ConcurTaskTrees and UML: how to marry them?

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

Abstract

Nowadays, UML is the most successful model-based approach to support software development. However, during the evolution of UML little attention has been paid to supporting the user interfaces design and development. In the meantime, the user interface has become a crucial part in 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. In this position paper there is a discussion of how the two approaches can be integrated and why this is a desirable goal.

1. Introduction

The success of UML [6] reflects the success of object-oriented approaches in software development. UML is already composed of nine notations. In practice, software engineers hardly 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 [11], various approaches have been developed in the field of model-based design of interactive applications [2] [3] 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. The issue of how to integrate model-based approaches to user interfaces with UML was discussed in some CHI workshops where participants were in agreement with the concept that it is possible to link development practice to HCI practice through the use of object models derived from task analysis and modelling [4].

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 easy as possible how the system available supports such activities, how they have to manipulate the interaction and application objects for this purpose 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) [2] 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 powerful set of operators describing the temporal relationships among tasks have been defined (also their formal semantics has been given). In addition, it allows designers to indicate a wide set of optional task attributes, 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. The notation is supported by CTTE (the ConcurTaskTrees Environment), a set of tools supporting editing and analysis of task models. 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). 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.

Aiming at integrating the two approaches (CTT and UML) there can be two basic philosophies that are outlined below. However, both approaches exploit, to different extents, the extensibility mechanisms built into UML itself (constraints, stereotypes and tagged values [6]); these enable extending UML without requiring to change the basic UML metamodel. The two types of approaches are:

1. *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, it can be represented as UML Class Diagrams. Specific UML class and association stereotypes, tagged values and constraints can be defined to factor out and better represent properties and constraints of CTT elements [5];
2. *Building a new UML for interactive systems*, which means to explicitly insert CTT in the set of available notations, still creating semantic mapping of CTT concepts into UML metamodel. This encompass 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 for both approaches. In the former 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 [5]: 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 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 are either hard to use or they require rather complicated expressions to

express task models describing flexible behaviours. To overcome these problems, in the sequel I will outline a proposal aiming at integrating the use of CTT and UML.

2. The Approach Proposed

The basic idea is that the UML for Interactive Systems should explicitly introduce the use of CTT. The approach should also be supported by a defined process and a related tool supporting an integrated us of CTT and the other notations useful for supporting the design in this class of software applications.

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, during the requirement elicitation phase, use cases, supported by related diagrams should be used. They have shown to be successful in industrial practise. 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 into 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.

Next, there is the task modelling phase that allows designers to obtain an integrated view of functional and interactional aspects. Especially, interactional aspects cannot be well captured in uses cases and to overcome this limitation they can be enriched with scenarios, informal descriptions of specific use 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 highly declarative, and generates compact descriptions.

The decomposition of the task model can be carried on until basic tasks (tasks associated with atomic actions) 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.

During the task modelling work, the domain modelling is also refined because designers need to associate tasks with objects in order to indicate what objects should to be manipulated to perform each task. 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 (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.

The association between domain objects and Use Cases can be inherited and further refined for the elements of the CTT model. This exploits the fact that in the 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, given the number of notations for object modelling existing, no new notation has been introduced for this purpose. Thus an integration with objects modelled with UML class diagrams can be easily and usefully performed.

CTTE, the tool supporting CTT, gives also the possibility of simulating the task model. This means that it is possible 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. Thus, scenarios can be represented using interaction diagrams that well represent limited sequential activities.

Finally, once the design has been sufficiently developed and analysed, it is possible to obtain the concrete user interface. A number of criteria have been identified [2] to take information in the task model that it is useful to obtain effective user interfaces: grouping of tasks can be reflected in grouping of presentation objects, temporal relationships among tasks can be useful to structure the user interface dialogue, and the type of tasks and objects (and their cardinality) should be considered when the corresponding widgets and presentation techniques are selected.

In conclusion, 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 activity diagrams are a specialisation of). However, these notations have mainly been conceived to describe either high level user work-flow activities, or low level interactions among software objects. They can be used to describe task models or user interactions - some research work in this direction has been done - but the results do not seem particularly convincing. Activity Diagrams do not seem to support well the possibility of 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: it is easy to obtain graphical representations rather difficult to interpret. It is possible to define such diagrams hierarchically but then it is not immediate for designers to follow their relationships and obtaining an overall view of the specification.

3. Conclusions and Discussion

This position paper has outlined an approach to integrating UML and CTT. The importance of task models and the notations that have shown to be effective in representing them cannot be overlooked by a successful modelling environment for interactive systems. The logical framework that has been introduced needs to be refined in many directions. However, it supplies clear indications of how to take into consideration research work on model-based design of user interfaces in the last decade to improve UML.

Further, both CTT and UML are supported by automatic tools freely available and implemented in java. Automatic support of the new method introduced in this paper seems to be a reasonable goal. We plan to apply this method in a case study considered in the GUITARE R&D European Project.

Acknowledments

I wish to thank Nando Gallo (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 at http://giove.cnuce.cnr.it/guitare.html

4. References

[1] 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.

[2] 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.

[3] Puerta, A., A Model-Based Interface Development Environment, IEEE Software, pp. 40-47, July/August 1997.

[4] Artim, J., et al, Incorporating Work, Process and Task Analysis Into Commercial and Industrial Object-Oriented System Development, SIGCHI Bulletin, 30(4), 1998.

[5] N.Numes, J.Falcao, "Towards a UML profile for user interface development: the Wisdom approach", Proceedings UML'2000, Springer Verlag.

[6] J.Rumbaugh, I.Jacobson, G.Booch, "The Unified Modeling Language Reference Manual", Addison-Wesley, 1999.