

Toward the Automatic Construction of Task Models from Object-Oriented Diagrams

Shijian Lu[†], Cécile Paris[†] & Keith Vander Linden[‡]

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

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

Abstract: Task models bridge the gap between HCI and Software Engineering. They are useful both for interface design and for generating user interface code and user documentation. These benefits, however, are difficult to achieve because building task models from scratch is difficult. In this paper, we describe an approach for automatically constructing task models from object-oriented diagrams in a CASE tool. The approach exploits the common semantic ground between task models and system-behaviour models, namely use cases, use case diagrams and sequence diagrams. We identify the useful information contained in these diagrams and how it can be augmented to support task model construction. A prototype system is then described, together with a working example.

Key words: Formal models of user interfaces, Task models, information reuse, methodology, tools, Object-Oriented analysis and design

1. INTRODUCTION

Task models are becoming popular in software development. They are employed in the early stages of the software development life-cycle, e.g., in requirements analysis (Sebillotte, 1995) and in design (Hix and Hartson, 1993). They are also used for implementation (Smith and O'Neill, 1996) and evaluation (Card *et al.*, 1983). They can drive prototyping of user interfaces (Johnson *et al.*, 1995) or act as communication tools between all participants in the software design process, i.e., between software engineers, HCI specialists, end users and even technical writers

(O'Neill, 1996, Balbo and Lindley, 1997). Properly formalised, task models can be used to generate user interface code (Szekely *et al.*, 1993, Wilson and Johnson, 1996, Puerta, 1997) and to produce user documentation (Paris and Vander Linden, 1996b).

Given that task models are so useful in software development, the question of how they can be produced becomes important, as building them by hand from scratch can be difficult and time consuming (Paris and Vander Linden, 1996a). It is thus desirable to acquire automatically as much of them as possible from existing information sources.

One possible source of information for task models is Object Oriented (OO) models as used in Computer Aided Software Engineering (CASE) tools. While CASE tools have traditionally been used to assist in software design, implementation and validation, the system behaviour models they contain can also be reused to build the task models required for other purposes, such as the generation of user interface code and user documentation. We have built a task model acquisition module (TMAM) to automate this process. A typical usage scenario for this type of software development environment is as follows:

1. Designers build OO diagrams during system analysis and design. They could also build them as part of reverse engineering or system re-engineering;
2. Designers use the TMAM to automatically construct task models from the OO diagrams;
3. Interface designers and writers then load the task models into a task model editor, modify them as appropriate, and use them to generate prototypes of the user interface and drafts of the user documentation.







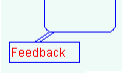



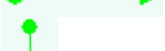

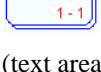
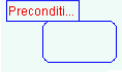

In addition to speeding up the software development process, the automatic acquisition of task models has other benefits. First, if the user documentation and on-line help are generated automatically from the acquired task models, they are guaranteed to be consistent with the underlying application they document, even if the functionality of the application changes during the development process. The task of producing the documentation in a timely fashion and yet ensuring its consistency with the application's functionality is otherwise a difficult task. Second, the user interface designed from the task models will be easier to integrate with the rest of the application modules because the task models were constructed with reference to the actual elements of the application design.

This paper presents a practical approach for augmenting information contained in OO diagrams in order to construct coherent task models. Although this approach is generally useful for interface design, it is directly motivated by reusing design information for the generation of user or task-oriented documentation. Throughout the paper, we illustrate our points with a sample application, STE (Simple Text Editor). STE is a simple freeware text editor that we reverse-engineered.

The paper is structured as follows. Section 2 gives a brief account of Diane+, the formal representation of task models we have chosen in our work. It is followed, in section 3, by an analysis of the different diagrams offered by the Unified Modeling

language (UML) as supported by Rational Rose™ version 4.0, the tool we employ.¹ Use cases, use case diagrams and sequence diagrams are identified as primary information sources for constructing task models. Sections 4 and 5 give a detailed comparison of task models with use cases and with scenario diagrams respectively. The prototype system is described in section 6 together with a working example.

Table 1. Some of Diane+ task attributes and their graphical representation

Task Attribute	Graphical form	Explanation
Interactive		Task in which user interacts with system
Manual		Task performed solely by user
Automatic		Task performed solely by system
Elementary		Task without a decomposition (no shading)
Composite		Task with a decomposition (shaded box)
Decomposition		Task refinement
Feed back		Feedback provided to the user by the application (e.g., message printed)
Mandatory		Task must be performed (box in solid lines)
Optional		Task is optional (box in dotted lines)
Parallelism		Tasks can be performed in parallel
Task sequence		Order of task to be performed
Sequence precondition.		Condition under which the link is to be followed
Name		Task name
Comment	(text area)	Task Comment
Task precondition		Condition that must be true for the task to be applicable
Terminal node		Normal end of the task

¹ Note that the diagrams offered by this software are fairly typical of the diagrams used in Object-Oriented CASE tools.

2. DIANE+ NOTATION

We chose Diane+ (Tarby and Barthet, 1996) as our task model formalism. It is a typical task model formalism in that it 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 interested reader is referred to (Paris *et al.*, 1997) for a more detailed account of the motivation for this choice.) Diane+ employs a graphical notation to represent task decomposition as well as temporal and logical relationships amongst the tasks. In a Diane+ diagram, tasks are represented by boxes which contain the name of the task and, when appropriate, the constraints on the number of times the task can be executed. The shape of the box represents the actor of the task, i.e., whether it is the end user, the system, or combination of both. Table 1 lists the attributes provided by Diane+ that we use in our work.²

3. WHICH UML DIAGRAM?

Task models as defined in HCI provide a user-oriented view of a system. Object-oriented modelling languages, on the other hand, tend to offer two system-oriented views, one describing the application's structure and the other the application's behaviour. UML (UML, 1997), the OO modelling language we have used in this project, is no exception. It models the application's structure with class diagrams and the application's behaviour with use cases, use case diagrams, state transition and interaction diagrams.³ Task models and OO models of the application's behaviour can be seen as semantically equivalent; with the former representing an overt, user-oriented view of application's behaviour, and the latter representing an internal, system-oriented view of that behaviour. In essence, thus, they are *both* concerned with dynamic behaviour of the application. Therefore, in our attempt to automatically generate task models, we focus our investigation on the models of the application's behaviour provided by a CASE tool, i.e., use cases and use case diagrams, interaction diagrams, and state transition diagrams. This section briefly describes each in turn. The next two sections then discuss how they can be exploited to construct task models automatically.

² We added the notion of *feedback* to the original Diane+ formalism, as this concept is an important one for documentation.

³ As in any rapidly evolving field, people often use different terms to refer to the same thing, or use the same term to refer to different concepts. For example, Booch's (1993) object diagrams and interaction diagrams are equivalent to collaboration diagrams and sequence diagrams respectively in UML (1997). However, in UML, object diagrams comprise static and dynamic forms, while interaction diagrams or scenario diagrams include both collaboration diagrams and sequence diagrams. In this paper, we use UML's terminology.

3.1 Use cases and use case diagrams

Use cases are created during object-oriented requirements analysis. Each identifies a thread of potential use for the system to be constructed (Pressman, 1997). Use case diagrams show the relationship between users and the use cases within an application. Together, use cases and use case diagrams identify the agents that will use the application and the high-level goals that these agents have with respect to the application. They are thus useful to describe the tasks a user will perform with the application.

3.2 Interaction diagrams

Interaction diagrams describe how objects collaborate to carry out the activities of a use case. They specify the sequence of messages that are passed between the application objects, some of which are user perceivable objects (e.g., interface widgets) and others are non-perceivable objects (e.g., internal application objects). Interaction diagrams are useful for constructing task models in that they indicate what actions the user performs, and what the system does in response to those actions.

Interaction diagrams come in two forms based on the same underlying information, but each emphasising on a particular aspect of it. *Sequence diagrams* show interactions arranged in time sequence, i.e., the messages that objects exchange are arranged in time sequence. *Collaboration diagrams* show interactions organised around the objects, and relationships amongst objects. Since these two forms of interaction diagrams can be automatically generated from each other, either can be used. We have arbitrarily used sequence diagrams in this study.

3.3 State (transition) diagrams

In UML, state diagrams can be used to show the sequence of states that either a single object, or perhaps the entire system, goes through during its life in response to received stimuli, together with its responses and actions. State diagrams for single objects are of limited use because an end user task is normally achieved through interactions between multiple objects. State diagrams for the entire system, on the other hand, would be a good source of information for task models. When fully developed, a state diagram for a system would contain exhaustive information about the application's behaviour in response to the user's actions. However, this comprehensive state diagram tends to be too complicated to build, and is, therefore, seldom used in practice. For these reasons, we have excluded state diagrams from further investigation, until such time as they are used more fully in practice.

4. USE CASES AND USE CASE DIAGRAMS VS TASK MODELS

As mentioned earlier, use cases capture the user requirements for an application by describing how and to what ends an application will be used (Jacobson *et al.*, 1995). Semantically, they also define abstract or composite tasks. Consider the example shown in Figure 1. It provides a simplified use case for the composite task of saving a file in STE, the application we employed for our test-bed. This use case gives a clear description of how the user interacts with the system to accomplish the task of saving a file. Task models expect these composite tasks to have explicitly defined attributes and to be hierarchically decomposable. This section discusses UML's support for these two things.

4.1 Composite task attributes

Although use cases and composite tasks are equivalent semantically, they are not equal. For instance, composite tasks in a task model, such as a Diane+ model, have task attributes explicitly defined (refer to Table 1). This is not the case for use cases. While some task attributes may be explicitly defined, others are defined implicitly and some may be of no concern at all:

- *Explicitly defined attributes* – In UML, the name and textual description are explicitly represented for a use case;
- *Implicitly defined attributes* – Although not explicitly defined, one can determine whether the task is manual, automatic, or interactive (see table 1) based on the actor specified for the task;
- *Unrepresented attributes* - Some information required to construct task models is difficult if not impossible to extract automatically from a UML use case. Examples include task preconditions and feedback. For example, the last statement in the Figure 1 is an expression of task feedback. Because UML has no explicit representation for task feedback, such information is usually included in the use case description. Extracting this information from free text would require sophisticated language processing.⁴ The same is true of preconditions in UML.

- | |
|---|
| <ol style="list-style-type: none"> 1) The user chooses the save option from file menu. 2) The system checks the document status. 3) If it is an existing file, the system saves the file. 4) If it is a new file, the Save File dialog appears. 5) The user chooses a folder. 6) The user enters a file name. 7) The user clicks the Save button. 8) The system saves the file. 9) When saving, the saving progress bar can be observed by the user. |
|---|

Figure 1. A simplified use case: Save file.

⁴ While it would not be possible to process free text, we are investigating the use of a *controlled language* for these descriptions, from which it is then possible to obtain information, because of the regularity it imposes on the text.

4.2 Hierarchical decomposition

In task modelling languages, composite tasks can be hierarchically decomposed. This is also true in UML. In Rose's implementation of UML, a use case is elaborated by attaching either a subordinate use case diagram (if it is relatively complex) or a sequence diagram (if it is relatively simple). In UML version 1.0, apart from communicate relationships between an actor and a use case, there are two types of elaboration relationships between use cases: *extend* and *use*. The extend relationship is used to express conditional behaviour while the use relationship is used to describe common behaviour between two or more use cases. For example, a possible use case diagram for Figure 1 would be something like Figure 2. There are 3 use cases in this diagram, **Start saving**, **Execute saving**, and **Specify file information**. **Start saving** will typically cover the behaviour described by (1) and (2) in Figure 1 while **Execute saving** from (8) to (9) and **Specify file information** from (4) to (7). The relationships between **Start saving** and **Execute saving**, **Specify file information** are extend relationships. That is, an instance of **Start saving** may include the behaviour specified by use case **Execute saving** or **Specify file information** depending on whether the file being saved is an existing or a new file. **Specify file information** is related to **Execute saving** by a use relationship. In other words, the behaviour described in **Execute saving** is mandatory to **Specify file information**.

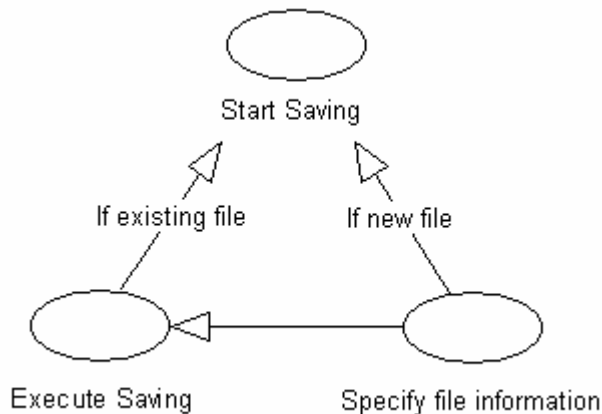


Figure 2. A use case diagram that elaborates Save file use case.

Based on the discussion above, we conclude that the nesting between use cases and use case diagrams is equivalent to task hierarchical decomposition. In particular, we see the following:

- All use cases inside a use case diagram can be seen as sub-tasks of the composite task corresponding to the use case diagram;

- Extend relationships (with associated preconditions) between use cases can be seen as conditional task sequence relationships, from the extended task to the extending task;
- Use relationships between use cases can be seen as task sequence relationships, from the using task to the used task;
- Unrelated use cases can be seen as tasks parallel to each other.

Given these similarities, we can automatically convert use cases and use case diagrams into hierarchically decomposed task models. The task attributes that cannot be determined automatically have to be added manually as appropriate.

5. INTERACTION DIAGRAMS VERSUS TASK MODELS

A use case is semantically equivalent to a composite task, and sequence diagrams are used to elaborate use cases. What is the relationship between sequence diagrams and task models? To answer this question, we need to explore how much information contained in sequence diagrams could be reused to generate task models. To help focus our comparison between these two formalisms, we have specified the behaviour of STE, our simple text editor, using both a sequence diagram, Figure 3, and a Diane+ task model, Figure 4.

The sequence diagram in Figure 3 was obtained by reverse engineering the STE application, by experimenting with it, and by referring to the system code. The diagram in Figure 3 describes a scenario in which the application contains a newly created document, and the user saves the document before quitting. As we can see, the selection of the “Quit” option leads to a series of messages being passed to a number of objects, including:

- When the user selects the “Quit” option, the menu object sends the “quit()” message to the application’s main object, “instance STE”.
- The “instance STE” object sends a “close()” message to “myDoc”, the main document object.
- The object “myDoc” checks whether it has been saved. (event 4)
- If “myDoc” has not been saved, it sends a message to open the “toSave” dialog box. This “if” condition is specified textually in the comment column on the left.
- The user presses the “OK” button. (event 11)
- The system determines that this is a new document, so it calls the “Save File Dialog”.
- The user selects a folder, enters a file name, and clicks the “OK” button.
- The system saves the document and exits the application.

We also built the corresponding Diane+ task model for the exit task, based on our experience using the application. This task model is presented in Figure 4. As shown there, “Quit STE” is an interactive composite task that is decomposed in a sequence

of elementary tasks. The interactive task “select quit option” is followed by an automatic task “show toSave dialog”, and so on. By comparing Figures 3 and 4, it becomes clear that sequence diagrams contain much more system-internal information than do their corresponding task models, and that sequence diagrams do not allow for the specification of all task attributes present in task models.

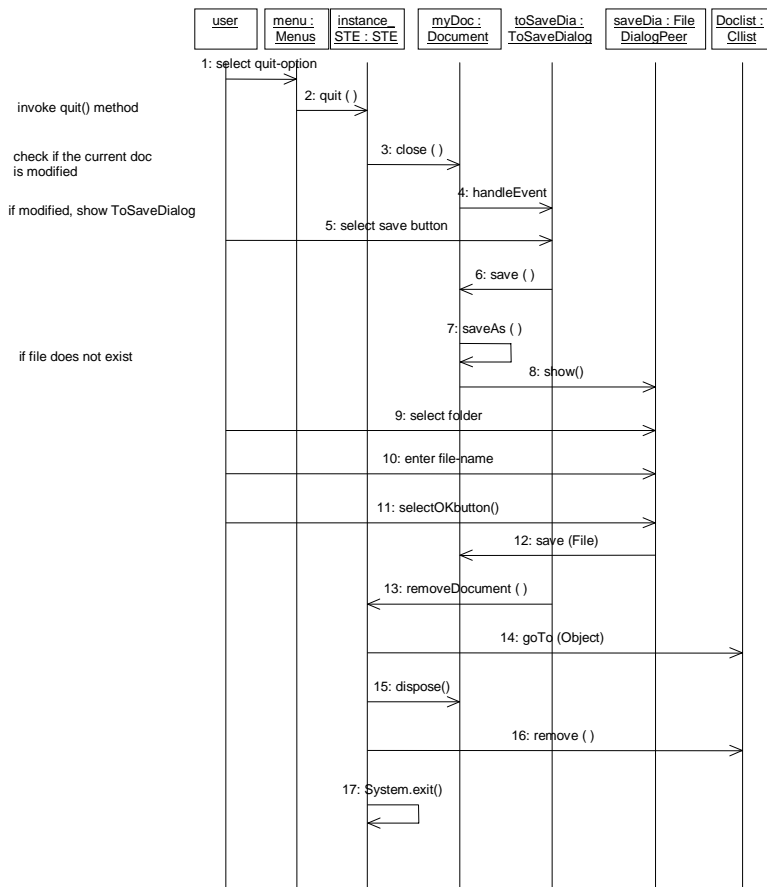


Figure 3. Sequence Diagram: Quit STE (Simple Text Editor).

5.1 Overt and Internal Behaviour

Sequence diagrams focus on system-internal, inter-object message activity. In Figure 3, for example, we can see that, when the user selects the quit button, the menu item instance sends a “quit()” message to the “instance STE” object. That object in turn sends a “close()” message to the document instance. These system details are largely invisible to the end user. This focus on internal activities is not

surprising. Sequence diagrams are intended to facilitate system design, and therefore focus on how objects, whether visible or invisible, collaborate with each other to deliver the required functionality. Task models, on the other hand, are intended to describe how users interact with a system in performing certain tasks. They must, therefore, focus more on overt behaviour.

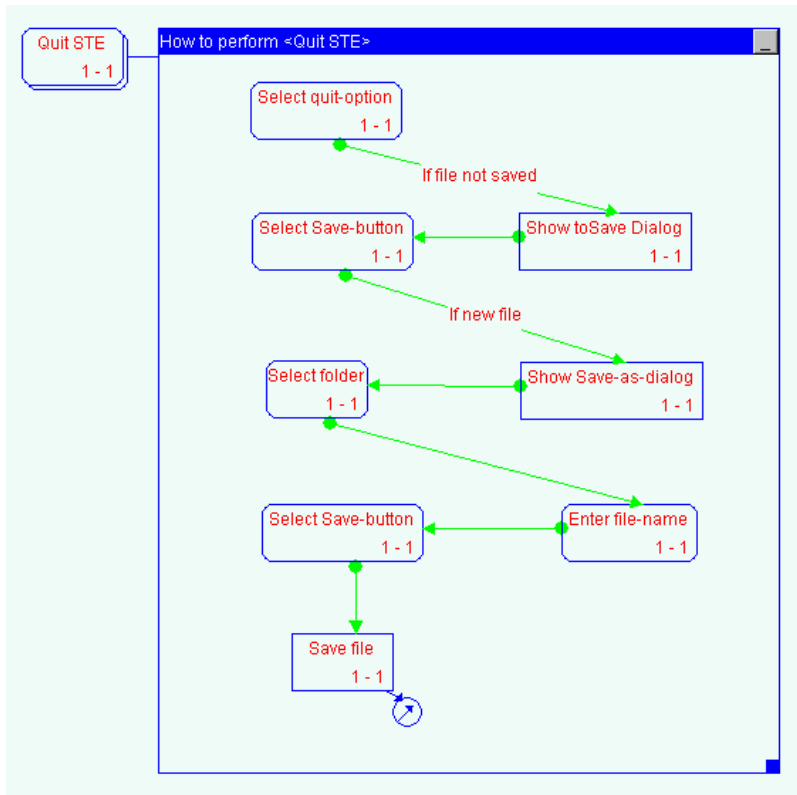


Figure 4. Diane+ model for quitting STE.

We should note that task models do capture some system actions. For example, message 4 in the sequence diagram roughly corresponds to the sub-task “[system]show toSave dialog box” in the task model. Similarly, message 8 corresponds to the action “[system] show Save-as dialog box”, and, finally, message 12 to the action “[system] save file”. A closer look at messages 4 and 8 reveals that they both display interface elements to the user. Message 12 saves the file, which is also discernable by the user.

Based on the above analysis and other examples, we conclude that sequence diagrams are a superset of task models in that task models mainly describe the overt behaviour of the users and the system, while sequence diagrams additionally represent the system’s internal behaviour.

5.2 Non-Composite Task Attributes

A second difference between sequence diagrams and task models is that messages in sequence diagrams do not explicitly specify all the task attributes present in task models. We made a similar observation concerning use cases in section 4.1:

- *Explicitly defined attributes* – Many task attributes are explicitly specified in the sequence diagram. For example, the temporal relationships amongst tasks;
- *Inferred attributes* – Some task attributes can be inferred from sequence diagrams. For example, whether a task is interactive or automatic can be inferred from whether the sender of the message is the user or not;
- *Unrepresented attributes* – Some task attributes are not captured in sequence diagrams. For example, sequence diagrams do not specify multiple paths of execution, task repetition, or whether the action is optional or required. Textual notes may be placed in the left column to indicate these attributes, but these texts, at least in general, are too unstructured to be converted easily into task attributes.

5.3 Summary

In summary, then, we have the following relationships between sequence diagrams and task models:

- sequence diagrams are roughly equivalent to composite tasks in a task model;
- messages between objects in a sequence diagram are roughly equivalent to elementary tasks;
- all messages inside a sequence diagram are roughly equivalent to sub-tasks of the composite task corresponding to the sequence diagram.

Given these similarities, we can design an algorithm to construct automatically a task model from a sequence diagram. Generally, the sequence diagram messages can be converted to task model tasks of the appropriate types. However, it is clear that the sequence diagram contains a considerable amount of system-internal information that is not needed in the task model. To address this problem, we use a number of heuristics that filter out irrelevant messages:

- keep all user-initiated messages;
- if the first message is not a user-initiated message, then keep the message immediately before the first user-initiated message;
- keep the message immediately following the last user-initiated message, if there is one;

- if the message immediately following the last user message is not the last the message in the sequence diagrams, then keep the last message as well.

6. THE SYSTEM AND A WORKING EXAMPLE

We have developed a task model acquisition module (TMAM) which automatically constructs Diane+ task models from UML use cases, use case diagrams, and sequence diagrams. The algorithms employed by this tool are based on the analysis given in the previous sections and are implemented within the Rational Rose CASE tool using Rose script.

The TMAM starts from the top-level use case diagram in an UML system behaviour model. After transforming all use cases and their relationships into partial task models, it will iterate through each use case and transform its subordinate use case diagram(s) or sequence diagram(s).

For STE, the TMAM first creates a use case diagram called “use stePro”. It is shown in Figure 5. This diagram models the fact that a user could perform 5 basic tasks with STE: “create file”, “save file”, “open file”, “print file”, and “close file”. All of these tasks

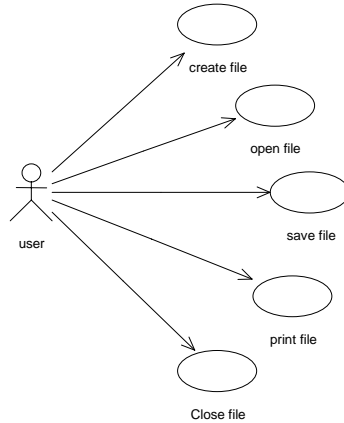


Figure 5. Use case diagram: use stePro.

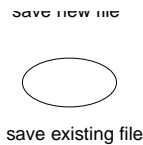


Figure 6. Use case diagram: save file.

are internally complex, but for the sake of simplicity, only the “save file” use case is further elaborated here. Its elaboration, the “save file” use case, is shown in Figure 6. This use case diagram models the fact that the user may save either a new file (“save new file”) or an existing file (“save existing file”). At this point, the designer decides that this is a sufficient level of detail for the use case analysis. He or she then defines the “save new file” sequence diagram, shown in Figure 7. This diagram specifies exactly how the objects in the application will collaborate to deliver the functionality required by the “save new file” use case. Briefly, the

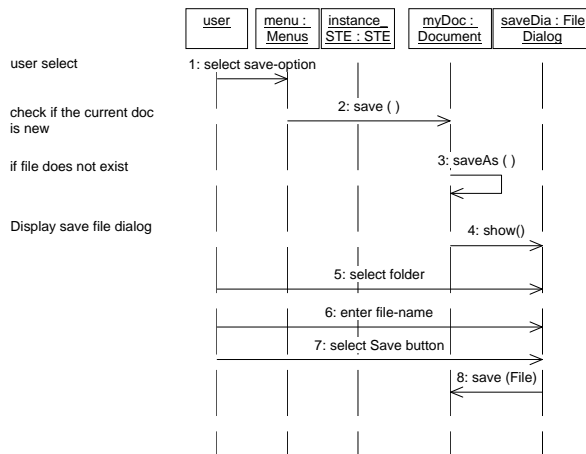


Figure 7. Sequence diagram: save new file.

sequence diagram shows that, when the user chooses **save** option from file menu, a series of internal system actions will eventually bring up a “**save file**” dialog box. At this point, the user must select the desired folder, enter a file name, and click on the “**save**” button. This results in the application actually performing the action of saving the file. A similar sequence diagram would be created for the “**save existing file**” use case.

Given the UML behavioural model just created, the TMAM automatically constructs the corresponding task model, as shown in Figure 8.⁵ This model is displayed using a task model editor (TAMOT), which we built in JAVA. In Figure 8, composite task “**use stePro**” is decomposed into 5 parallel composite sub-tasks, one of which, “**save file**”, is further decomposed into “**save new file**” and “**save existing file**”. The “**save new file**” task is decomposed into a sequence of elementary tasks which correspond to the “**save new file**” sequence diagram. Note that messages 2 and 3, modeled in the “**save new file**” sequence diagram, have been filtered out using the heuristics presented in the previous section.

Although the task model shown in Figure 8 is accurate and useful, it is seldom the case that automatically constructed task models can be used without modification. This is the motivation for building the dedicated editor, the TAMOT. It allows one to modify various aspects of the task model. Typically, changes will include:

⁵ The layout has been modified. The TMAM does not yet produce a totally acceptable layout for the task model derived automatically. This is due to the fact that coordinates have to be added to the task model on the fly. We are currently working on obtaining a reasonable layout automatically.

- *The names of elementary tasks* - The displayed name of an elementary task depends on whether its corresponding message is the user or not. If it is the user, the name of the message is taken as the task name (e.g., “select save-option” and “enter file name”). Otherwise, the name of the message together with the name of the receiving object’s class name is used (e.g.,

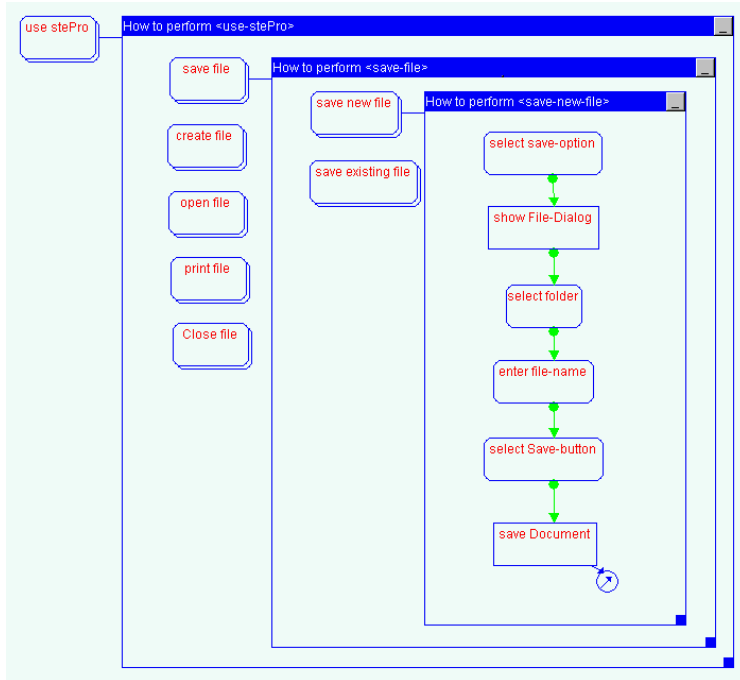


Figure 8. Resulting task model (displayed by TAMOT).

“show File Dialog” and “save Document”). While the names are acceptable in this example, it is possible that inaccurate names may be constructed. TAMOT allows the user to modify them manually.

- *The number of elementary tasks* - The number of elementary tasks that get created in the task model is determined by the filter heuristics described in Section 5. The filtering rules worked nicely for this example, but their effectiveness in a large example is yet to be tested. It is likely that in some cases the user will have to add or remove elementary tasks.
- *Task preconditions and feedback* - As discussed earlier, it is difficult to automatically derive task preconditions and feedback from the textual comments in the UML model. The user, therefore, will need to add them manually by referring to the ancestral use case description.

7. SUMMARY AND CONCLUSIONS

Task models are increasingly used for a variety of purposes, but constructing them remains a difficult task. In this paper, we presented a practical way of automatically acquiring task models from system behaviour models defined by object-oriented diagrams. In particular, we focussed on UML use cases, use case diagrams, and sequence diagrams. By successfully exploiting the common semantic ground covered by system behaviour models and task models, we showed how information contained in these diagrams could be augmented and filtered to construct task models. We then presented a working prototype, demonstrating that substantial portions of a task model can be obtained automatically.

Of course, it is not clear that building UML behavioural models is any easier than building Diane+ task models. However, it is clear that OO CASE tools are gaining widespread acceptance. This means that we will be able to take advantage of pre-existing knowledge sources by reusing the information they contain in order to cut the effort required to build task models.

In addition, our approach brings the software engineering and interface design communities a step closer together. This is important as the two communities need to work together to produce software systems that are correct and useful. Furthermore, the approach may also allow for the documentation production to be better integrated into the software development life cycle, as task models serve as a basis to produce documentation.

In our work, we have successfully used our derived task models to generate on-line user documentation (Paris *et al.*, 1998). In future work, we intend to apply our technique in a “real world” setting. This will generate the empirical data that we need to improve our system to the point where it is truly practical and deployable. We also intend to investigate the use of the task models for generating user interface code. Finally, we are exploring the possibility of building a unified model that integrates task models and OO diagrams (Lu *et al.*, 1998). This unified model would further tighten the relationship between human computer interaction and software engineering.

ACKNOWLEDGMENTS

This work is partially supported by the Office of Naval Research (ONR) – Grant N00014096-1-0465 – in the program for User Centred Direct Interaction Systems. We gratefully acknowledge the participation of Sandrine Balbo and Nadine Ozkan, and are thankful to Valery Anciaux and Christophe Plier for their contribution to the implementation of the task model editor.

REFERENCES

- (Balbo and Lindley, 1997) Balbo, S. and Lindley, C. Adaptation of a task analysis methodology to the design of a decision support system. In *Proceedings of Interact'97*, Sydney, Australia, 1997.
- (Booch, 1993) Booch, G. *Object-oriented Analysis and design with applications*. Benjamin Cummings, Redwood City, 2nd edition, 1993.
- (Card *et al.*, 1983) Card, S.K., Moran, T.P. and Newell, A. *The psychology of human computer interaction*. Lawrence Erlbaum Associations, 1993.
- (Hix and Hartson, 1993) Hix, D. and Hartson, R. *Developing user interfaces, ensuring usability through product and process*. Wiley, 1993.
- (Jacobson *et al.*, 1995) Jacobson, I, Christerson, M, Jonsson, P. and Overgard, G. *Object oriented software engineering: a use case driven approach*. Menlo Park, California, Addison-Wesley, 1995.
- (Johnson *et al.*, 1995) Johnson, P. Johnson, H. and Wilson, S. Rapid prototyping of user interfaces driven by task models. In *Scenario based design: envisioning work and technology in system development* (ed. J.M Carroll), John Wiley, New York, 1995.
- (Lu *et al.*, 1998) Lu, S., Paris C. and Vander Linden K. Integrating task modelling into the object oriented design process: a pragmatic approach. Position paper at CHI'98 workshop on Incorporating Work, Processes and Task Analysis into Industrial Object-Oriented Systems Design. April, 1998.
- (O'Neill, 1996) O'Neill, E.J. Task model support for cooperative analysis. In *Proceedings of CHI'96*, 1996.
- (Paris and Vander Linden, 1996a) Paris, C. and Vander Linden K. An overview of on-line documentation and CASE tool: Isolde, Report on task 1 and 3. Technical report ITRI-95-16, ITRI, University of Brighton, UK, 1996.
- (Paris and Vander Linden, 1996b) Paris, C. and Vander Linden K "DRAFTER: An interactive support tool for writing multilingual instructions", *IEEE Computer*, 29(7):49-56, 1996.
- (Paris *et al.*, 1997) Paris, C., Balbo, S. and Ozkan, N. Novel uses of task models: two case studies. Presented at the *NATO/ONR workshop on cognitive tasks*, CSIRO Technical Report, 1997.
- (Paris *et al.*, 1998) Paris, C., Vander Linden, K. and Lu, S.: Automatic Document Creation from Software Specifications. In the *Proceedings of the 3rd Australian Document Computing Symposium (ADCS'98)*, J. Kay and M. Milosavljevic (eds), Sydney, August 1998.
- (Pressman, 1997) Pressman, R.S. (1997) *Software Engineering: A Practitioner's Approach*. International Edition, McGraw-Hill.
- (Puerta, 1997). Puerta, A.R A model-based interface development environment. *IEEE Software*, July/August, 1997.
- (Sebillotte, 1995) Sebillotte, S. Methodology guide to task analysis with the goal of extracting relevant characteristics for interfaces. Technical report for Esprit project P6593, INRIA, Rocquencourt, France, 1995.
- (Smith and O'Neill, 1996) Smith, M.J and O'Neill, J.E. Beyond task analysis: Exploiting task models in application implementation. In *Proceedings of CHI'96*. (1996)

- (Szekely *et al.*, 1993) Szekely, P. Luo, P. and Neches, R. Beyond Interface builders: Model-based interface tools. In *Proceedings of InterCHI'93*, ACM, 1993.
- (Tarby and Barthelet, 1996) Tarby, J-C and Barthelet, M-F (1996) the Diane+ method. In *Proceedings of the second international workshop on computer-aided design of user interfaces*. Namur, Belgium, June, 1996.
- (UML, 1997) UML Notation Guide: unified modelling language version 1.0, Rational Software corporation, 1997.
- (Wilson and Johnson, 1996) Wilson, S. and Johnson, P. Bridging the generation gap: from work tasks to user interface design. In *Proceedings of CADUI'96*, 1996.

BIOGRAPHY

Dr Shijian Lu is currently a senior research scientist in CSIRO Mathematical and Information Sciences, Australia. Before joining CSIRO in 1995, he was a research fellow in the Center for Cognitive Science, Denmark. He holds a BS degree from Shandong Institute of Mining and Technology, China and PhD degree from Leeds University, UK. His research interests include many aspects of HCI such as information and knowledge representation, multimedia multimodal interface design, and incorporating HCI into object oriented software development process.

Dr Cécile Paris is the research leader of the Intelligent Interactive Technology Project (IIT) at CSIRO, the Australian National Research Center. This research group is part of the Mathematical and Information Sciences Division, and is concerned with a variety of issues dealing with human-computer interaction (HCI). Dr Paris' research is primarily in language research engineering, but also involves multimedia presentation, user modeling, user interface design and evaluation, authoring tools, integration of HCI into software engineering, and software internationalization and localization. Dr Paris received a BA in computer science from the University of California in Berkeley, and an MS and a PhD in computer science from Columbia University (New York).

Dr Keith Vander Linden is a faculty member in the department of Computer Science at Calvin College, Grand Rapids, MI, USA. He received an MS in Computer Science from the University of Iowa and a PhD in Computer Science and Cognitive Science from the University of Colorado. His research interests include Artificial Intelligence, Cognitive Science, Software Engineering and User Interface Design.

Discussion

Gilbert Cockton: Do you really think use cases represent the user context?

Shijian Lu: Yes, I do. Use cases do capture SOME aspects of user context. This is one of the main reasons why use cases are popular in the OO community. Having said that, however, I, by no means claim that use cases capture ALL aspects of user context.

Fabio Paterno: So why do you use cases rather than task specifications?

Shijian Lu: Because they are mainstream within OO software engineering

Fabio Paterno: But they do not address good design, they do not start with a good set of task descriptions

Shijian Lu: Yes/no, but I think there are two different issues here. The first issue is that given that use cases are widely used in OO community, is there anyway in which use cases as defined in the design specification can be put into uses other than motivating system design? And that is the concern of this paper. The second issue is that use cases as a mechanism for capturing requirements have aforementioned short-comings, how can we do better in the future? The latter was addressed at the CHI'98 workshop on incorporating work, process and task analysis into commercial and industrial Object-Oriented systems development, where suitable extensions to UML were proposed.

Fabio Paterno: So would you try to base use cases on task modelling?

Shijian Lu: Ideally, in my view, it would be better to use task analysis and modelling to replace use cases altogether. But, the pragmatic may lead to different courses.

Remi Bastide: Wouldn't that put ergonomists out of a job? They currently begin the software lifecycle with task modelling. Their results are then used by engineers. Your implicit lifecycle is different

Shijian Lu: No, typically, what gets constructed would be a draft task model. Therefore, ergonomists are still needed for fleshing out the model. You are right in that, the lifecycle here is different. We are concerned with re-using information contained in an existing design specification.

Joelle Cockton: One thing to remember in system design is that the initial task model will not match the final task model for the user once the system has been developed.

Len Bass: The question of whether use cases or task models should be done first is a marketing question, not a technical question. The fact is that use case modelling is the only modelling technique that is used by software developers, and leveraging off this fact may increase the usage of task models

Morton Borup Harning: Given that task models that your system generate are not sufficient when seen from a user's point of view, could you edit the generated task models and then feed these changes back into the original use cases, improving or at least pointing out its shortcomings

Shijian Lu: That's a good idea, that we need to explore.

