Mapping ConcurTaskTrees into UML 2.0

Leonel Nóbrega[1], Nuno Jardim Nunes[1] and Helder Coelho[2]

[1] Department of Mathematics and Engineering, University of Madeira,
Campus da Penteada, 9000-390 Funchal, Portugal
{lnobrega,njn}@uma.pt
[2] Department of Informatics of the Faculty of Sciences,
University of Lisbon, Bloco C6, Piso 2, 1749-016 Lisboa, Portugal
hcoelho@di.fc.ul.pt

Abstract. The ConcurTaskTrees (CTT) is one of the most widely used notations for task modeling, specifically tailored for user interface model-based design. The integration of CTT with a de facto standard modeling language was already identified as an import issue, but there is no consensus about the best approach to achieve this goal. The purpose of this paper is to examine the relative strengths and weaknesses of control and data flow specification in UML 2.0 Activity Diagrams to represent CTT semantics. The analysis is conducted by the definition of pattern-based activities for the temporal operators in CTT. Here, we propose an extension to the UML 2.0 abstract syntax that fully supports the concepts behind CTTs and provides an adapted graphical notation for an UML like representation.

1 Introduction

Task modeling is a central and familiar concept in human-computer interaction (HCI) but seldom-used in object-oriented software engineering (OOSE). A task model details users' goals and the strategies adopted to achieve those goals, in terms of actions that users perform, the objects involved in those actions and the underlying sequencing of activities [1]. Task models captures the dialog model of interactions and is crucial for enabling model-base approaches for building interactive systems. The UML insufficiencies for interaction design are widely recognized [2,3] and the integration of task model are a step further its limitations. The ConcurTaskTrees is one of the most widely used notations for task modeling, specifically tailored for user interface model-based design and its integration with UML is already identified as a desirable goal.

Integrating CTT in the UML can be generally achieved with the following approaches:

- Using the UML extension mechanisms (profiles), to represent elements and operators of a CTT model by an existing UML notation,
- Extending the UML metamodel, introducing a separate user task model, and establishing relationships between the CTT elements and existing UML elements.

The first solution is feasible and was already proposed in [2]. This approach represents CTT as stereotyped class diagrams. Constraints associated with UML class, association and dependency stereotypes are defined to enforce the structural correctness of the CTT models. The major drawbacks to this proposal are the expressiveness of the notation and the semantic validation of the CTT temporal constraints in terms of UML class diagrams.

The second solution is outlined in [3] and covers the definition of an UML for Interactive Systems. The approach proposed describes the integration points between UML models and task models in a complementary way. Yet, a unified integration at semantic and notational level should be provided towards an effective incorporation of task models into UML.

In this paper we propose a different approach that enables the integration of CTT in the UML through the extensions of UML 2.0 activity diagrams. Our approach takes advantage of the new characteristics of the UML 2.0 semantics, in particular the separation of statecharts and activity diagrams, that enables a better definition of the temporal operators underlying CTT, without compromising the usability of the notation. We strongly believe the enhancements in the UML 2.0 activity diagrams, finally enabled an effective integration of CTT into the UML. The solution presented here could provide a common ground to effectively bring task modeling into software engineering, promoting artifact interchange between tools and practitioners in SE and HCI.

The remaining of the paper is organized as follows. Section 2 and 3 briefly introduces ConcurTaskTrees and UML 2.0 Activity diagrams. Section 4 reports the evaluation of UML Activities Diagrams to express CTT semantics. Section 5 presents an extension to UML abstract syntax in order to support CTT concepts and the underlying notation. Finally, section 6 concludes the paper.

2 Overview of ConcurTaskTrees

ConcurTaskTrees is a notation that has been developed taking into account the previous experience in task modeling and adding new features in order to obtain an easy-to-use and powerful notation to describe the dialogue in interactive systems. CTTs are based in a graphical notation that supports the hierarchical structured of tasks, which can be interrelated through a powerful set of operators that describe the temporal relationships between subtasks. The formal semantics of the temporal relationships in CTT are defined using a Labelled Transition System (LTS) formalism. In addition, CTTs allow designers to indicate a wide set of optional task attributes, such as the category (how the task performance is allocated), type, manipulation of objects, frequency, and time requested for performance.

The CTT notation is supported by CTTE (the ConcurTaskTrees Environment), a set of freely available tools supporting editing and analysis of task models. The CTT notation is widely recognized as one of the more popular task notations in the HCI field, it is used for teaching and research purposes in several universities, and there is also evidence of usage in development projects. The CTT environment includes a simulator and a number of features enabling, for instance, designers to dynamically

adjust and focus their attention in subsets of large task models, while analyzing large specifications.

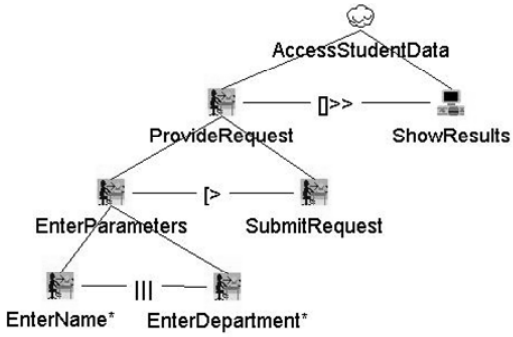

Fig. 1. Example of a ConcurTaskTree

Figure 1 illustrates a simple task model in CTT (provided in CTTE distribution). As we can see from the example, task models are represented in CTT as inverted trees (the task structured), each task can differ in type (user task, interactive task, abstract task and system task). The temporal relationship between subtasks at the same level is represented through lines annotated with the different temporal relationships. For a full description of CTT refer to [4].

3 Overview of UML 2.0 Activity Diagrams

Since the early versions of the standard, the UML included the popular Harel statecharts as the key notation to represent the dynamic aspects of software intensive systems. In addition, the UML 1.x versions also included activity diagrams, defined as a subset of statecharts. Activity diagrams have since become one of the key diagrams of the UML, in particular for business process modeling and user modeling [5].

Recognizing the problems with UML 1.x activity diagrams, in UML 2.0, activities where redesigned to follow "Petri-like" semantics thus separating activity diagrams from statecharts. Among other benefits, this widens the number of flows that can be modeled, especially those that have parallel flows of control [6]. The fundamental unit of behavior specification of an Activity Diagram is an Action. An action takes a set of inputs and converts them to a set of outputs, though either or both sets may be empty. There are three types of actions, namely:

- Invocation Actions, used for performing operations calls, signal sending and accept event actions;
- Read and Write Actions, for accessing and modifying objects and their values; and
- Computation actions, transforming input values into output.

Besides actions, an activity diagram can also contain control nodes, object nodes, activity edges and activity groups. In the Figure 2 we represent a subset of the UML 2.0 activity diagram notation, which is of particular interest to the discussion in this paper.

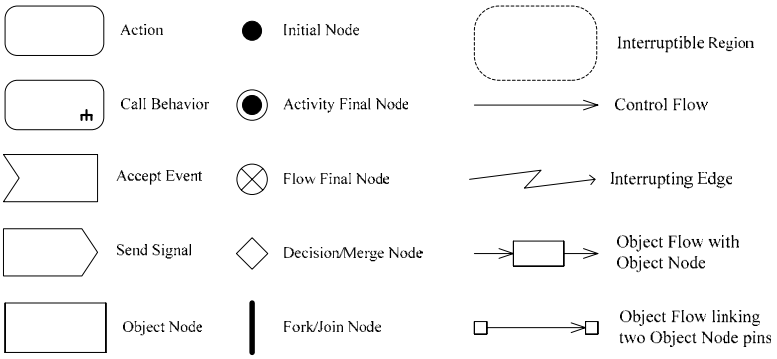

Fig. 2. UML 2.0 notation for Activity Diagrams

Among the possible activities groups we highlight the Interruptible Activity Region due to its intense use on this paper. This type of group is described as an activity group that supports termination of tokens flowing in the portions of an activity [6]. The termination of all tokens occurs when a token leaves an interruptive region via an interrupting edge. A token transition is never partial; it is either complete or it does not happen at all [6].

4 Mapping CTT into UML 2.0 Activity Diagrams

The analysis of the UML activity diagrams to represent CTT semantics is provided in terms of the behavior obtained by applying a temporal operator between two tasks. For the purpose of the evaluation of the approach described in this paper, one can consider atomic tasks (i.e., tasks that are not refined into subtasks) or composed tasks (resulting from application of temporal operators to subtasks). We highlight these definitions because they are slightly different from the original concepts in CTT. A CTT task is either an atomic task or the task resulting from applying a full sequence of operators to all sibling tasks. Furthermore, the composition of tasks (via temporal operators) requires a detail control of starting and terminating conditions of composed tasks. For instance, in the following sequence T1|=|T2>>T3, the T3 task only becomes enabled either after termination of the T1 or T2 tasks (depending upon the selected order). This particular type of dependency between tasks implies that we have to express the start and termination of composed tasks in terms of the start and termination of the tasks being composed. In our UML 2.0 based approach we use signals to control the aforementioned conditions. We assume that an atomic task signals its own start and termination of execution. Hence, the composition of tasks

must take in account the precedence of temporal operators, for instance, the sequence T1|||T2[>T3 must be considered as (T1|||T2)[>T3 and not as T1|||(T2[>T3).

To prevent the problem described previously, we consider the following sequence of operator precedence: >>, [>, |>, [], |=|, |||.

In our UML 2.0 approach, a CTT task is mapped into an Action, if it is atomic, and into a Call Behavior Action, otherwise. Finally, from the UML 2.0 semantics, a task is enabled if it possesses a token and disabled otherwise.

In the following subsections we examine the UML semantics for each CTT temporal operator. All the operators descriptions used were taken from [7].

4.1 Independent Concurrency (T1 ||| T2)

Description: Actions belonging to two tasks can be performed in any order without any specific constraint.

Proposed UML 2.0 mapping: The independent concurrency is captured by a Fork node to create two independent flows of control for each task and a Join node to synchronize them. The starting of the composed task T1|||T2 corresponds either to the start of T1 or T2. This condition could be modeled using two Accept Event actions for the start signals of T1 and T2 and, once of these actions succeed, a signal Start T1|||T2 is produced, through a Send Signal action. An interruptible region is necessary because only one signal must be produced. The termination of the composed task occurs when both tasks signals its terminations and must wait for both signals before sending a signal that correspond to the termination of task T1|||T2.

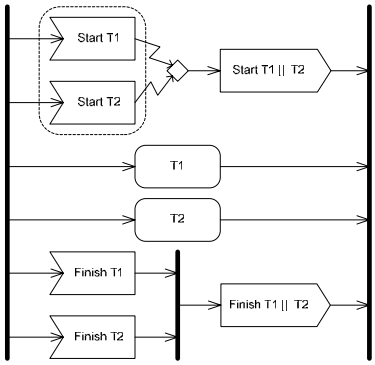

Fig. 3. UML specification for Independent Concurrency temporal operator

4.2 Choice (T1 [] T2)

Description: It is possible to choose from a set of tasks and, once the choice has been made the task chosen can be performed and the other tasks are not available at least until it has been terminated.

Proposed UML 2.0 mapping: In this case both tasks must be enabled at beginning but once one of them starts its execution the other must be disabled. This can be modeled using an interruptible region with the task and an Accept Event action for the Start signal of the other task. Using this strategy we ensure that only one task is executed. The start of T1[]T2 occurs when one of the tasks sends its Start signal (only one Start signal will be produced). The same occurs with the task termination.

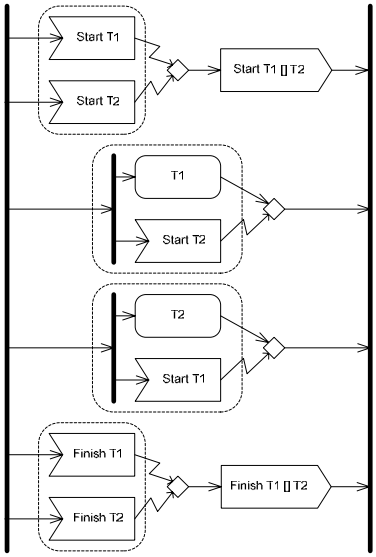

Fig. 4. UML specification for Choice temporal operator

In the previous figure two Accept Event actions are used to signal Start T1 (and also to signal Start T2). In fact both signals should be considered as only one. This simplification is adopted here and in the following figures, in order to increase the readability of the diagrams.

4.3 Concurrency with information exchange (T1 | [] | T2)

Description: Two tasks can be executed concurrently but they have to synchronize in order to exchange information.

Proposed UML 2.0 mapping: The solution is identical of the one presented for independent concurrency operator. Additionally, a Central Buffer node must be used for information exchange purposes.

4.4 Order Independence (T1 | = | T2)

Description: Both tasks have to be performed but when one is started then it has to be finished before starting the second one.

Proposed UML 2.0 mapping: The solution for this operator is similar to the choice operator. The difference is that when a task is disabled (due to the execution of the other), we must wait for the termination of the execution, before enabling the task again. Moreover, both tasks must be executed before the send of Finish T1|=|T2 signal.

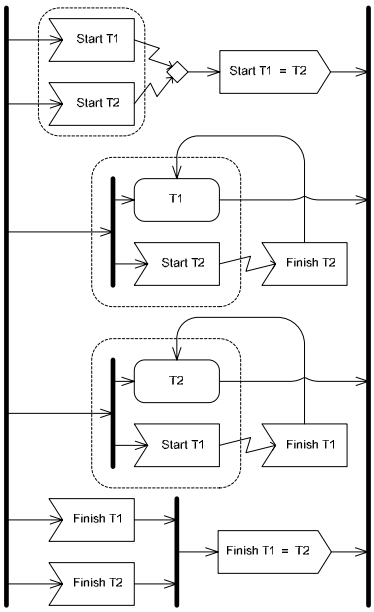

Fig. 5. UML specification for Order Independent temporal operator

4.5 Deactivation (T1 [> T2)

Description: The first task is definitively deactivated once the first action of the second task has been performed.

Proposed UML 2.0 mapping: If the task T1 executes normally, T2 must be disabled after the completion of task T1. This case is ensured by grouping task T2 and an Accept Event action for Finish T1 signal in an interruptible region. The other case, when T2 aborts the execution of T1, is modeled using an interruptible region and an Accept Event action for Start T2 signal, thus if T2 starts its executions T1 will be interrupted. The start of T1[>T2 corresponds to the start of T1 or the start of T2 with-

out starting T1. The termination corresponds to the termination of one of the two tasks.

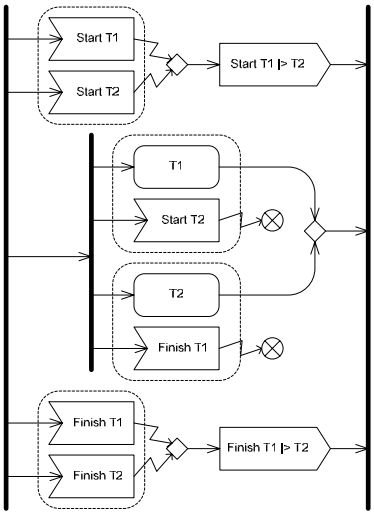

Fig. 6. UML specification for Deactivation temporal operator

4.6 Enabling (T1 >> T2)

Description: In this case one task enables a second one when it terminates.

Proposed UML 2.0 mapping: In this case a control flow is used to connect both tasks. The start of T1>>T2 corresponds to the start of T1 and the termination corresponds to the termination of T2.

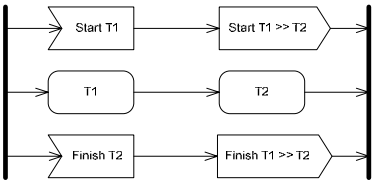

Fig. 7. UML specification for Enabling temporal operator

4.7 Enabling with information passing (T1 []>> T2)

Description: In this case task T1 provides some information to task T2 other than enabling it.

Proposed UML 2.0 mapping: Similar of enabling operator assuming in this case an object flow between task T1 and T2.

4.8 Suspend-Resume (T1 |> T2)

Description: This operator gives T2 the possibility of interrupting T1 and when T2 is terminated, T1 can be reactivated from the state reached before the interruption.

Proposed UML 2.0 mapping: There is no evidence in the UML 2.0 specification that any behavior supports the resume functionality. Therefore, this operator is not supported by existing UML semantics.

4.9 Iteration (T*)

Description: The task is performed repetitively.

Proposed UML 2.0 mapping: This unary operator has straightforward mapping in UML. The Start signal occurs at first execution of task T1 and a flow loop is created for task T1.

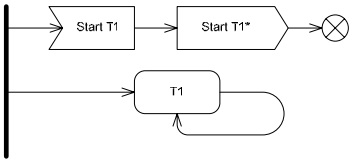

Fig. 8. UML specification for Iteration operator

4.10 Finite Iteration (T1(n))

Description: It is used when designers know in advance how many times a task will be performed.

Proposed UML 2.0 mapping: A finite iteration can be mapped into UML using a local variable for counting the iterations. The start of the iteration begins with the first execution of task T1 and termination is signaled after n occurrences of Finish T1 signal. We use in this case an exception rule to the normal execution in Activities: If an AcceptEventAction has no incoming edges, then the action starts when the containing activity or structured node does. In addition, an AcceptEventAction with no incoming edges is always enabled to accept events, no matter how many it accepts. [6]

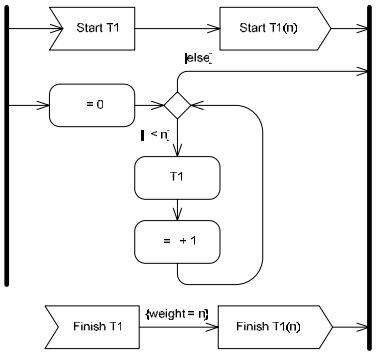

Fig. 9. UML specification for Finite Iteration operator

5 An UML notation for ConcurTaskTrees

In the previous section we described how the new UML 2.0 standard can successfully support the CTT semantics taking advantage of the redesigned activity diagrams. However, even a simple task tree results in very complex sets of activities with a remarkable number of actions and control nodes. This is a well-known problem with statechart like notations, that become unreadable as the number of states increases. Although semantically correct, the previously described UML mappings to the CTT temporal operators, will be completely useless even for a simple task model. This was one of the major problems with the previous proposals to map CTTs into the UML: in order to propose a useful solution to the mapping of temporal relationships one would compromise the semantic correctness and the other way around.

The existing graphical representation for CTTs, based on a hierarchical structure, is one of the most significant factors of its success. The hierarchical structure reflects the logical approach of most designers, allowing the description of a rich set of possibilities that is both highly declarative, and generates compact descriptions [4]. In the following we propose to solve this dilemma with a small increment to the UML abstract syntax. With this approach the concepts required for modeling CTTs can be added to the UML. In the following figures we detail this original approach to extend the UML abstract syntax.

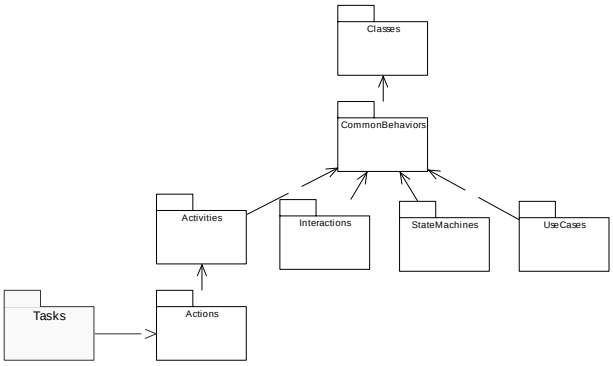

Fig. 10. Package dependencies

As we can see from Figure 10, in order to isolate the extensions from the actual UML specification, we create a new Package named Tasks to contain the new concepts required for task modeling.

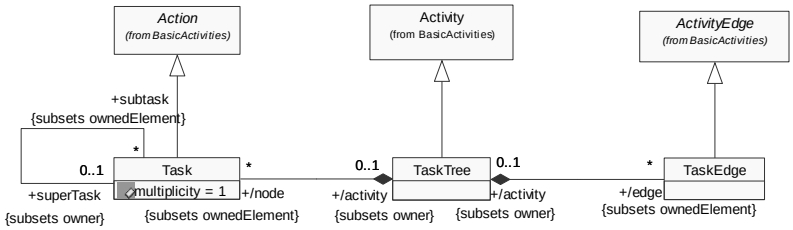

Fig. 11. Tasks Package

Figures 11 and 12 details the Task package. We introduce three new concepts: TaskTree as a specialization of the Activity concept; Task for modeling the concept of task; and TaskEdge for modeling temporal operators. These new concepts allow the creation of a specialized type of Activity Diagrams for modeling task trees (we may name this diagrams Task Tree Diagrams).

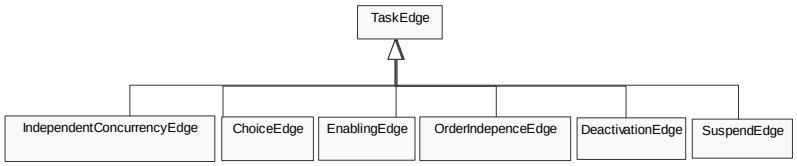

Fig. 12. Tasks Operators

In order to foster recall from the existing UML 2.0 notation, we decided to maintain a very close relationship between the new task concepts and the existing activity

diagrams. Figure 13 illustrates the extension to the UML 2.0 abstract syntax and provides an example for each of the previously described temporal relationship.

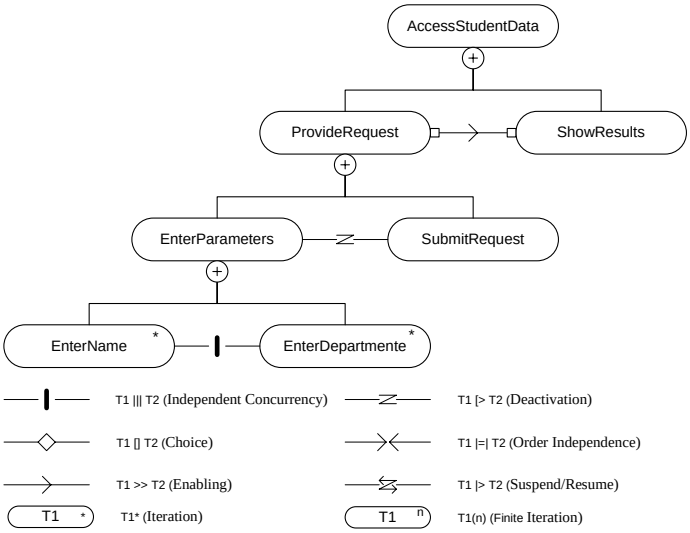

Fig. 13. Example and summary of the notation

The previous illustration depicts a simple task tree that illustrates our approach (already presented in figure 1 using CTTs notation). The notation used for temporal operators are inspired in UML activities notations, namely Independent Concurrency, Choice and Deactivation have obvious similarities with Fork/Join Node, Decision Node and Interrupting Edge, respectively. For information exchange between tasks we adopt the Object Flow and Pin notation showed in the link between ProvideRequest and ShowResults tasks. The relations between a task and its refinement subtasks are inspired on the notation for showing Packages and its contents on the same diagram, providing an adequate hierarchical representation.

6 Conclusions

We show in this paper that the ConcurTaskTrees semantics can be expressed in terms of UML Activities semantics, allowing a truly unified integration of task model concepts within UML framework, fostering co-evolutionary development of interactive systems, providing a common ground to effectively bring task modeling into software engineering and promoting artifact interchange between tools and practitioners in SE and HCI. The Petri-net like semantics of UML 2.0 Activities represents a clearly improvement over previous versions and brings this new opportunity to integrate CTT in UML. The extensions and the adapted notation described here keep expressiveness and effectiveness of CTTs notation and reduce the difficulty of acceptance of another notation. Finally, an activity based semantic for CCTs can take full advan-

tage of existing work on verification and execution of activities diagrams and promotes the inclusion of task modeling in model-based approaches.

References

1. van Harmelen, M., Artim, J., Butler, K., Henderson, A., Roberts, D., Rosson, M. B., Tarby, J. C., Wilson, S.; Object Models in User Interface Design, 29(4), SIGCHI Bulletin, New York, ACM, 1997
2. Nunes, N. J., Cunha, J. F.: Towards a UML profile for interactive systems development: the Wisdom approach, in Proceedings of UML´2000 Conference, Kent – UK, A. Evans (Ed.), Springer Verlag LNCS, New York (2000) 50-58
3. Paternò, F: Towards a UML for Interactive Systems, in Proceedings of Human-Computer Interaction Conference HCI'2001 (2001) 175-185
4. Paternò, F: Model-Based Design and Evaluation of Interactive Applications, Springer Verlag (1999)
5. Patricio, L., Cunha, J. F., Fisk, R., Nunes, N. J.: Designing Interaction Experiences for Multi-Platform Service Provision with Essential Use Cases in Proceedings of the 9th international conference on Intelligent user interface: Short Papers, Funchal, Madeira, Portugal pp. (2004) 298-300
6. OMG: UML 2.0 Superstructure Specification, Revised Final Adopted Specification (ptc/04-10-02) October 8 (2004)
7. Mori, G., Paternò, F., Santoro, C.: CTTE: Support for Developing and Analysing Task Models for Interactive System Design, IEEE Transactions on Software Engineering, Vol. 28, No. 8, IEEE Press, (2002) 797-813