Empowering UML Application Design With Task Models

Ha Minh Lam
Oxford University Clinical Research Unit
Ho Chi Minh City, Vietnam
minhlam.ha@gmail.com

Tanguy Wettengel
University of Limoges
Limoges, France
tanguy@teamtim.com

Huynh Trung Thong
The University of Melbourne
Melbourne, Australia
thong.huynh29@gmail.com

Abstract –**This paper describes a practical method to include task modelling into Unified Modeling Language-based (UML-based) software engineering. The reason for this undertaking is increasing awareness of the capability and handiness of task models to picture the functionalities of a system (as awaited by customers), but also the lack of expressive power of UML use case diagrams, when underspecified, to make the functionalities of a system fully understandable to software developers. The described method builds on a task modelling software, a plugin transforming task models into use case diagrams and use case descriptions, and a UML software enabling to display use case diagrams. The plugin puts a set of rules interpreting the content of task models at work, in order to generate an XML file readable by the UML software, and a ready-to-use PDF output containing the use case descriptions. The advantages of this workflow are to enforce the link between the different project stakeholders (customer, architect, developers), to allow for cost-free iterations affecting the requirement scheme, to enable usable incremental modelling and to introduce interface design into the first step of the design process.**

I. Introduction: Are Use Cases Enough Specified To Ground Application Design?

UML use cases drive the system's requirement elicitation process in many well-known software engineering methodologies [1]. The main purpose of use cases is to capture the functional requirements of a system [2], although not to state the procedure of execution. Because of their simple graphical notation and accessible natural language enhancements, they are granted to enable communication with customers. However, industrial practitioners tend to call their generality and simplicity into question [3] [4] [5].

According to Hamilton and Miles [6], "A use case diagram might be a good leverage, but the information [it provides] is too inadequate for designers to understand completely how the functional requirements should be met, who the most important actor is, and what steps are involved in the use case". To overcome this drawback, the coupling of task modelling and use case diagramming has sometimes been suggested [4].

Within task modelling, the steps contributing to goal achievement are considered from the user's perspective. Tasks may unfold into subtasks or actions within a hierarchical structure, and may be associated with inputs, outputs, pre- and post-conditions, performance values, time of execution prediction, etc.

Task modelling is increasingly being considered as fundamental in the interactive application development process. While some years ago, forerunners already thought of methods to combine task modelling with UML representations [8], building on the fact that "Software engineers are focused on the tasks a user has to perform, on the object he has to manipulate and on the work situation" [7], the idea of promoting task models as a critical component for application design is currently gaining growing attention [9].

Nevertheless, Sinnig [4] notes that "The industry and the mainstream developers have not adopted task models as a key method to elicit user requirements". According to Paris [10], the main reason for this situation is the complexity of task models, which are considered to be hard to build and maintain.

This paper introduces a methodology for integrating light and easy-to-maintain task structures and use case engineering. In our proposal, task and use case modelling are merged into a single design step. The reason for this is that we allow use case diagrams to be automatically generated from a task model, which thus appears as the initial formal item within the requirement documentation.

To illustrate the automated production of use case diagrams and the generation of use case descriptions, we here use a free task modelling software (TimBox®), enhanced with the TASK2UML plugin, and Enterprise Architect®, for the graphical display of the derived use cases. Use case descriptions are directly generated in PDF format by the plugin.

II. Use Case Diagrams' Underspecification

Object Management Group (OMG)'s UML [2] posits that use cases capture the requirements of a system, "that is, what a system is supposed to do." We understand from this that use case diagrams are meant to capture the *functional* requirements of a system. To enable a more efficient fulfilment of this goal, upgrading the expressive power of diagrams and the related use case descriptions can be considered. We next underline some limitations of use case diagrams (solutions to overcome them, both applying to the diagram and the use case descriptions, will be presented in Section IV):

- Despite the fact that system designers tend to use multi-level structures when diagramming use cases, this is not mandatory as per the standard. Without this facility, the only information we get from a use case diagram is a list of functions the system should enable users to engage.

- The list of system usages can only show functions of the system but not the order (if any) in which these functions are put at work. This can be vital in many situations.
- There are situations where a functionality comprising sub-steps is selected for execution by the user or the system. The firing of the function itself or of a step leading to goal fulfilment might be subjected to some special system configuration. A use case diagram has no capability to describe the steps inside a use case nor to define the conditions under which particular functions or steps can be engaged.
- In software applications, using alternative ways to accomplish the same goal is common. For example, to play in cooperation mode in any Xbox game, either the System Link or the Split-Screen option can be chosen. A use case diagram has no capacity to express this fact. Developers with no background in video games may find it difficult to realize that 'Play via system link' and 'Play via Xbox live' are 2 alternatives for playing in cooperation. This can finally lead to a deficient user interface design.
- Functions subjected to necessary or possible iterations ("cycles") in order to allow goal achievement, must execute (or be engaged) over and over until the configuration described in a stopping condition is met. Accessing a file, for instance, may need successive opening of embedded folders. Since UML cannot express the sequence of use cases, the diagram proves unable to explicitly inform developers about this particular situation.

Because of the facts underlined above, we feel that OMG's UML does not allow to capture the requirements of a project fully, at least with regard to an integrated view of a system's functions. Moreover, complementary views (such as activity or sequence diagrams, yet closely related to use cases), fail to fill the gap.

III. Task Models

The rule system allowing to generate UML use cases and use case descriptions out of task structures (see Section IV), is based on a specific modelling language (TIM, Task-oriented Information Modelling, as implemented in TimBox® [11]). Nevertheless, it can be easily adapted to other methods, provided the meta-model they put at work proves rich enough to capture task and task hierarchy semantics. Fig. 1 illustrates a simple task viewed from the TIM perspective, and worded in natural language.

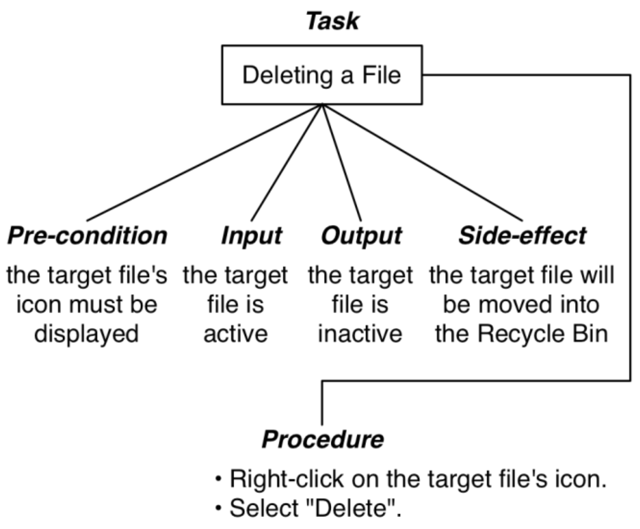
Task
Deleting a File
Pre-condition
the target file's icon must be displayed
Input
the target file is active
Output
the target file is inactive
Side-effect
the target file will be moved into the Recycle Bin
Procedure
• Right-click on the target file's icon.
• Select "Delete".

Figure 1. The structure of TIM tasks

Some major features of this structure appear in the following list:

- The goal of a task is described by two ordered states of the system, termed "Input" and "Output".
- Any task that may only be engaged under some definable system configuration is considered as "Conditional". The referred system configuration acts as a condition for task engagement and appears in the "Pre-condition" component of the task description.
- Only tasks that do not unfold into subtasks (termed "Last-level Tasks") contain a list of one or more (ordered) actions allowing to reach the goal they describe. This list is known as "Procedure". The activity allowing to reach the goal other tasks is distributed into its subtasks.
- Any last-level task may have more than one procedure allowing to reach its goal.
- Any task that needs to be carried out more than once in order to allow a superordinate goal fulfillment is considered as "Cyclic" ("Undoing a bolt", for example, within the process of unmounting a wheel). The referred superordinate goal acts as the cycle-stop condition for task engagement and appears in the "Pre-condition" component of the task description.

In turn, TIM task hierarchies exhibit the following distinguished properties:

- In a task hierarchy, first-level tasks capture the goal of a user interacting with a system ("Saving a file", for example).
- Subtasks are categorized as "Optional", "Non-optional" and "Alternative".
- Alternative subtasks describe different ways of reaching the same goal ("Configuring Lan connection"

and "Configuring Wireless Connection" for the goal "Configuring Internet Connection", pertaining to the task that encompasses both). Tasks unfolding into alternative subtasks are termed "Alternative tasks".

- An Alternative task can be carried out more than once if more than one of the goals appearing in its (alternative) subtasks need to be reached (both "Configuring Wireless Connection" and "Configuring Lan Connection", for instance, in the example above).
- Optional subtasks are those that, if not engaged, do not disrupt the process allowing to reach the goal of their superordinate task ("Changing the format of a file", within the "Saving a file" process). Subtasks that do not conform to this constraint are considered as "non-optional".

IV. Generating Use Case Diagrams And Descriptions From Task Models

As stated in the introduction, there are good reasons to take advantage of task modelling when defining specifications, without disregarding the benefits of use case diagrams. Consequently, we propose a solution to combine use cases with task models, satisfying the following constraints:

- Whenever possible, use case diagrams must adopt multi-level structure.
- Use case descriptions must be compliant with a formal and unambiguous standard template (a proposed template is disclosed in Table 1).

As shown in Fig. 2, our fundamental idea is placing a task modelling step between scenarios (user information records) and use cases in the requirement elicitation. After the task model has been finished, use case diagrams and descriptions can be automatically generated from it.

However, a set of predefined constructs allowing to transform task models into use case diagrams and use case descriptions is needed to this end. After analysing entities and

TABLE 1. Use Case Description Template

Use Case Name	[Name of the use case]
Summary	[Short description of the use case]
Goal	[Goal of the use case]
Actor	[List of all users that interact with this use case]
Pre-conditions	[Specify conditions that must be undertaken by user in order to get the use case initiated] [Specify environmental conditions, which users have nothing to do with]
Cycle stop condition	[If this use case is repeated until meeting a particular stopping condition, this section will specify the stopping condition]
Main flow	[Specify the basic steps to go through]
Alternative flow	[There are usually many ways to accomplish the same goal in software. This section specifies secondary possible flows.]
Side-effect	[Specify an activity or a use case that is executed as a chain-reaction effect]

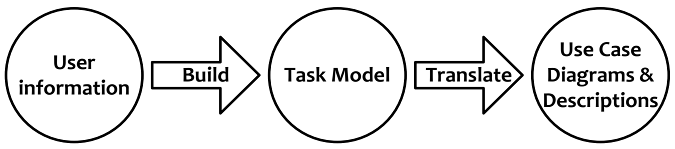

Figure 2. Proposed requirement elicitation process

properties of use case diagrams, use case descriptions and TIM task models, we have identified 11 basic rules to generate use case diagrams and use case descriptions from task models (see Table 2).

Consider the following design example, aimed at engineering an ATM system that delivers 5 functions: “Withdraw Cash”, “Transfer Money”, “Change Pin-code”, “Apply for a New Account” and "View Stock Exchange Prices". As a constraint, we consider that, except for the two last ones, all functions require a secure session to be executed.

The task model appearing in Fig. 3 shows “Print Operation Receipt” as an optional subtask (if it is not engaged, the goal of the superordinate task will still be reached), all other non-alternative subtasks are mandatory with respect to goal satisfaction of the task above them. The execution of any alternative subtask ("Withdraw Cash", "Transfer Money", "Change Pin-code") satisfies the goal of the superordinate alternative task ("Perform Operation"), although this task may be engaged repeatedly if more than one transaction has to be performed (see Section III). The "Perform Operation" task offers 3 alternatives: “Withdraw Cash”, “Transfer Money” and "Change Pin-code" (all of them being considered as

TABLE 2. Rules For Translating Task Models Into Use Cases

	#	**Rules**
For Use case Diagrams	1	First level tasks are translated into base use cases.
	2	If a non-alternative subtask is optional, it is converted into an extending use case.
	3	If a subtask is non-optional, we consider it as an included use case.
	4	A TIM-alternative-task and its subtasks are translated into a generalization-set where the latter are specific classifiers of the former.
For Use Case Descriptions	5	If a task has only 1 action in the “Procedure”, this task is considered as a step in the flow, not a use case.
	6	The “Output” of a task is transferred to the “Goal” of its equivalent use case description.
	7	For last-level tasks, actions are considered to be steps in the flow.
	8	When translating non-last-level tasks, subtasks within them are considered as steps in the flow.
	9	For conditional or cyclic tasks, the “Access” information is transferred to the “Pre-conditions” or “Cycle Stop Condition” of the use case descriptions.
	10	For last-level tasks that have multiple procedures, one of them is taken to be the “Main Flow” and the others are considered as “Alternative Flows” of the corresponding use case descriptions.
	11	If a task has an output that is the access of some other one, it will be considered a non-alternative-subtask of the latter when applying the remaining rules.

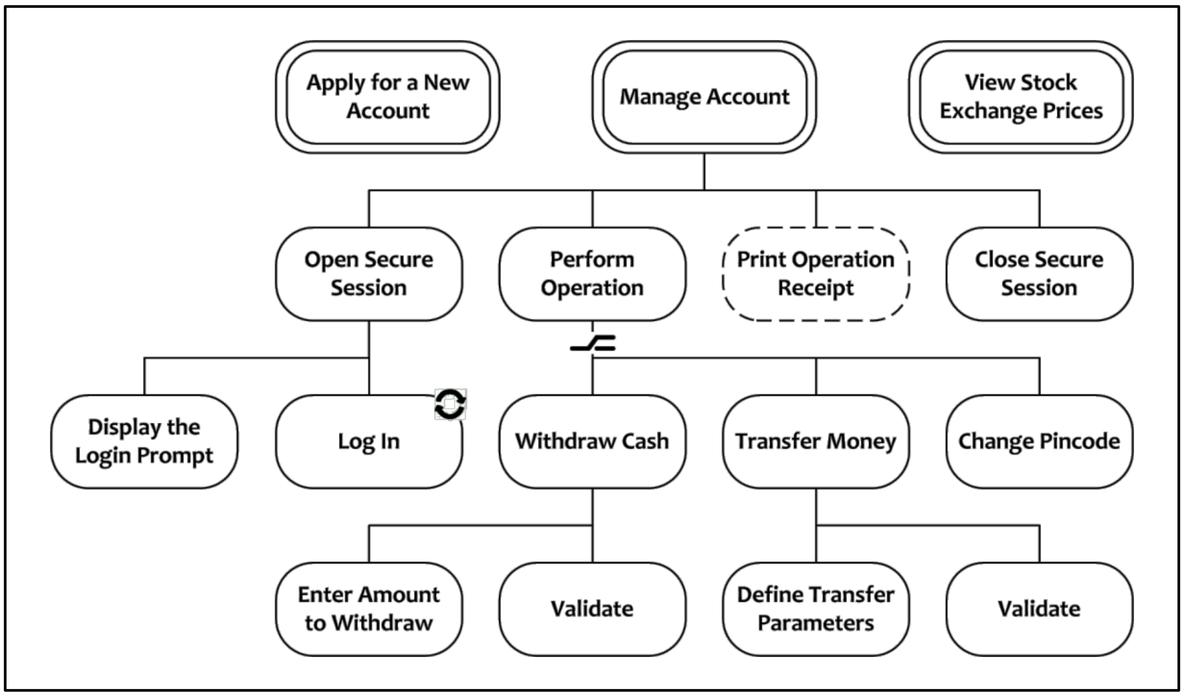

Figure 3. Task model for ATM system

operations). The only cyclic task in this model is "Log in", because the user might have to enter his pin-code repeatedly (if an error occurs) for it to be recognized by the system.

Once the task model stands, a set of rules will be applied on it to generate a corresponding use case diagram and a description for each use case. But, even before this process starts, the task model might be re-arranged by rules specific to the conversion of task models into use case diagrams: in our case, rule 11overrides the canonical structure of alternative tasks, by making of "Open Secure Session" a non-alternative subtask of the "Perform Operation". This occurs because the access of "Perform Operation" is the output of "Open Secure Session".

In Fig. 4, the diagram illustrates that the "Apply for a New Account", "Manage Account" and "View Stock Exchange Prices" tasks have been translated into base use cases (applying rule 1). Rule 2 causes the optional subtask "Print Operation Receipt" to become an extending use case of the superordinate "Manage Account" one, while rule 3 transforms the remaining non-optional subtasks ("Perform Operation" and "Close Secure Session", at this level) into included use cases.

As for the alternative task ("Perform Operation", included in the "Manage Account" one), it is translated as a generalized use case. Its alternative subtasks generate use cases that act as its specific classifiers (rule 4).

To create use case descriptions, besides the task model's structure, the access, the output and the action(s) within each task are equally taken into account by the plugin. As the procedure of the subtask "Display the Login Prompt", contains a single action (i.e. entering a card into card slot), the task will be shown as a step in the description of its super-ordinated task "Open Secure Session", instead of appearing as an included use case in the diagram (see rule 5 and Table 3).

"Validate" is another single action task. However, it has 2 alternative procedures (the user being able either to press the hardware validation button or to touch the validation button appearing on the screen). Consequently, after making it become a step in the "Transfer Money" task (by rule 5), the plugin puts all its procedures into the alternative flow of the same task, applying rule 10. Table 4 illustrates the description of the "Transfer Money" use case, where the alternative procedures for "Validate" show up.

As for the cyclic "Log In" task (Table 5), besides other properties, it has a stop condition which is transferred to the equivalent use case description as a "Cycle stop condition".

TABLE 3. "OPEN SECURE SESSION" USE CASE DESCRIPTION

Use Case Name	Open Secure Session		
Summary			
Goal	A secure session is open.		
Actor	User		
Pre-conditions			
Cycle stop condition			
Main flow	*Step*	*Action*	*Optional*
	1	Enter a card into card slot	No
	2	Log in	No
Alternative flow			
Side-effect			

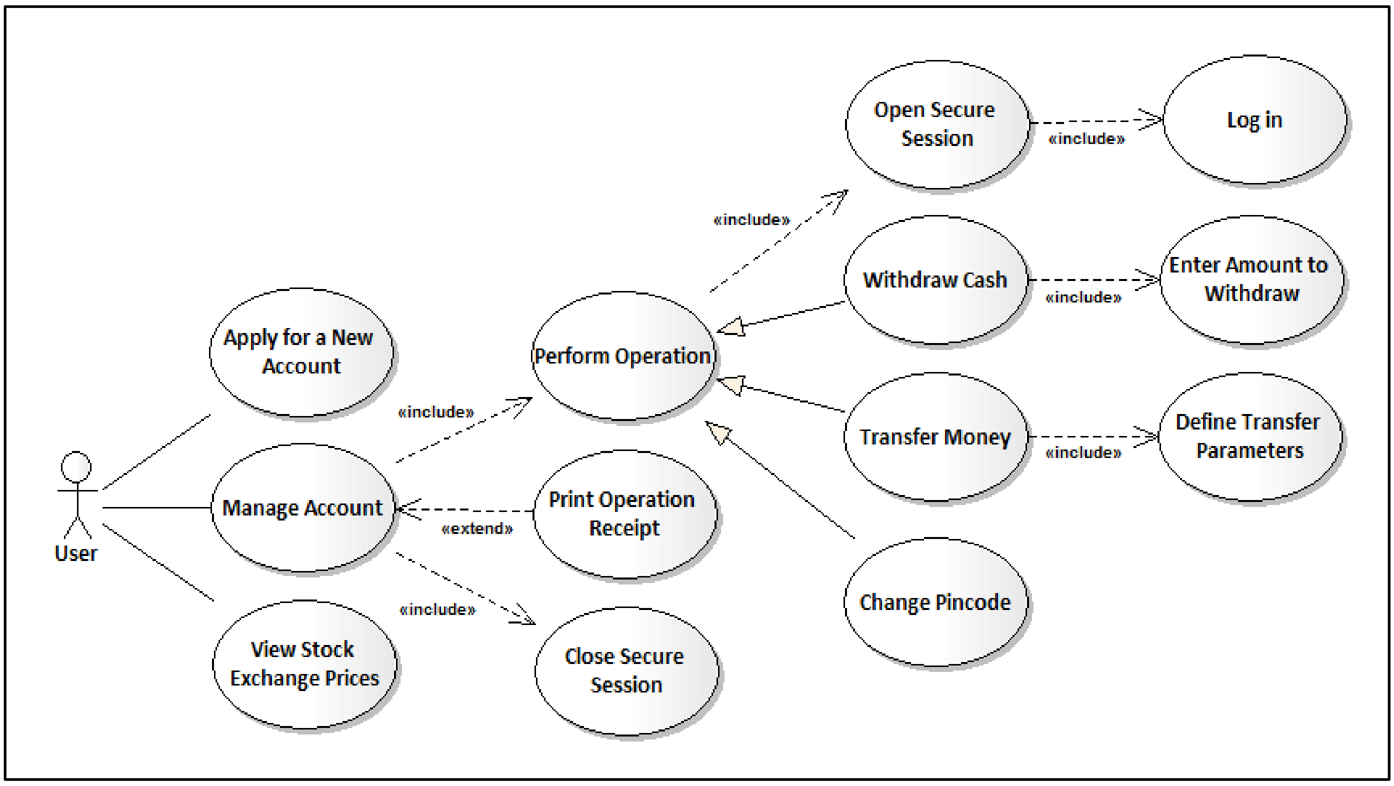

Figure 4.Use case diagram for ATM system

TABLE 4. "TRANSFER MONEY" USE CASE DESCRIPTION

Use Case Name	Transfer Money		
Summary			
Goal	An amount has been transferred.		
Actor	User		
Pre-conditions	A secure session is open.		
Cycle stop condition			
Main flow	*Step*	*Action*	*Optional*
	1	Define transfer parameters	No
	2	Validate	No
Alternative flow	*Alt.*	*Action*	
	2.1	Press the validation button.	
	2.2	Touch the validation button on the screen	
Side-effect			

TABLE 5. "LOG IN" USE CASE DESCRIPTION

Use Case Name	Log In		
Summary			
Goal	The user is logged in.		
Actor	User		
Pre-conditions			
Cycle stop condition	A pincode has been recognized by the system.		
Main flow	*Step*	*Action*	*Optional*
	1	Enter pincode	No
	2	Validate	No
Alternative flow			
Side-effect			

To summarize, Table 6 reveals all the characteristics of each task and lists the rules applied to it.

TABLE 6. RULES APPLIED FOR EACH TASK

Task Name	**Optional**	**Alternative**	**Cyclic**	**One-action-task**	**Multiple Procedures**	**First/Last Level (F/L)**	**Applied Rules**
Apply for a New Account						F/L	1, 6, 7
View Stock Exchange Prices						F/L	1, 6, 7
Manage Account						F	1, 6, 8
Open Secure Session							11, 3, 6, 8
Display the Login Prompt				✓		L	5
Log In			✓			L	3, 6, 7, 9
Perform Operation		✓					4
Withdraw Cash							4, 6, 8
Enter Amount to Withdraw						L	3, 6, 7
Transfer Money							4, 6, 8
Define Transfer Parameters						L	3, 6, 7
Validate				✓	✓	L	5, 10
Change Pincode						L	4, 6, 7
Print Operation Receipt	✓					L	2, 6, 7
Close Secure Session	✓					L	3, 6, 7

V. Evaluation

The proposed workflow builds on a free task modelling platform (the community version of TimBox®), a plugin developed for it (TASK2UML) and a UML software (Enterprise Architect®). The translation of task models into UML thus takes the task typology and structure of the TIM meta-model into consideration, although implementations using different task modelling approaches and software are obviously possible.

Effortless updating of TIM task models follows from the fact that task hierarchies are kept separate from the content of the tasks within them (their effects on the environment, such as the Input / Output transformation and possible Side-effects of it). These specifications are, in turn, kept separate from procedural information (the actions allowing to reach the Output of a task). Modifying a task hierarchy amounts to replacing, deleting, adding qualified nodes to tree, or defining new positions for them.

Beyond deployment, generating use case diagrams and use case descriptions from task models bridges a significant gap in well acknowledged software engineering methods, such as the Rational Unified Process (RUP), where it flows into the requirement specification phase within the Initial Planning, but generally in all those grounding design on UML representations (such as Extreme Programming, XP), and methodologies tailored to accept dynamic changes in the requirements (Agile Modelling).

The main strength of our proposal is to enable a user-centred perspective without stepping out of the mainstream UML design frame, to make it the very first step in a structured flow of design documentation, to implement a lightly formalized view of the system that remains understandable for the customer (the task model), and to automatically generate a rich use case diagram enhanced with use case descriptions that suit the needs and the background of developers.

VI. Conclusion

As task hierarchies are easy to update via a dedicated modelling software (such as TimBox®), and because the derived UML components are generated from them, the method above allows for dynamic reviews of the requirements: iterations affecting the task model are effortlessly translated into UML (in our example, by a TimBox® plugin that applies the transformation rules described in Table 2).

Besides, the engineering of task models is an incremental process that allows for a top-down strategy, starting at first-level tasks to capture the user's goals, then sub-tasking these to define system transactions, and finally designing user-system interactions within the description of last-level tasks. The collaboration of stakeholders (customers and customer stories for the first level, architects for the remaining ones) in the setting up of models is, thus, one of the major interests of a task-driven design process.

References

[1] Larman, C., Applying UML and Patterns: An introduction to object-oriented analysis and design and the unified process. Second edition, Prentice-Hall, 2002.

[2] OMG Unified Modelling Language (OMG UML), Superstructure, V2.1.2, Use Case, pp 601- 618. Internet: www.omg.org/spec/UML/2.1.2/Superstructure/PDF/

[3] Lilly, S., Use case pitfalls: Top 10 problems from real projects using use cases. In Technology of Object-Oriented Languages and Systems. TOOLS 30. Proceedings,1999.

[4] Daniel Sinnig, FrédéricRioux, Patrice Chalin, Use cases in practice: a survey, Dependable Software Research Group, Department of Computer Science and Software Engineering, Concordia University. Internet: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.84.8898&rep=rep1&type=pdf [Feb. 27, 2011]

[5] Gottesdiener, E., Top ten ways project teams misuse use cases – and how to correct them. Rational Edge. Internet: http://www.ibm.com/developerworks/rational/library/content/RationalEdge/jul02/TopTenWaysJul02.pdf [Feb. 27, 2011]

[6] Hamilton, K., Miles, R., Learning UML 2.0, O'Reilly Media, 2006.

[7] Forbrig, P., Buchholz, G., Dittmar, A., Wolff, A., Reichart, D., Model-based software development and usability testing. Internet: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.77.6506&rep=rep1&type=pdf [Feb. 27, 2011]

[8] Paterno, F., ConcurTaskTrees and UML: how to marry them? Internet: giove.isti.cnr.it/attachments/publications/2000-A2-041.pdf [Feb. 27, 2011]

[9] England, D., Palanque, J., Vanderdonckt, J., Wild, P. (Eds), Task models and diagrams for user interface design. 8th International Workshop, TAMODIA 2009, Brussels, Belgium, September 23-25, 2009, Revised Selected Papers. LNCS, Volume 5936, Springer, 2010.

[10] Paris, C., Tarby, J., Vander Linden, K. A flexible environment for building task models. In Vanderdonckt, J., Blandford, A., &Derycke, A. (Eds.), Proceedings of the IHM-HCI 2001 (Lille). Toulouse: Cépaduès-Editions, 2001

[11] Wettengel, T., TimBox, a multi-platform task modelling environment. Internet: http://www.timgroup.fr/?page_id=1015 [Feb. 27, 2011]

[12] Ha L., Huynh, T., Generating UML use cases from TIM task models. Graduating Thesis in Computer Engineering (supervisor: Wettengel, T.), International University, Vietnam National University of Ho Chi Minh City, 2009.