USING TRANSFORMATIONS TO INTEGRATE TASK MODELS IN THE UML

Position Paper to the WTUML: Workshop on Transformations in UML
ETAPS'2001 – European Joint Conference on Theory and Practice of Software

Nuno Jardim Nunes	Fabio Paternò	João Falcão e Cunha
njn@math.uma.pt	fabio.paterno@cnuce.cnr.it	jfcunha@fe.up.pt
Univ. Madeira - Portugal	CNUCE-C.N.R. – Italy	FEUP – Portugal

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 describes the activities in which users are involved to achieve some goal. A task model details the users' goals and the strategies adopted to achieve those goals, in terms of the actions that users perform, the objects involved in those actions and the sequencing of activities [van Harmelen et al 1997].

From an object-oriented software engineering perspective, task models can be considered similar to use-case models. However the definition of a use-case model only refers to system functionality that provides users with results of value [Jacobson et al 1999]. As we discussed in [Nunes and Cunha 2000a], the adoption of use-cases in the UML acknowledges the importance of identifying the different roles users play when interacting with an application to support their tasks. However they are still mainly used to structure the application internals and don't provide an efficient way to support the usability aspects of the interactive system. Essential use-cases identified some of the requirements for user interface development enabling the connection between the structure of use and the structure of the user interface. Therefore, the UML compromises notations have mainly been conceived to describe, either high-level user workflow activities for the purpose of functional decomposition of the system internals (use-cases and activity diagrams), or low-level interactions among software objects (sequence and collaboration diagrams).

2 Integrating Task Models in the UML

ConcurTaskTrees (CTT) [Paternò 1999] is a notation that has been developed taking into account previous experience in task modeling 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 was 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 modeling 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 analyzing large specifications.

As we discussed in [Paternò 2000], integrating CTT and the UML can be generally achieved with two types of approaches:

- Extending the UML metamodel, introducing a separate user task model, and establishing relationships between the CTT elements and the existing UML elements. Thus enabling traceability between tasks and other UML model elements. This approach was attempted in a CHI workshop [Artim et al 1998] but involves a number of problems. On the one hand, the UML metamodel is not yet entirely compliant with the OMG's MOF architecture [OMG 1999]; hence this kind of heavyweight extension mechanism will likely introduce more inconsistencies in the existing UML semantics. Furthermore it is not possible, in the current UML standard, to devise mappings between the new extended semantics and the notations. On the other hand, heavyweight extensions require tools that support metamodeling facilities and also impact the interchange formats [Cook 2000];
- Using the UML lightweight extension mechanisms (profiles) to represent elements and operators of a CTT model by an existing UML notation. For instance, 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;

Although the first solution seams unlikely, in the light of the existing UML standard, the second solution is feasible and was already proposed in [Nunes and Cunha 2000a]. This approach, illustrated in Figure 1, represents CTT diagrams as stereotyped class

diagrams. Constraints associated with UML class, association and dependency stereotypes are defined so to enforce the structural correctness of CTT models.

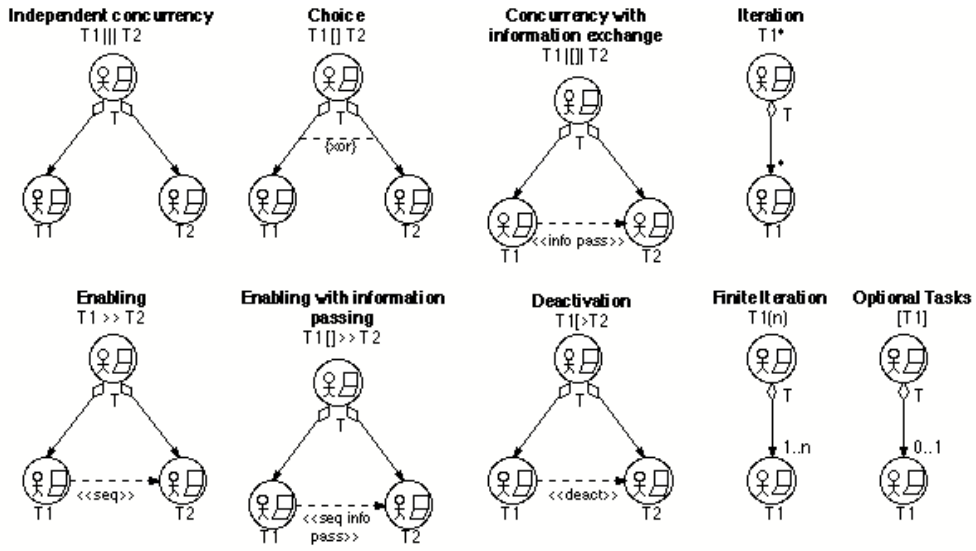

Figure 1 – Summary of the UML adaptation of ConcurTaskTrees [Nunes and Cunha 2000; Nunes and Cunha 2001]

However, the key issues are not uniquely related to the correctness of a given UML extension. The expressive power of notations should be effective and support designer's in their work rather than complicate it. This usability aspect of notations 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 behaviors.

3 Transforming CTT Models into the CTT UML extension

One solution for the problem of expressing task models in the UML that are usable and supported by specific tools, is to take advantage of the UML XMI interchange format. The proposed solution encompasses using two XSL style sheets to transform the UML XMI extension into the CTT XML schema and vice-versa. This process enables user interface designers or HCI experts to use the powerful CTT notation, and the associated tools, while being able to change artifacts with their software development counterparts.

This UML transformation is briefly outlined in the following figures. The example used here is from a check availability task in a hotel reservation system (refer to [Nunes and Cunha 2000b] for a detailed description of this example).

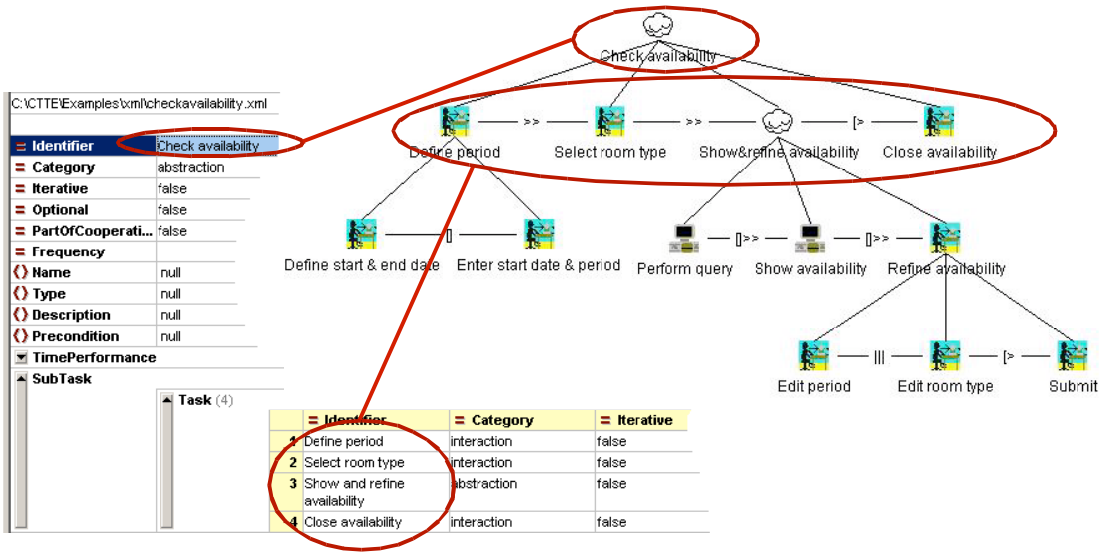

Figure 2 – A CTT Model expressed in the original CTT notation

Figure 2 illustrates a CTT task model in both diagrammatic and XML formats. The diagrammatic model, at the far right, was produced using the CTTe tool. At the left of the figure is the corresponding XML specification generated by the CTTe tool and later opened for browsing in a popular XML editor. As we can see from the red highlighting, the hierarchical structure of the task tree is built using a sequence of <task> <subtask> XML tags. Although it's not visible in the figure, the temporal relationships are specified with the following tags per subtask: <Parent name="Parent Task"/> <SiblingRight name="subtask"/>.

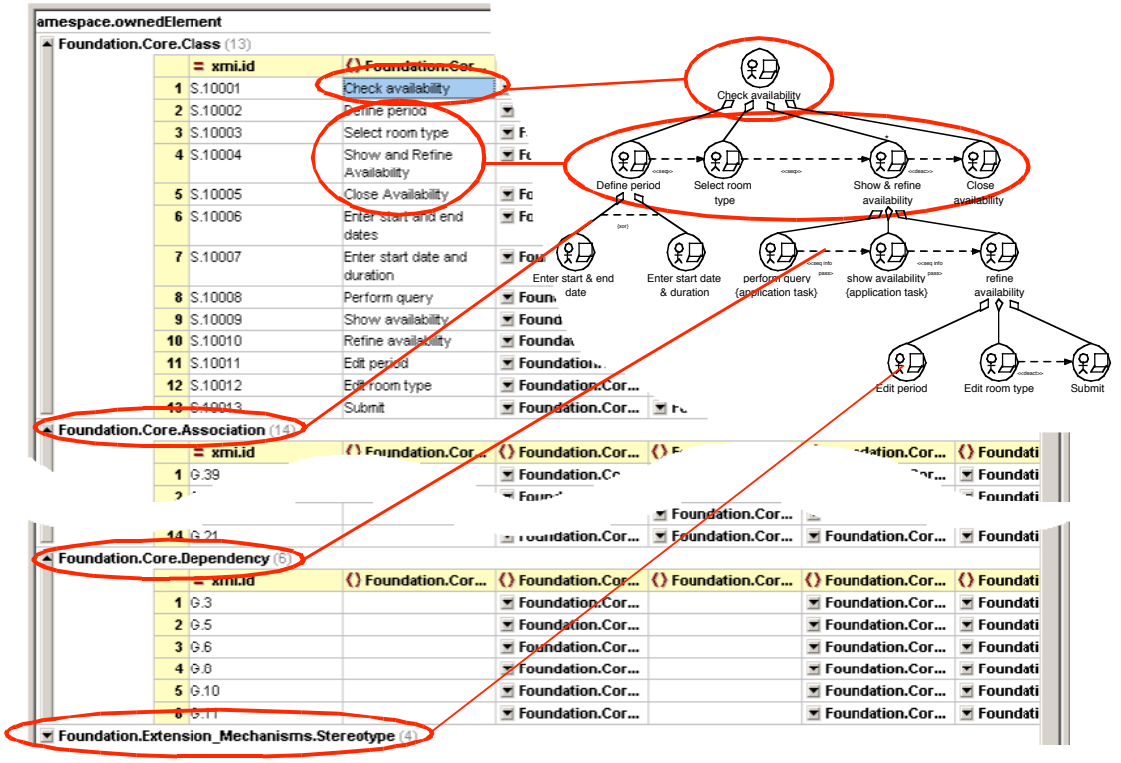

Figure 3 – A CTT Model expressed in the CTT UML extension

Figure 3 illustrates the same task model expressed with a UML class diagram (at the upper right) and the corresponding XMI specification. As we can see from the figure, the XMI schema is significantly different form the CTT XML schema. The different classes in this example are under the XML tag <Foundation.Core.Class>; the hierarchical structure is captured with association stereotypes and is under <Foundation.Core.Association>; and finally the temporal relationships are captured with dependency stereotypes under <Foundation.Core.Dependency>. In addition all the stereotypes for the core elements are under <Foundation.Extension_Mechanisms.Stereotype>. This way the XSL style sheet must lookup the different stereotypes from the corresponding section and them filter the different modeling components. The hierarchical decomposition of tasks is reconstructed following the aggregation relationships and the temporal relationships generated from each level of subtasks. For a detailed description of the CTT UML extensions refer to [Nunes and Cunha 2000a].

4 Conclusion and Discussion

In this paper we described an approach were we use the XMI interchange format and XSL style sheets to transform a UML extension into a well known and commonly used task-modeling notation. Our approach enables designers to take advantage of existing modeling, eliciting and simulation tools for the CTT task notation; while supporting automated artifact change with UML tools and developers of the internal software functionality that supports the end user tasks. This approach overcomes one of the major problems with the UML standard for expressing user behavior while performing envisioned or existing tasks in interactive systems.

We recognize that the approach described here is somewhat pragmatic and we hope the new UML 2.0 standard will support more flexible extensions to the language. For the problem at hand, we would require an extension facility that enables, not only extensions to the UML metamodel, but also a mechanism that enabled us to specify and connect the syntax (notation) of the extension to the new semantics. This mechanism should be consist with the interchange formats and also maintain interchange consistency at the notational level (including layout).

The usability and expressive power of notations is an important factor to support developers in their work, thus, providing effective ways to build new and high-quality software intensive systems. Fortunately it seams the forthcoming UML revision is taking a step towards this direction, enabling the standard to include different contributions and notations from other fields involved in software development. It is our belief that this

step is ultimately important for the future of the UML – to become in fact a consistent and congruent family of languages.

5 References

[Nunes and Cunha, 2000a] Nunes, N. and J. F. Cunha, Towards a UML profile for interactive systems development: the Wisdom approach, in Proceedings of the UML'2000 Conference, Kent – UK, A. Evans (Ed.), Springer Verlag LNCS, New York, 2000.

[Nunes and Cunha 2000b] Nunes, N. and J. F. Cunha, Wisdom: A Software Engineering Method for Small Software Development Companies, IEEE Software 17 (5), Sep./Oct. 2000.

[Nunes and Cunha 2001] Nunes, N. and J. F. Cunha, Whitewater Interactive System Development with Object Models (Wisdom), in van Harmelen (Ed.), Object-Oriented User Interface Design, Addison-Wesley Object-Technology Series, 2001 (to appear).

[Paternò 1999] Paternò, F., Model-Based Design and Evaluation of Interactive Applications, Springer-Verlag, London, UK, 1999.

[Paternò 2000] Paternò, ConcurTaskTrees and UML: how to marry them?, Position paper to the Tupis'2000 Workshop at UML'2000, http://www.math.uma.pt/tupis00

[Jacobson et al 1999] Jacobson, I., Booch, G., Rumbaugh, J., The Unified Software Development Process, Addison-Wesley Object Technology Series, 1999.

[Cook 2000] Cook. S., The UML Family: Profiles, Prefaces and Packages, in Proceedings of the UML'2000 Conference, Kent – UK, A. Evans (Ed.), Springer Verlag LNCS, New York, 2000.

[van Harmelen et al 1997] van Harmelen, M., J. Artim, K. Butler, A. Henderson, D. Roberts, M. B. Rosson, J. C. Tarby, S. Wilson, Object Models in User Interface Design, 29(4), SIGCHI Bulletin, New York, ACM, 1997.

[Artim et al 1998] Artim J. M., M. van Harmelem, K. Butler, J. Guliksen, A. Henderson, S. Kovacevic, S. Lu, S. Overmeyer, R. Reaux, D. Roberts, J. C. Tarby, and K. Vander Linden, Incorporating work, process and task analysis into industrial object-oriented systems development, SIGCHI Bulletin, 30(4), New York, ACM, 1998.