

Generating UML Diagrams from Task Models

Shijian Lu[†], Cécile Paris[†], Keith Vander Linden[‡] and Nathalie Colineau[†]

[†]CSIRO/CMIS, Locked Bag 17,
North Ryde, NSW 1670, Australia.
{shijian.lu cecile.paris,
nathalie.colineau}@csiro.au

[‡]Department of Computer Science, Calvin
College, Grand Rapids, MI 49546, USA.
kvlinden@calvin.edu

ABSTRACT

The importance of task analysis and modelling to software system development is well recognised. After all, the central mission for most software systems is to help users accomplish their tasks. However, while object-oriented analysis and design (OOAD) has been established as the major paradigm in the software industry, effectively incorporating task analysis and modelling into that process is still in the realm of research. In this paper, we propose a novel approach for integrating OOAD and Task Modelling. By exploiting the common semantic ground between task models and system behaviour models, we describe how UML diagrams, such as use cases, use case diagrams and scenario diagrams, can be automatically generated from task models represented in a semi-formal notation. We demonstrate our approach with a working example.

Keywords

UML, task models, information reuse, methodology, tools, object-oriented analysis and design, domain models.

1. INTRODUCTION

The importance of software usability has long been voiced by the user community and fervently advocated by the HCI community. However, it has only slowly gained popularity in the software development community. There are numerous reasons for this. In the early days, computer resources were very expensive, which meant that it made sense to focus on the delivery of the core functionality. Further, the notion of usability is complex and hard to identify clearly.

In the software development community, HCI techniques tend to be employed toward the end of the software development life cycle, primarily for evaluation purposes (e.g., usability evaluation). In practice, therefore, the impact of usability evaluation on software development is limited because by the time it is employed, it is too late to make significant changes to the design of a system. It is now well acknowledged that usability issues should be considered in the early design stages. One well-established HCI technique, task analysis and modelling, could potentially have a positive impact on software usability if used early enough in the life cycle.

Most interactive systems are intended to help users accomplish their tasks. It would, therefore, seem logical to ensure that the tasks in which users are to be supported are well understood either before designing the system or in the early stages of its design. That is where task analysis and modelling (TA&M) comes in. So far, however, its use and impact on software development have been minimal. One of the main obstacles to a rapid uptake of task analysis and modelling is that it is predominately a manual process, and the result (a task model) is not always formalised. This not only makes it a laborious task in itself, but it also renders difficult the reuse of the resulting model in subsequent stages of the design process. To alleviate these problems, researchers have in recent years developed software tools to support task analysis [2, 14, 3, 10, 6].

The issue of the integration of task analysis and modelling into the object oriented design process still remains. Currently, two main approaches have been proposed. One is to extend the UML notation to include a new set of features to cover elements specific to task modelling [1, 11]. The other is to represent task model elements using existing UML constructs such as stereotyping/profiling [7, 4]. Both approaches have shortcomings. In particular, with the first approach, it is likely to take a considerable amount time to specify and standardize a task-modelling notation in UML. As for the second approach, while feasible, the readability and comprehensibility (and thus the usability) of the resulting models is unclear, due to the extensive amount of stereotyping/profiling required. In this paper, we put forward a third, more pragmatic approach, namely the generation of system behaviour models from task models by leveraging on the common semantic ground between the two. This allows the re-use of task models in system design, while keeping the different models (with their different concerns) separate, thus preventing overloading a notation.

The work described in this paper is inspired by an earlier study on the relationship between system behaviour models and task models [5]. In that study, we found that there was a semantic common ground between system behaviour models and task models. At the time, our motivation was to find information sources that facilitate the construction of task models to be used as input to an

instruction generation system. Our motivation now is to explore ways for integrating task analysis and modelling into the object oriented design process.

Conceptually, going from task models to system behaviour models or from system behaviour models to task models is similar to forward and reverse engineering. So it is not surprising that different issues arise. In this paper, we report on some of the issues we encountered when going from task models to system behaviour models, and how we resolved them.

We use Diane+ [13] as our task modelling formalism. As with most task modelling notations, Diane+ allows for the representation of hierarchically structured tasks and provides a variety of procedural annotations for these tasks. We chose Diane+ over other task models because of its coverage of the information required to produce documentation (the original goal of our project was to automate the production of written instructions) [13].

The paper is organised as follows. For the sake of completeness, section 2 provides a detailed analysis of how task models in Diane+ can be transformed into UML diagrams, while section 3 discusses how domain knowledge might be used to improve this transformation process. Our prototype system is introduced in section 4, together with a working example. Finally, section 5 presents some concluding remarks.

3. TASK MODELS VERSUS UML DIAGRAMS

Semantic similarities between task and system models

Our previous work [5] has shown that some UML diagrams, i.e., use cases, use case diagrams and scenario diagrams, share a common semantic ground with task models. In brief, task models represent an overt, user-centred view of an application's behaviour, while system behaviour models provide an internal, system-centred view of that behaviour. This is illustrated in Figure 1. There are three basic building blocks here: the user, the system and the tasks. The relationship between them is that the user uses the system to perform tasks. There are also two intellectual constructs in the figure, the task model and the system behaviour model. Conceptually, the task model covers tasks to a large extent, some aspects of the system and limited aspects of the user. The system behaviour model, on the other hand, covers, to a large extent aspects of the system, some aspects of tasks, and limited aspects of the user. Clearly, there is an overlapping range between the two modelling constructs and this overlapping range is the semantic common ground.

By exploiting this common semantic ground, system behaviour models can be a source of information for task models. More importantly, though, it provides a basis for generating system behaviour models from task models,

thus allowing the integration of task models into the object oriented design process.

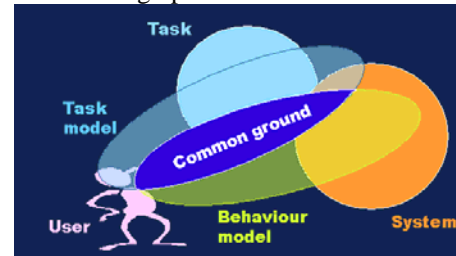


Figure 1. Relationship between task models and system behaviour models

Detailed mapping

Although task and UML models share common semantic features, they have different features. This section outlines how various task model constructs in Diane+ can be mapped to system behaviour model constructs as defined in UML.

The root of a task model & the main use case diagram

As mentioned in section 2, Diane+ is a hierarchical task model notation. The root of a Diane+ task model can either contain a single (decomposable) task or multiple tasks. It represents the highest level of abstraction in the task model. It thus corresponds to the main use case diagram in the use-case view category in the Rational Rose [12].

Composite tasks versus use cases

Composite tasks in a task model are equivalent to use cases in UML, although, in Diane+, a task has a much richer set of attributes than are currently defined in use cases. In fact, use case attributes are a sub-set of Diane+'s composite task attributes. Therefore, it is relatively easy to map composite task attributes to use case attributes.

In UML, there are only 2 core use case attributes: name and documentation. Clearly, composite task names can be directly mapped to use case names. As for use case documentation, it can be seen as the aggregation of the following composite task attributes: Precondition, constraints, feedback, actor, and comment. The rest of task attributes do not have any effect on generated use cases.

Task decompositions versus use case diagrams and scenario diagrams

A composite task is typically decomposed into a sequence of simple tasks or a collection of (linked) simple/composite tasks or/and Boolean connectors. A use case diagram will be created for task decompositions except those task decompositions which contain a sequence of elementary tasks. In that case, a scenario diagram will be created instead.

Generating use case diagrams

Use case diagrams explicitly represent the relationships between actors and use cases. They also employ different

types of relationships between use cases. This section will discuss how use case diagrams can be extracted from task models.

In a use case diagram, the actor is explicitly identified by association with the use case. In Diane+, the actor of a task is implicitly defined by its “style” attribute. If it is an automatic task, then its default actor is “user”, otherwise the actor is “system”. Besides the definition of the default actor (“user”), it is possible to specify finer distinctions by explicitly stating the actor attribute, such as “editor”, “author”, or “application server”, etc. The task model thus is able to provide the information required to specify the actors in a use case diagram.

UML defines 2 types of relationships between use cases: “uses” and “extends”. The “uses” relationship indicates that the source use case must perform the target use case. Figure 2(b), for example, represents the fact that withdrawing cash requires account verification. This is similar to links in task models except that links in Diane+ signify logical and temporal order. Figure 2(a) represents the roughly equivalent fact that account verification must precede cash withdrawal. Note that the direction of “uses” relationship is opposite that of task link.



(a) Task model in Diane+



(b) Use case diagram in UML

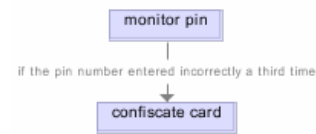
Figure 2. Task sequence and “use” relationship

The “extends” relationship expresses optional or conditional behaviour. This is very similar to conditional links between tasks. For example, there could be use cases for “monitor pin” and “confiscate card” in an ATM application. The “monitor pin” use case can be extended such that the “confiscate card” use case is activated if the pin number is entered incorrectly for a third time (Figure 3 (b)). This is expressed in Diane+ in Figure 3 (a). Again, note that the arrow directions for the “extends” and conditional link are opposite.

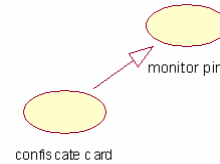
Generating scenario diagrams

Just as use cases and composite tasks are semantically similar, scenario/interaction diagrams are semantically similar to a task decomposition that contains a sequence of elementary tasks. However, in terms of scope, tasks are a subset of scenario diagrams since composite tasks are only concerned with overt system behaviour, while scenario diagrams are concerned with both overt and

covert (internal) system behaviour. Consequently, scenario diagrams generated from composite tasks will be a skeleton that includes only the interactions between the user and the perceivable system objects. More specifically, elementary tasks are roughly equivalent to messages between objects in a scenario diagram.



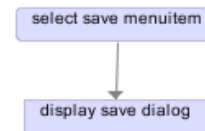
(a) Conditional link



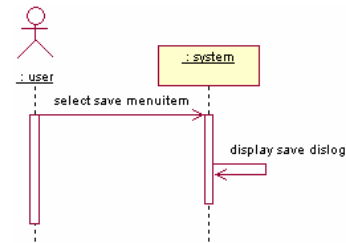
(b) Extends relationship

Figure 3. Conditional link and “extends” relationship

This can be illustrated by an example (Figure 4 (a)). There are 2 tasks in the Figure: an interactive (user) task, “select save menuitem”, which is linked to an automatic task: “display save dialog box”. Semantically, it depicts a situation where, when the user selects the “save” menu item, the system will display the “save” dialog box. This is expressed in a scenario diagram in Figure 4 (b).



(a)



(b)

Figure 4. From task sequence to scenario diagram

Figure 4 (b) indicates that when the user selects the “save” menu item, the system will send a message to itself to display the “save” dialog box. In actual design, the system object is likely to be composed of many sub-objects to complete this sequence of interaction. Nevertheless, even a skeleton sequence of interaction could be potentially beneficial to design usable systems. In section 4, we see how domain knowledge will help to generate more detailed interaction diagram.

Boolean connectors

Boolean connectors are also basic elements (constructs) of Diane+. They are used to show logical relationships between tasks. For example, in our task model editor Tamot [6], in order to design a model, the user can either “create new model” or “open existing model”. This can be expressed in Diane+ as in Figure 5 with a **OR** Boolean connector. Boolean connectors in Diane+ include XOR, OR, and AND relationships.

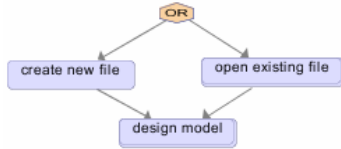


Figure 5. A Boolean connector example

Clearly, there is no equivalent notation in use case diagrams. Therefore, Boolean connectors are removed in the process of generating UML diagrams. Boolean connectors are removed as if all the entry links extend their linkage to all exit links. For instance, the task models shown in Figure 6 (a) and (b) will result the exactly the same use case diagram.

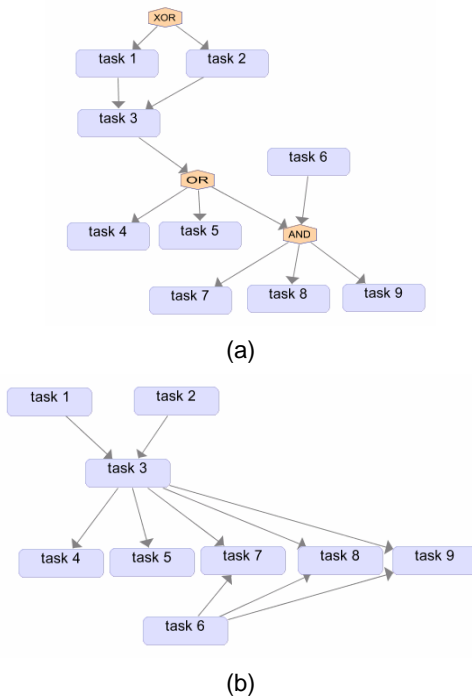


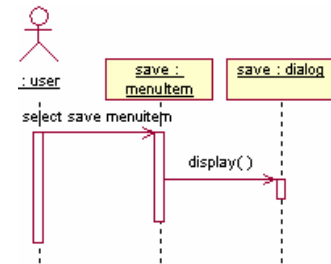
Figure 6. Equivalent task models in terms of UML generation

4. EMPLOYING DOMAIN KNOWLEDGE TO ENHANCE GENERATION

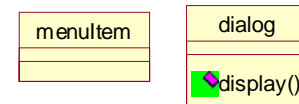
Task names are generally composed of three parts, *actor*, *process*, and *actee*. For instance, in the task “user select save menuitem”, “user” is the *actor*; “select” is the *process*; and “save menuitem” is the *actee*. This forms

part of domain knowledge we use in Tamot for instruction generation [9]. Improved scenario diagrams can be if the domain knowledge about the actions that can be performed (process), the agents (*actors*) that can perform them and the objects (*actees*) on which they are performed is known. Our task modelling editor, Tamot, also allows for the construction of such domain knowledge, either automatically, using a natural language module that parses the task names to obtain its primitive elements (actors, processes, and actees), or manually through an editor. This was necessary in our environment because this information is required by the instruction generator. As it turns out, we also found that this information is useful if a task model is to serve as the basis for generating draft system behaviour models. With domain knowledge, the following rules can be used to generate scenario diagrams:

- Create an actor stereotype for interactive tasks
- Create an object for the actee
- Create a class from the concept of the actee
- Create an operation from the process
- Create a message from the actor object to the actee object



(a) Scenario diagram



(b) Class diagram

Figure 7. A generated simple scenario diagram

If the above rules are followed, the scenario and class diagrams shown in Figure 7 can be generated. As we can see, instead of “system” class (Figure 4), “menuitem” and “dialog” classes are created from the actees in tasks “select save menuitem” and “display save dialog”. Furthermore, the “display()” operation is created in class “dialog”. After receiving the “select save menuitem” message, the save menuitem object activates “display()” method of the save dialog object.

Heuristics for creating scenario diagrams

The following heuristics can be used to convert a task diagram into an appropriate scenario diagram:

- When a non-automatic task is followed by an automatic task, use the actee of the proceeding task as the actor of the automatic task;
- When an automatic task is followed by an interactive task, use the actee of the first task as the actee of the second task;
- When an interactive task is followed by another interactive task, if there is any automatic task preceding these interactive tasks, then use the last automatic task's actee as these interactive tasks' actee.

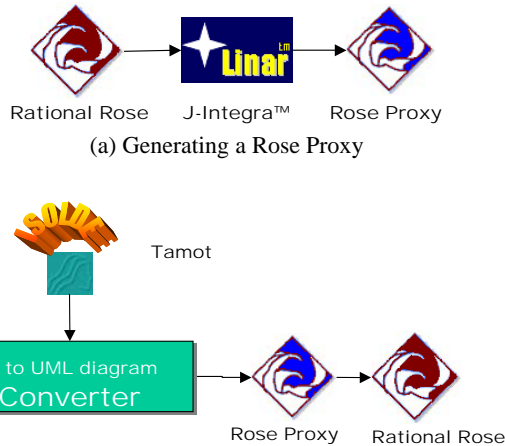


Figure 8. Schematic diagram for Task to UML Generator

5. THE PROTOTYPE AND AN EXAMPLE

The prototype

Based on the analysis presented here, we designed a prototype system that generates UML models from task models in Diane+, Task-to-UML (T2U). T2U, implemented in Java, creates use cases, use case diagrams, interaction diagrams, and class diagrams in Rational Rose remotely. This is achieved using J_Integra, a software tool for bridging Java and COM (Component Object Model) components. With J_Integra, Rose proxies can be generated from the Rose application (Figure 8(a)). The Rose proxies, looking like Java objects, present Rose COM properties, methods, and events as Java properties, methods, and events. As shown in Figure 8(b), the T2U converter sits between the Rose proxy and Tamot, our graphical task modelling tool [6]. By taking task models from Tamot as input, Rose proxy creates UML models in Rose as required by T2U.

An example

Now, let us consider the task model shown in Figure 9. This task model represents how to use the Tamot task model editor to model tasks. For the sake of illustration, the model is deliberately kept simple. The top most level is an abstract task “model task with Tamot”, which is decomposed into three sub-tasks: “create model”, “save model”, and “generate report”, linked with an OR. The

“create model” task is further decomposed into three sub-tasks: “create new model”, “open existing model”, and “design model”. There is another OR relationship between “create new model” and “open existing model”, while both tasks proceed the “design model” task. The “open existing model” task is elaborated in a sequence of elementary tasks, i.e., the user “select open menuitem from file menu” task followed by the system task of “display open file dialog”. After the user “select model file” and “click ok button”, the system responds by first displaying “opening file message dialog”, then by closing it.

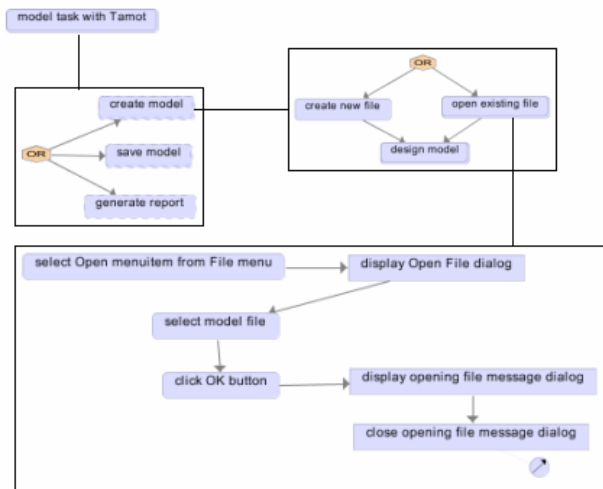


Figure 9. A sample task model

Figure 10 shows the corresponding generated UML model. The top-level task “model task with tamot” is naturally mapped as the use case diagram with the same name in the Use Case View category. Inside the diagram, there is one Actor (user) with three associated use cases. For the “create model” use case, there is a “use case diagram attached. That use case diagram includes three use cases: “design model”, “create new model”, and “open existing model”. “use” relations are generated from task links between the first and second as well as third use cases. Notice that the Boolean connectors in both tasks “model task with tamot” and “create model” are lost.

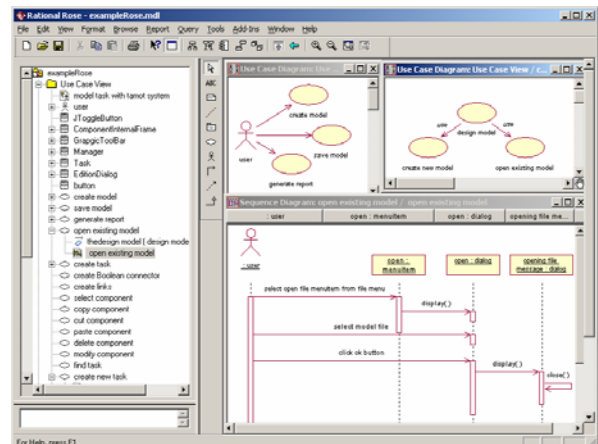


Figure 10. Generated UML Rose model

The “open existing model” use case is further elaborated with an interaction diagram where, apart from Actor “user”, three objects are created: open menu item, open dialog, and opening file message dialog, as well as message passing between them. Actually, while creating the interaction diagram, a class diagram is also created which contains two classes: menu item and dialog. These classes are created from the concepts of the actees in the tasks. For instance, the concept for the actee of “open file menuitem” is “menuitem”, so the class “menuitem” is created. When an actee appears in an automatic task, such as “open file dialog” in system task “display file dialog”, the operation will be added to the class associated with the actee, i.e., “display() operation for “dialog” class. At the end of the process, there will be 2 operations created for the dialog class: “display()”, and “close()”.

6. CONCLUSION

The importance of task analysis and modelling for the usability of software system is recognized, but its uptake in the industrial software design process is yet to come. One of the main obstacles to this uptake is the lack of mechanisms for conveniently integrating the resulting task models into existing software design processes. In this paper, we presented such a mechanism along with a software tool to automatically generate object oriented UML diagrams from task models. The effectiveness of this approach needs to be assessed. That is now our priority. This approach offers the following potential benefits:

- The tight integration between HCI techniques and software engineering design processes;
- Highly usable software systems;
- A bridge between HCI and software engineering community.

ACKNOWLEDGEMENTS

The authors wish to thank the Isolde team, including Sandrine Balbo, Thomas Lo, Nadine Ozkan, and Jean-Claude Tarby. We also acknowledge the support of ONR (grant #N00014-99-0906), CSIRO and Calvin College.

REFERENCES

- 1 Artim, J., van Harmelen, M., Butler, K., Guliksen, J., Henderson, A., Kovacevic, S., Lu, S., Overmeyer, S., Reaux, R., Roberts, D., Tarby, J.-C. and Linden, K. V. (1998). "Incorporating Work, Process and Task Analysis Into Commercial And Industrial Object-Oriented Systems Development", *ACM SIGCHI Bulletin*, Vol. 30, NO. 4, Oct. 1998.
- 2 Beard, D., Smith, D. & Danelsbeck, K. (1996), QGOMS: A direct-manipulation tool for simple GOMS models. In the *Proceedings of CHI 96 companion on Human factors in computing systems: common ground*. April 13 - 18, 1996, Vancouver Canada. Pp 25 – 26.
- 3 Bomsdorf, B. and Szwillus, G. (1999). Tool Support for Task-Based User Interface Design. In *Proceedings of CHI '99: The CHI Is the Limit*.
- 4 da Silva, P. and Paton, N. (2000). UMLi : The Unified Modeling Language for Interactive Applications, *Proceedings UML'2000*, LNCS, Springer Verlag
- 5 Lu, S., Paris, C. and Vander Linden, K. (1999). "Towards the automatic generation of task models from object oriented diagrams", in *Engineering for Human-Computer Interaction*, Stephane Chatty and Prasun Dewan (eds), Kluwer academic publishers, Boston, 1999.
- 6 Lu, S., Paris, C. and Vander Linden, K. (2002) "Tamot: towards a usable tool for supporting task modelling", in preparation.
- 7 Numes, N. and Falcao, J. (2000). Towards a UML profile for user interface development: the Wisdom approach, In *Proceedings UML'2000*, LNCS, Springer Verlag.
- 8 Paris, C., Balbo, S. and Ozkan, N. (2000). "Novel Uses of Task Models: Two Case Studies", In *Cognitive Task Analysis*. J.M.C. Schraagen, S.E. Chipman and V. L. Shalin (Eds.), Mahwah, NJ: Lawrence Erlbaum Associates. 261:274.
- 9 Paris, C. and Vander Linden, K and Lu, S. (2002). "Automated Knowledge Acquisition for Instructional Text Generation", submitted.
- 10 Paternò, F, Mori, G., Galimberti, R. (2001). CTTE: "An Environment for Analysis and Development of Task Models of Cooperative Applications", In *Proceedings ACM CHI'01*, ACM Press, Seattle, 2001.
- 11 Paternò, F. (2001). "Towards a UML for Interactive Systems". In *Engineering HCI '01*, Lecture Notes Computer Science N.2254, pp.7-18, Springer Verlag, Toronto, 2001.
- 12 Rumbaugh, J., Jacobson, I., & Booch, G. (1999): *The Unified Modeling Language Reference Manual*, Addison-Wesley, Reading, MA.
- 13 Tarby, J-C and Barthet, M-F (1996). the Diane+ method. In *Proceedings of the 2nd International Workshop on Computer-Aided Design of User Interfaces*. Namur, Belgium, June, 1996.
- 14 van Welie, M., van der Veer, G. C. and Eliens, A. (1998). "EUTERPE - Tool support for analyzing cooperative environments". In *Proceedings of the Ninth European Conference on Cognitive Ergonomics*. Limerick, Ireland