] [8] and they have not affected at all the development of UML. The developers of UML did not seem particularly aware of the importance of model-based approaches to user interfaces: UML is biased toward design and development of the internal part of software systems, and has proven its effectiveness in this. However, despite UML use cases and other notations can effectively capture "functional" requirements or specify detailed behaviours, UML does not specifically - nor adequately - support the modelling of user interfaces aspects.

In the European R&D project GUITARE, we developed a set of tools for task modelling and have proposed their use to software companies. The first issue raised in response was how to integrate their use with their UML-guided design and development practices. The issue of how to integrate model-based approaches to user interfaces with UML was discussed in some CHI workshops where participants agreed that it is conceptually possible to link development practice to HCI practice through the use of object models derived from task analysis and modelling [1].

M. Reed Little and L. Nigay (Eds.): EHCI 2001, LNCS 2254, pp. 7–18, 2001.

Last generation of model-based approaches to user interface design agree on the importance of task models. Such models describe the activities that should be performed in order to reach users' goals. Task models have shown to be useful in a number of phases of the development of interactive applications: requirements analysis, design of the user interface, usability evaluation, documentation and others. Tasks are important in designing interactive applications: when users approach a system, they first have in mind a set of activities to perform, and they want to understand as easily as possible how the available system supports such activities and how they have to manipulate the interaction and application objects in order to reach their goals. Likewise, when modelling human-computer interaction it is more immediate to start with identifying tasks and then the related objects. Conversely, when people design systems (with no attention to the human users) they often first think in terms of identifying objects and then try to understand how such objects interact with each other to support the requested functionality.

ConcurTaskTrees (CTT) [7] is a notation that has been developed taking into account previous experience in task modelling and adding new features in order to obtain an easy-to-use and powerful notation. It is a graphical notation where tasks are hierarchically structured and a rich set of operators describing the temporal relationships among tasks has been defined (also their formal semantics has been given). In addition, it allows designers to indicate a wide set of task attributes (some of them are optional), such as the category (how the task performance is allocated), the type, the objects manipulated, frequency, and time requested for performance. Each task can be associated with a goal, which is the result of its performance. Multiple tasks can support the same goal in different modalities. The notation is supported by CTTE (the ConcurTaskTrees Environment), a set of tools supporting editing and analysis of task models. It has been used in many countries in a number of university courses for teaching purposes and for research and development projects. It includes a simulator (also for cooperative models) and a number of features allowing designers to dynamically adjust and focus the view, particularly useful when analysing large specifications.

In the paper there is first a discussion of possible approaches to integrating task models and UML. Next, the approach selected is presented along with criteria to indicate how to use the information contained in use case diagrams to develop task models. For this part, examples taken from a case study concerning applications supporting help desks are given. Finally, some concluding remarks and indications for further work are presented.

2 Integrating UML and Task Models

Aiming at integrating the two approaches (task models represented in CTT and UML) there can be various basic philosophies that are outlined below. Such approaches can exploit, to different extents, the extensibility mechanisms built into UML itself (constraints, stereotypes and tagged values [9]) that enable extending UML without having to change the basic UML metamodel. The types of approaches are:

- *Representing elements and operators of a CTT model by an existing UML notation*. For example, considering a CTT model as a forest of task trees, where CTT

operands are nodes and operators are horizontal directed arcs between sibling nodes, this can be represented as UML class diagrams. Specific UML class and association stereotypes, tagged values and constraints can be defined to factor out and represent properties and constraints of CTT elements [6];

- *Developing automatic converters from UML to task models*, for example in [5] there is a proposal to use the information contained in system-behaviour models supported by UML (use cases, use cases diagrams, interaction diagrams) to develop task models.
- *Building a new UML for interactive systems*, which can be obtained by explicitly inserting CTT in the set of available notations, still creating semantic mapping of CTT concepts into UML metamodel. This encompasses identifying correspondences, both at the conceptual and structure levels, between CCT elements and concepts and UML ones, and exploiting UML extensibility mechanisms to support this solution;

Of course there are advantages and disadvantages in each approach. In the first case, it would be possible to have a solution compliant with a standard that is already the result of many long discussions involving many people. This solution is surely feasible. An example is given in [6]: CTT diagrams are represented as stereotyped class diagrams; furthermore constraints associated with UML class and association stereotypes can be defined so to enforce the structural correctness of CTT models. However, I argue that the key issue is not only a matter of expressive power of notations but it is also a matter of representations that should be effective and support designers in their work rather than complicate it. The usability aspect is not only important for the final application, but also for the representations used in the design process. For example, UML state charts and their derivative, activity diagrams, are general and provide good expressive power to describe activities. However, they tend to provide lower level descriptions than those in task models and they require rather complicated expressions to represent task models describing flexible behaviours. In [10] there is another example of using existing UML notations for describing concepts important for designing interactive systems such as task and presentation models. Again, this proves its feasibility but the readability and comprehensibility of the resulting diagrams are still far from what user interface designers expect. Thorny problems emerge also from the second approach. It is difficult to model first a system in terms of objects behaviour and then derive from such models a meaningful task model because usually object-oriented approaches are effective to model internal system aspects but less adequate to capture users' activities and their interactions with the system. The third approach seems to be more promising in capturing the requirements for an environment supporting the design of interactive systems. However, attention should be paid so that software engineers who are familiar with traditional UML can make the transition easily. A possible variant to this solution is to use CTT as a macro-notation for an existing UML notation, such as activity diagrams. This would offer the advantages of being more comprehensible and better able to highlight the elements of interest for user interface designers, while still allowing the constructs to be mapped onto a standard UML notation. In particular, in the following I will outline a proposal aimed at obtaining a UML for interactive systems based on the integration of CTT and current UML.

3 The Approach Proposed

In our approach to UML for Interactive Systems the use of task models represented in CTT is explicitly introduced. The approach should also be supported by a defined process and a related tool for integrated use of CTT and the other notations useful for supporting design in this class of software applications.

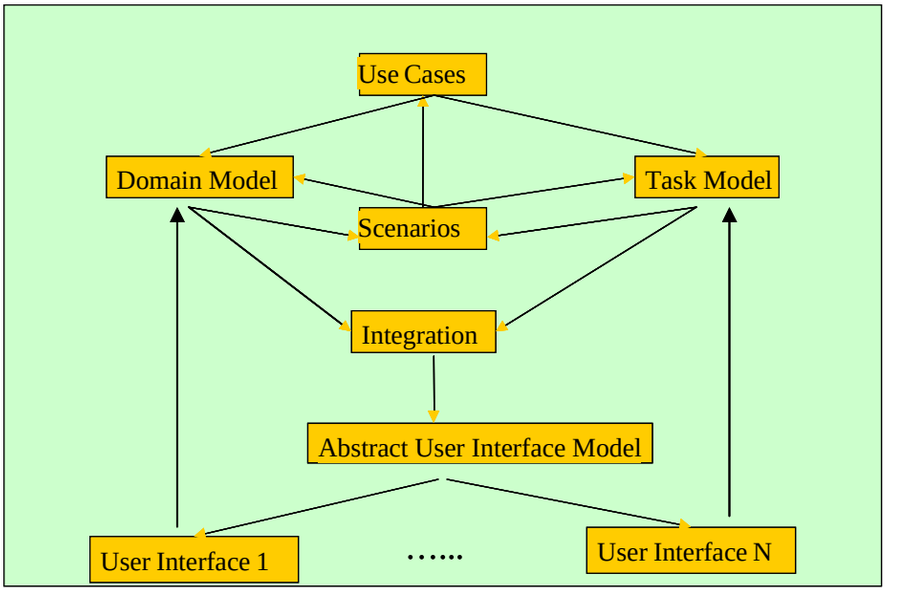

Fig. 1. The Representations Involved in the Approach Proposed

Not all UML notations are equally relevant to the design of interactive systems; the most important in this respect appear to be use case, class diagrams and sequence diagrams. In the initial part of the design process (see Figure 1), during the requirement elicitation phase, use cases supported by related diagrams should be used. Use cases are defined as coherent units of externally visible functionality provided by a system unit. Their purpose is to define a piece of coherent behaviour without revealing the internal structure of the system. They have shown to be successful in industrial practise.

Next, there is the task modelling phase that allows designers to obtain an integrated view of functional and interactional aspects. Interactional aspects, related to the ways to access system functionality, in particular cannot be captured well in use cases; so, in order to overcome this limitation they can be enriched with scenarios, informal descriptions of specific uses of the system considered. More user-related aspects can emerge during task modelling. In this phase, tasks should be refined, along with their temporal relationships and attributes. The support of graphically represented hierarchical structures, enriched by a powerful set of temporal operators, is particularly important. It reflects the logical approach of most designers, allows

describing a rich set of possibilities, is declarative, and generates compact descriptions.

The decomposition of the task model can be carried on until basic tasks (tasks that involve no problem solving or control structure component) are reached. If this level of refinement is reached it means that the task model includes also the dialogue model, describing how user interaction with the system can be performed. If such a level of description is reached then it is possible to directly implement the task model in the final system to control the dynamic enabling and disabling of the interaction objects and to support features, such as context-dependent task-oriented help.

In parallel with the task modelling work, the domain modelling is also refined. The goal is to achieve a complete identification of the objects belonging to the domain considered and the relationships among them. At some point there is a need for integrating the information between the two models. Designers need to associate tasks with objects in order to indicate what objects should be manipulated to perform each task. This information can be directly introduced in the task model. In CTT it is possible to specify the references between tasks and objects. For each task, it is possible to indicate the related objects, including their classes and identifiers. However, in the domain model more elaborate relationships among the objects are identified (such as association, dependency, flow, generalisation, …) and they can be easily supported by UML class diagrams. There are two general kinds of objects that should be considered: the presentation objects, those composing the user interface, and application objects, derived from the domain analysis and responsible for the representation of persistent information, typically within a database or repository. These two kinds of objects interact with each other. Presentation objects are responsible for creating, modifying and rendering application objects. Given one application object there can be multiple presentation objects (for example, a temperature can be rendered by a bar chart or a numeric value). The refinement of task and objects can be performed in parallel, so that first the more abstract tasks and objects are identified and then we move on the corresponding more concrete tasks and objects. At some point the task and domain models should be integrated in order to clearly specify the tasks that access each object and, vice versa, the objects manipulated by each task.

Thus, we follow a kind of middle-out, model-first [11] approach, by which we identify elements of interest through the Use Cases and then use them to create more systematic models that allow clarifying a number of design issues and then to develop the resulting interactive software system.

Scenarios are a well known technique in the HCI field often used to improve understanding of an interactive applications. They provide informal descriptions of a specific use in a specific context of an application. CTTE, the tool supporting CTT, gives also the possibility of simulating the task model. This means that it is possible to interactively select one task and ask the system to show what tasks are enabled after the performance of that task, and then carry on this activity iteratively. A sequence of interactions with the simulator allows designers to identify an abstract scenario, a specific sequence of basic tasks that are associated with a specific use of the system considered. Figure 2 provides an example describing a scenario derived from a task model for a cooperative ERP application (in the *Scenario to be performed* sub-window). The scenario was created beforehand and now is loaded for interactive analysis within the task model simulator

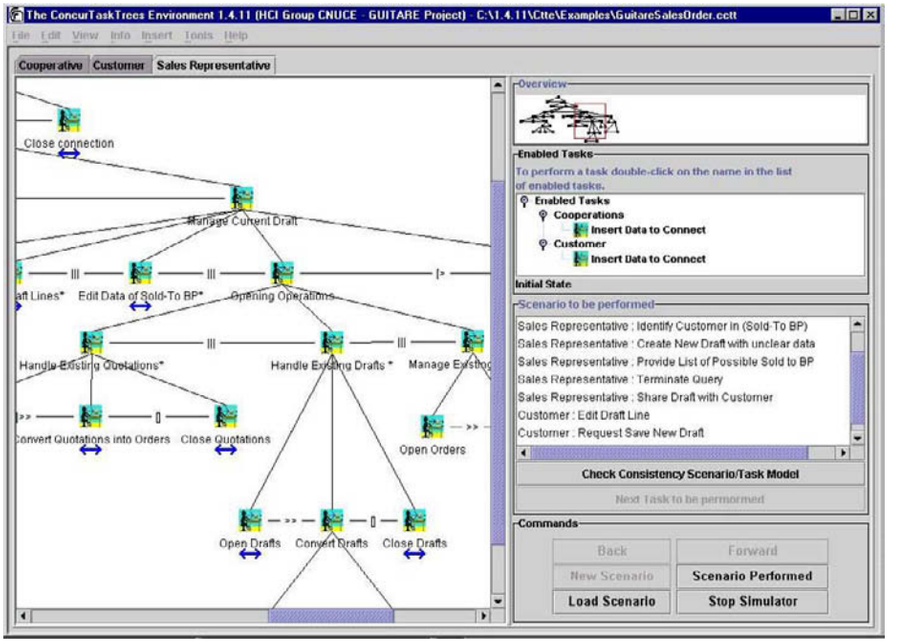

Fig. 2. Example of Abstract Scenario Derived from Task Model

A similar concept is supported by sequence diagrams in UML: two-dimensional charts where the vertical dimension is time, while the horizontal dimension shows the classifier roles representing individual objects in the collaboration. Such diagrams provide good representation of limited sequential activities. Their specificity is that they represent the messages sent with arrows, indicating the objects involved and the direction of data flow. In both cases, we can think about them as representations of scenarios that are used to validate the dynamic behaviour of both the domain and task models. In addition, informal scenarios can be used as a technique for preliminary identification of requirements, and thus can give useful input for the development of use cases that provide more structured descriptions.

Task models are useful to support the design of user interfaces. Deriving the concrete user interface can be looked at as a two-step process. First, an abstract user interface model should be identified: it gives a platform-independent description of how the user interface should be structured. Next, the concrete user interface is generated taking into account the specific platform considered (desktop, laptop, handheld device, …). Note that this framework allows designers to support plasticity [12], the possibility of adapting to a specific device (although this is beyond the scope of this paper).

A number of criteria have been identified [7] to gather information from the task model that is useful to obtain effective user interfaces. The concept of enabled task sets is used to identify the presentations of the user interface. An enabled task set is a group of tasks enabled over the same period of time. Thus it is a good candidate to be supported by a single presentation (however, multiple enabled tasks sets that differ by

only a few elements may be supported by a single presentation). Once we have identified the presentations, they can first be structured and then refined according to a number of criteria, for example:

- grouping of tasks can be reflected in the grouping of presentation objects;
- temporal relationships among tasks can be useful to structure the user interface dialogue and indicate when the interaction techniques supporting the tasks are enabled;
- the type of tasks and objects (and their cardinality) should be considered when the corresponding widgets and presentation techniques are selected.

The design process is iterative. So, when a user interface is drafted, then it is still possible to return to the task and domain models, modify them and start the cycle over again.

To summarise, the rationale for the approach proposed is that in current UML notations it is possible to describe activities by activity diagrams or state chart diagrams (which are specialised types of activity diagrams). However, these notations have mainly been conceived of to describe either high-level user work-flow activities, or low-level interactions among software objects. Although they can be used to describe task models or user interactions and some research work has been done in this direction, the results do not seem particularly convincing. Activity diagrams do not seem to provide good support for multiple abstraction levels, which has been found particularly useful in HCI (all successful approaches in task modelling, including HTA, GOMS, UAN share this feature). In addition, they do not seem to scale-up well: when medium-large examples are considered, the graphical representations obtained are often rather difficult to interpret. It is possible to define such diagrams hierarchically, but designers often have difficulties in following their relationships and obtaining an overall view of the specification.

4 Moving from Use Cases to Task Models

Use cases have been well accepted in the phase of requirement elicitation and specification. There is some confusion over exactly how to use and structure them (see for example [2] for a discussion of the many ways to interpret what a use case is). Here we will refer to UML use cases that are usually structured informal descriptions accompanied by use case diagrams. Thus, it becomes important to identify a set of criteria helping designers to use the information contained in use cases diagrams as a starting point to develop task models.

Use cases can give useful information for developing the CTT model in a number of ways. They identify the actors involved. Such actors can be mapped onto the roles of the CTT models. Indeed, CTT allows designers to specify the task model of cooperative applications by associating a task model for each role involved, and then a cooperative part describes how tasks performed by one user depend on the tasks performed by other users. In addition, a use case allows the identification of the main tasks that should be performed by users, the system or their interaction. This distinction in task allocation is explicitly represented in CTT by using different icons. Also abstract objects can be identified from an analysis of use cases: during use case

elicitation, a number of domain objects are also identified, and collected in term of UML classes and relationships; this enables the incremental construction of the core of the domain model which is a principal result of analysis activities.

In use case diagrams there are actors, use cases and relationships among use cases. Actors can be associated with the roles of the task model and use cases can be considered as high-level tasks. In the use case diagram it is indicated to what use cases actors are involved. This means that they will appear in the corresponding task model. In addition, use cases that involve multiple actors must be included in the cooperative part of the task model. The relationships among use cases are: association (between an actor and a use case), extension (to insert additional behaviour into a base use case that does not know it), generalisation (between a general use case and a more specific one), inclusion (insertion of additional behaviour into a base use case that explicitly describes the insertion).

It is possible to identify criteria to indicate how to consider these operators in the task model. For example, the inclusion relationship can be easily mapped onto the decomposition relation supported by hierarchical tasks, while the generalisation relationship requires both task decomposition and a choice operator between the more specific alternatives. The extension relationship can be expressed as either an optional task or a disabling task performed when some particular condition occurs. These criteria should be interpreted as suggestions rather than formal rules, still giving the possibility of tailoring them to the specific case considered. In Figure 3 there is an example of Use case concerning a case study described in [4]. It is an abstract use case (HD Call) whose goal is to identify and enable access to help desk (HD) services to registered users. It is easy to see that a number of roles are involved: the help desk user, the help desk operator, the expert and the support manager. The relationships used in this diagram are: associations between actors and use cases (for example the File HDU Feedback is associated with the HD User actor); extend (for example, Abort HD Call extend HD Call); include (for example HD Call includes Start HD Call, it is possible also to indicate one-to-many relationships, such as in the case of HD Call and HD Interactions).

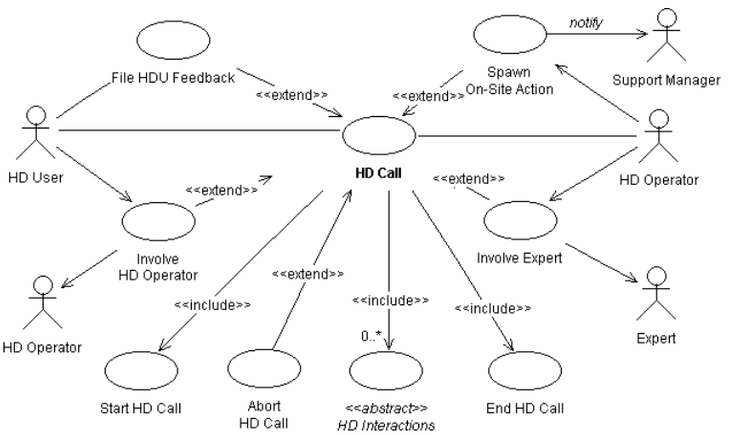

Fig. 3. Example of multi-actor Use case Diagram

The use cases included in the above diagram can be further expanded through specific diagram associated with each of them. It is possible then to group all the use cases that involve a certain actor. An example is provided in Figure 4, where the help desk user is considered. We can also find an example of use case generalisation (Identify HD Issue is a generalisation of Init New HDU Issue and Resume HDU Issue). This type of representation (a use case diagram describing all the use cases involving a certain role) can be used as starting point for the development of the corresponding task model.

Figure 5 shows the task model corresponding to our example. A part from the root task (Access HD), this task model just associates a task with each use case. However, the task model already adds further information to the use case diagram because the temporal relationships among the activities are indicated. The Abort HD Call task can interrupt the HD Call task ([> is the disabling operator) at any time. The HD Call is decomposed into a number of tasks: we first have the Start HD Call task, which is further decomposed into Identify HD User followed by Identify HD Issue (>> is sequential operator), and then a set of concurrent cooperating tasks (|[]| is the concurrent with information exchange operator). Then it is possible to perform either Suspend or Close the HDU Issue ([] is the choice operator) and lastly, after the optional performance of File HDU Feedback (optional tasks have the name in squared brackets), the HD Call ends. The model in Figure 5 concerns high-level tasks and represents a good starting point for the design of the user interface.

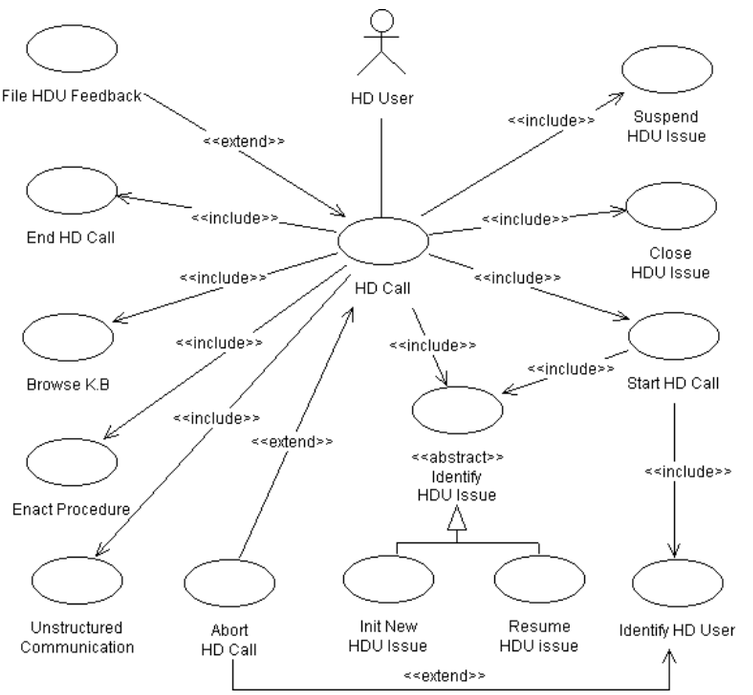

Fig. 4. Example of Use case Diagram associated with an Actor

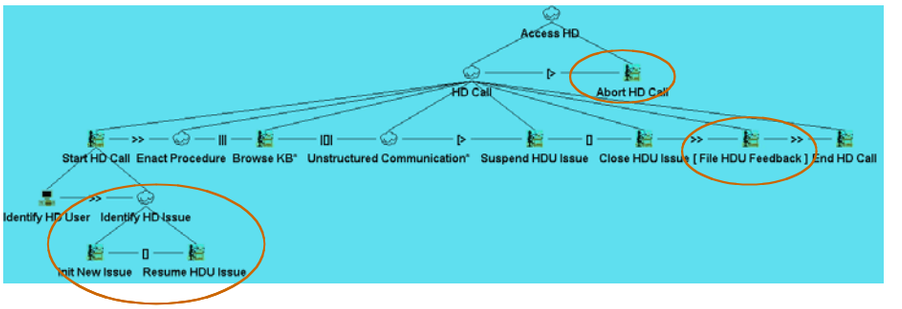

Fig. 5. Example of Task Model corresponding to the previous Use Case Diagram

Figure 6 provides a more readable representation of the task model by introducing high-level tasks grouping strictly related sub-tasks that share some temporal constraints (HD Call Session, HD Interactions and Dispose HDU Issue). Thus, we obtain a more immediate representation of the various phases of a HD Call.

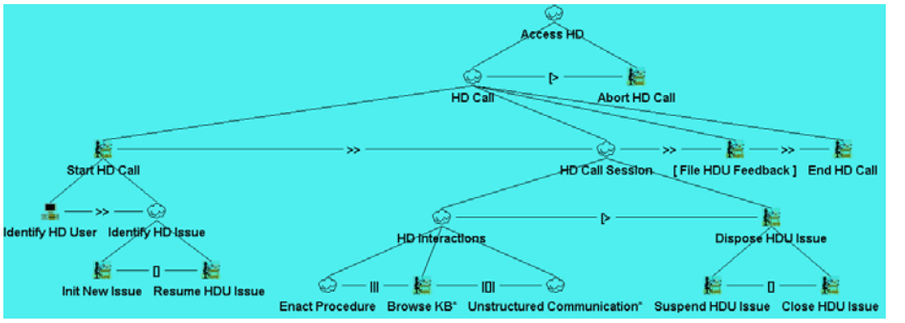

Fig. 6. Structured Representation of the Example of Task Model

It is clear that the model obtained at this point is not yet the complete task model. Some tasks need further refinements and should be structured more precisely (for example, by indicating further sub-tasks). In addition, during the final refinement more attention should be paid to designing how activities are supported with the support of the user interface. On the other hand, we have seen how it is possible to obtain a preliminary structure of the task model satisfying the requirements indicated in the use cases.

5 Conclusions and Discussion

This paper has discussed an approach to integrating UML and CTT for designing interactive systems based on the use of use cases, Class Diagrams, CTT task models and scenarios. A successful modelling environment for interactive systems must account for the importance of task models and the notations that have proved to be effective in representing them. The logical framework that has been presented gives

indications of the types of representations to use in the various phases and their relationships. It supplies clear suggestions on how to leverage recent research work on model-based design of user interfaces to improve UML.
Automatic support for the new method introduced in this paper seems to be a reasonable goal. To this end, an editor of uses cases has been added to the CTTE tool. At this time, this is the most engineered tool for task modelling and analysis and it is publicly available (http://giove.cnuce.cnr.it/ctte.html).

Acknowledgments

I wish to thank Nando Gallo, Vincenzo Benvenuto, Giovanni Radice (Intecs Sistemi S.p.A.) for useful discussions on the topics of the paper. GUITARE is a R&D Project funded by the European Commission. Support is gratefully acknowledged. More information is available at http://giove.cnuce.cnr.it/guitare.html

References

1. Artim, J., et al, Incorporating Work, Process and Task Analysis Into Commercial and Industrial Object-Oriented System Development, SIGCHI Bulletin, 30(4), 1998.
2. Cockburn A., Structuring Use cases with Goals, Journal of Object-Oriented Programming, http://members.aol.com/acockburn, 1997.
3. Foley, J., Sukaviriya, N., History, Results, and Bibliography of the User Interface Design Environment (UIDE), an Early Model-based System for User Interface Design and Development, in F. Paterno' (ed.) Interactive Systems: Design, Specification, Verification, pp. 3-14 Springer Verlag, 1994.
4. F.Gallo, A.Paladino, E.Benvenuto, Application Domain Analysis and Requirements for Pilot Applications, GUITARE Deliverable 3.1, 2000.
5. S.Lu, C.Paris, K.Linden, Toward the Automatic Construction of Task Models from Object-Oriented Diagrams, Proceedings EHCI'98, Kluwer Academic Publishers, pp.169-189
6. N.Numes, J.Falcao, Towards a UML profile for user interface development: the Wisdom approach, Proceedings UML'2000, LNCS, Springer Verlag.
7. Paternò, F., Model-Based Design and Evaluation of Interactive Application. http://giove.cnuce.cnr.it/~fabio/mbde.html Springer Verlag, ISBN 1-85233-155-0, 1999.
8. Puerta, A., A Model-Based Interface Development Environment, IEEE Software, pp. 40-47, July/August 1997.
9. J.Rumbaugh, I.Jacobson, G.Booch, The Unified Modeling Language Reference Manual, Addison-Wesley, 1999.
10. P. da Silva, N.Paton, UMLi : The Unified Modeling Language for Interactive Applications, Proceedings UML'2000, LNCS, Springer Verlag
11. M.B. Rosson, Designing Object-Oriented User Interfaces from Usage Scenarios, ACM CHI'97 workshop on Object Models in User Interface Design, http://www.cutsys.com/CHI97/Rosson.html
12. Thevenin D., Coutaz J., Plasticiy of User Interfaces: Framework and Research Agenda, Proceedings INTERACT'99, pp. 110-117, IOS Press, 1999.

Discussion

P. Smith: Why do you say to software developers that believe that use cases + prototypes are sufficient and that task modeling is unnecessary?
F. Paterno: These people are making a lot of design decisions that are not captured well in use cases and prototypes alone.

N. Graham: I didn't understand how you represent temporal relations in UML
F. Paterno: You could use sequence diagram or activity diagrams for instance but we prefer to represent that using the CTT notation.