Use Case Maps: A Roadmap for Usability and Software Integrated Specification

A. Alsumait, A. Seffah, and T. Radhakrishnan
Computer Science Department, Concordia University,
Maisonneuve Blvd. W. Quebec, Canada.
seffah Email: @cs.concordia.ca

Abstract: The purpose of this paper is to explore use case maps as a medium for integrating task analysis and usability requirement into the traditional software requirement engineering process. The paper responds to major gaps in user interface specification in HCI, in software development methods for interactive software, and in the communication between usability specialists and software developers. We illustrate, via a concrete example, the usage of use case maps as an approach for specifying user interface and usability requirements. The Use Case Maps (UCMs) is a scenario-based notation for describing, in an abstract way, how the organizational structure of a complex system and the emergent behavior of the system are intertwined. It provides a first-class design model for the “how it works” aspect of both object-oriented and real time systems. Use case maps give a road-map-like view of the cause-effect paths traced through a system by scenarios or use cases (Buhr, 1998).

Key words: Use cases, use case maps, task analysis, user requirement, usability requirement, user interface requirement, functional requirement.

1. INTRODUCTION

It has been reported that a large number of change requests to modify an application are made after its deployment. Several studies have shown that 80% of total maintenance costs are related to problems of users with the system and not technical bugs (Randolph and Mayhew, 1994; Landauer,

1995). Among them, 64% are usability problems (Landauer, 1995). In a survey of 8,000 projects, the Standish Group found that the lack of user involvement and incomplete user requirements represent the major reason of project success or failure (Standish Group, 1995). One of the major reasons for this situation is that software engineering methodologies, when used for developing highly interactive software with a significant user interface, have a major limitation. Most of them do not propose, at least explicitly, any mechanisms for: empirically identifying and specifying user needs and usability requirements, or testing and validating requirements with end-users (Forbrig, 1999; Nunes and Falcãoe Cunha, 1999; Seffah and Hayne, 1999).

Among the software engineering techniques, use cases have been investigated as a potential approach for user requirement (Constantine and Lockwood, 1999; Forbrig, 1999; Krutchen, 1999; Nunes and Falcãoe Cunha, 1999). A use case is a collection of possible scenarios between the system and actors, characterized by the goals the primary actor has, while a scenario is a sequence of interactions happening under certain conditions (Cockburn, 1997). Although, in theory, use cases have the potential to gather the non-functional requirements that are a simplified description of the context of use, in practice, use cases have been used for gathering the system functionalities and features including technical capabilities and constraints. They attempt to describe representative ways in which the user will interact with the software but not in comprehensive format (Constantine, 1995). Another limitation of use case-driven requirement approach is that the main people involved in this process are stakeholders and technical persons including use case specifiers and user interface developers. End-users are not directly involved (Seffah and Hayne, 1999) and the notation used is not easily understandable.

User-centered requirement techniques and in particular task analysis suggested by the human-computer interaction (HCI) community has the power to complement traditional software engineering approaches and in particular use cases (Nunes and Falcãoe Cunha, 1999; Seffah and Hayne, 1999). User-driven requirement approaches are used for gathering a complete description of the context of use and usability goals including user characteristics, task analysis, as well as the physical, technical and organizational environments in which the system will be used (Setten et al, 1997; Arnowitz et al, 2000). However, most of the currently available HCI techniques are not intended for dialog modeling: system reactions are not represented, and error-free, non-interrupted action sequences are assumed (Jambon, 1996; Setten et al, 1997).

Therefore, both HCI and software engineering approaches each have their own strengths and weaknesses; in addition their objectives overlap in some areas but differ in others. For example, in model-based task analysis in HCI, the objective is normally to achieve a generic and thus abstract model of the

task, typically in a hierarchical form of goals and sub-goals (Dayton et al, 1998). In object-oriented development, use cases are often employed in gathering functional requirements (Constantine, 1995; Krutchen, 1999). HCI techniques enhance functional requirement descriptions by adding information that improves the users' understanding of the future system. Basically, this additional information concerns the context of use and takes into account usability problems that may be encountered by end-users when performing certain tasks.

Here, we discuss our investigations on the use of use case maps as an approach for specifying user interface and usability requirements as well as for bridging the current gap between software and usability engineering methods. The Use Case Map (UCM) is a scenario-based notation for describing graphically how the organizational structure of a complex system and the emergent behavior of the system are intertwined. It provides a first-class design model for the "how it works" aspect of both object-oriented and real time systems (Buhr, 1998). Use case maps give a road-map-like view of the cause-effect paths traced through a system by scenarios or use cases. Already, UCMs have been used in a number of areas (Buhr, 1998):

- Requirements engineering and design such as real-time systems, object-oriented systems, telecommunication systems, distributed systems, multimedia systems, agent systems.
- Detection and avoidance of undesirable feature interactions.
- Performance analysis and prediction.
- Evaluation of architectural alternatives.
- Functional testing.

UCMs are intended to be a useful tool for functional requirement, scenario-based software development, as well as for stakeholders-to-developers communication (Buhr, 1998; Seffah and Hayne, 1999; Amyot and Mussbacher, 2001). However, their support for designing user interfaces is still acknowledged to be insufficient. There is a common misconception that the same models developed to support the design of the application internals are not adequate to support interaction design, supporting the usability aspects of the applications. In this paper, we propose an extension of UCMs for user interface systems. This extension provides the necessary integration between user interfaces and underlying functionalities. In fact, by having a single modeling notation, common structures and behaviors of user interfaces and the underlying functionalities can be shared at design time among usability engineers and developers. Furthermore, user interface developers can have the benefits of designing user interfaces in a declarative, systematic, and also standard way. Our starting position is that a good proportion of user interface (usability) design problems may be served by

scenarios and use cases, as perhaps the primary design representation. We should keep in mind that our proposed extension should not be tied to a specific view of the design process. It should try to serve the needs of user interface designers within the framework of the UCM model.

2. BACKGROUND AND RELATED WORK

The following are only some of the many investigations that, over the last few years, have tried to study use cases and task analysis for requirement engineering and in general build a bridge between object-oriented development and task analysis techniques.

Forbrig (1999) introduced a framework for combining task models, user models, and object models. These multiple dimensions require tools enabling the manipulation of all these models along the different phases of software development. Related to this work, Seffah and Hayne (1999) and Antunes et al. (2001) also investigated the similarities of task analysis and use case techniques via an attempt to recast the results of a task analysis into the format of use cases as well as software-to-usability communication, which is not explicitly described in use cases-driven development. Paternò (2001) discussed an approach to integrating the Unified Modeling Language (UML) for designing interactive systems based on the use of use cases, class diagram, ConcurTaskTrees (CTT) task models and scenarios. The logical framework that has been presented gives an indication of the types of representations to use in the various phases and their relationships.

Constantine and Lockwood (1999) suggest that use case specifiers should first prepare lightweight use case model descriptions (essential use cases) that do not contain any implicit user interface decisions. Later on, user interface designers can use these essential use cases as input to create the user interface without being bound by any implicit decisions. Related to this work, Krutchen (1999) introduces the concept of use case storyboard as a logical and conceptual description of how a use case is provided by the user interface, including the interaction required between the actor(s) and the system.

Theoretically, having a unique and consistent model that provides simple views for any actor and automatically includes user's concerns should be enough to enable the software engineers to keep track of user needs during their design process. However, as Artim and Van Harmelen (1998) pointed out in their case study, the culture of the software engineer does not include collaborating with the user in the process of building a better system. These sociological forces within development teams limit and even discard any impacts of the user in the development of the system, thus providing a

system that fits to static user specifications, rather than fitting the best to user needs. We can deduce from this case study that even though the process of development determines directly the product being created, it is not the only factor. Jarke (1999) tries to clarify the purpose and manner to use scenarios in the modeling process, since this concept can be used in very different ways. He defines the scenarios as constructs that describe possible sets of events that might reasonably take place; they offer “middle-ground abstraction between models and reality”.

3. USE CASE MAPS – A BRIEF INTRODUCTION

Use Case Maps (UCMs) is a visual notation developed at Carleton University (Buhr, 1998). The key idea of UCMs is to describe scenario paths in terms of causal relationships between responsibilities. A causal relationship between components can be interpreted in many ways, depending on the component structure. As shown in Figure 1, UCMs are composed of:

- **Start points** (filled circles representing pre-conditions or triggering causes)
- Causal chains of **responsibilities** (crosses, representing actions, tasks, or functions to be performed)
- **End points** (bars representing post-conditions or resulting effects).

Figure 1. An example of a Use Case Map (Buhr, 1998).

Start point, responsibilities, end points, and components are the basic elements of the UCM. The responsibilities can be bound to **components**, which are the entities or objects composing the system. The wiggly lines are **paths** that connect start points, responsibilities, and end points.

UCMs are used to describe and integrate use cases representing the requirements (Amyot and Mussbacher, 2001). The construction of UCMs can reveal problems with the use cases, which may be incomplete, incorrect, ambiguous, inconsistent, or at different levels of abstraction (Amyot and Mussbacher, 2001; Buhr, 1998). In fact, the usefulness of UCMs is broader than their name suggests. They aim to support human understanding of the entire picture of requirements during all phases of the lifecycle, and not just to specify scenario sequences by combining a set of scenarios in a single diagram that enables designers to express scenarios and scenario interactions in a graphical manner (Amyot and Mussbacher, 2001). UCMs provide a mechanism that can be used by designers to analyze the overall system behavior that emerges from scenario combinations (Buhr, 1998). Moreover, UCMs provides a bird's-eye, path-centric view of system functionality. It allows for dynamic behavior and structures to be represented and evaluated, and these in turn improve the level of reusability of scenarios (Buhr, 1998). For more information, readers may visit the UCMs web site at www.UseCaseMaps.org.

3.1 UCMs Extension for Usability Requirements

One important goal of user interface and usability requirements engineering is to ensure that a consistent and feasible requirements specification can be developed, while ensuring that the specification is a valid reflection of user requirements. To achieve this goal, we investigated the possibility of making use of the prospective standard Use Case Maps (UCMs) for the support of user interface and usability requirements.

Our strong belief that UCMs are powerful for user interface and usability requirements is based on the simplicity of UCMs notation. Based on previous studies (Buhr, 1998; Amyot and Mussbacher, 2001), we believe that the UCMs notation is easily understandable by both the user (customer) and the software developer. In fact, this helps user interface designers to handle different users' understanding and expectation of the interface, and bridge the gaps by refining the requirements earlier. If the software developer is using UCMs, this will support the idea of concurrent engineering where a whole overview of the system (functionality and user interface) can be presented using one abstract model. Consequently, this improves the probability that software and the interface will be correct when it is finally put together, since contradictions in the requirement can be captured in early stages, which shortens the overall time required and makes it correct. It will also help to reduce the chances of introducing errors when changes are made. Furthermore, many studies are focusing on how to

integrate UCMs to UML and how to formalize UCMs in XML (Amyot and Mussbacher, 2001).

Finally, a major strength in the extended UCMs is its ability to capture most of the user interface requirements. We found that UCMs can be used in the following three dimensions:

- **Task Dimension** seeks to represent tasks that are relevant for interactions. Thus, UCMs are used as a simple and expressive visual notation that allows describing task scenarios at an abstract level in terms of sequences of responsibilities and tasks over a set of components. Tasks can be split up into subtasks (actions, operations) or inherited from 'super'-tasks.
- **Dialog Dimension** is intended to explain the style of human-computer interaction and also describes the sequence of dialogs that can take place between the user and the system.
- **User Interface Dimension** identifies the objects comprising the user interface, their grouping and specifies their layout, e.g. by indicating approximate placement or by indicating topological relations between groups. This dimension represents the space within the user interface of a system where the user interacts with all the functions, containers, and information needed for carrying out some particular task or set of interrelated tasks. Moreover, successive displays of different screens and interactive objects are presented.

The 'look and feel' of an interactive system concerns the detailed reactive behavior of interactive objects that make up the interface. Using UCMs, these objects can be described in a range of forms. Use cases can provide a description that abstracts away from the software structure, while the dialog dimension can provide a more detailed account of this interaction. Figure 2 introduces the extended notation to provide UCMs with more expressive power for interaction design. These notations are described below:

- a. System initiates a dialog.
- b. The initiator of the dialog could be the system or the user.
- c. Dialog between the system and the user is repeated until one of them ends the dialog.
- a. Same as 3 except that the dialog is repeated n times.
- e. The dialog is optional.
- f. Proactive message; message passed from the system to the user.
- g. A question-and-answer dialog
- h. A form-filling dialog
- i. A menu dialog.

Figure 2. Extended notation for UCMs

In short, our proposal for the extended UCM framework for user interface systems should include a new UCM interaction model to accommodate the three dimensions. Therefore, the user interface model encompasses the task, dialog and layout structure dimensions, clearly mapping the conceptual architectural models for interactive systems, while maintaining the desired separation of concerns.

1.2 AN ILLUSTRATIVE EXAMPLE

Filter Agent (FA) aims to assist the user with email management using a memory based learning technique. The interface agent (FA) learns by continuously “looking over the shoulder” of the user as the user is performing actions. The interface agent monitors the actions of the user over long periods of time, finds recurrent patterns and offers to automate them. If the user drags and drops a particular electronic mail message to a specific folder, the mail agent adds a description of this situation and the action taken by the user to its memory of examples. The agent keeps track of the sender and receiver of a message, the Cc: list, the keywords in the Subject: line, whether the message has been read or not, whether it is a reply to a previous message, and so on. If a new mail arrives, the agent predicts which action is appropriate for the current situation and also measures its confidence in each prediction by determining how many examples the agent has memorized. If the agent is confident, then it will autonomously take the action on behalf of

the user. Otherwise, the agent either offers suggestion to the user and wait for the user's confirmation to automate the action, or wait and observe the user action.

1.2.1 Task and Dialog Dimension

The behavior of the filter agent and consequently the relationship among the functions mentioned earlier are better understood by following UCMs flows shown in Figure 3. Based on the root map in Figure 3, users and designers can consider early decisions regarding the sequence in which functions are performed. This map describes the system behavior that starts when a pre-condition is satisfied, for example, new email arrives (filled circle labeled MA).

A dialog illustrates that a conversation between the user and the agent is taking place where details are delayed to a sub-UCMs. The dialog notation is applied to our work not only to hide details, but also to decompose the system into small manageable units. This scenario ends when one or two of the following triggering events occur: user exits the email application (bar labeled E1) or an email is filed (bar labeled E2). These events are represented by post-conditions in the UCMs. A route is a path that links an initial cause to a final effect. For example, <S, [a2], [a4], D, E2> represents a route that starts when new email arrives, user reads the mail and sends it to a specific folder, followed by a dialog between the user and the agent.

Figure 3. The Root Use Case Map

Figure 4. UCM explaining the dialog between the agent and the user

Figure 4 depicts the second level of the requirement model when a conversation or a dialog goes on between the agent and the user. The agent offers its suggestion to the user waiting for a confirmation. The user either confirms the suggestion or denies it. Alternative paths (called OR-forks) represent composite UCMs that can be split into two different paths (no level of concurrency is associated with them). For instance, a responsibility point (cross-labeled AA in the figure) is activated along the [b1] path to decide whether the user confirms the suggestion or denies it. The alternative sub-paths (labeled [b2] and [b3]) are generated after this suggestion.

1.3 Task and Structural Dimension

The enriched UCMs not only describe the sequence of tasks and dialogs that can take place between the user and the system, but also help to understand and reason about the requirements of the user interface, including usability aspects.

Figure 5 shows the user interface structure of the email application that consists of a tool bar, folder menu, mailbox, and an agent window.

Figure 5. Structured UCMs for the user interface of the email application

Figure 5 describes the approximate position of the components of the user interface and the user/system behavior that starts when a user opens the email application. The user downloads new arrived mails using the tool bar menu. Then the user continuously reads a mail then folders it until either the agent initiates a dialog or the user quits the application.

2. CONCLUSION AND FURTHER INVESTIGATION

The most valuable contribution of this ongoing research effort is the development and analysis of a complete UCMs notation for high-level requirements of user interfaces. Our enriched UCMs put together three dimensions of user interface requirements including the task, the dialog, and the structure of the user interface. In this manner, functions and tasks are distributed over the use case map defining how the users will be permitted to move among various tasks and how and when the user will move from one object to another in the interface. Moreover, the enriched UCMs capture the initiator and style of interactions between the user and the system.

The improved UCMs have also the potential to capture a complete picture of user interface requirements by describing how a system will be used and to what ends. However, the amount of work it generates shows the vital need of a CASE tool to manage the use-case descriptions along with additional information pertinent to user interface analysis and design. This tool should also manage many views of the same model, and is able to generate five types of reports on a model corresponding to the needs of different actors and steps of the development process:

- A "user's view" of the use cases,
- A "developer's view" of the use cases,
- A glossary of terms,
- A more focused user interface analyst's report,
- An actor-system interaction report.

5. REFERENCES

- Amiot, D. & Mussbacher, G. (2001), Bridging the requirements/design gap in dynamic systems with Use Case Maps (UCMs). In: *23rd International Conference on Software Engineering -ICSE'01*, May 12-19. Toronto, Canada. 743-744.
- Antunes, A., Seffah, A., Radhakrishnan, S. & Pestina, S. (2001) Improving and mediating software-to-usability communication. In: *Workshop on Multiple User Interfaces over the Internet: Engineering and applications Trends In conjunction with HCI-IHM'2001*, <http://www.cs.concordia.ca/~faculty/seffah/ihm2001/index.shtml>, May 10-14, Lille, France.
- Arnowitz, J., Fijma, D. & Verlinden, J. (2000), *Communicating a task analysis with task layer maps*. ACM Press New York, NY, USA, 346-353.
- Artim, J.M. & Van Harmelen M. (1998), Incorporating work, process and task analysis into commercial and industrial object-oriented system development. In: *Proceedings of the conference on CHI 98 summary: human factors in computing systems*, ACM Press, 198.
- Buhr, R.J.A. (1998), Use case maps as architectural entities for complex systems, *IEEE Transactions on Software Engineering*, 24(12), 1131-1155.
- Cockburn A. (1997), Structuring Use Cases with Goals. *Journal of Object-Oriented Programming*, Sep/Oct, 1997, 35-40, and Nov/Dec, 1997, 56-62.
- Constantine, L. & Lockwood, A.D. (1999), *Software for use: a practical guide to the models and methods of usage-centered design*. Reading, Massachusetts: Addison-Wesley.
- Constantine, L. (1995), Essential modeling: use cases for user interfaces. *Interactions*, ACM Press, 2(2), 34-46.
- Dayton, T., McFarland, A. & Kramer, J. (1998), Bridging user needs to object-oriented GUI prototypes via task object design. In: L. Wood (ed.), *Design: User Interface Design: Bridging the Gap from User Requirements to Design*, Boca Raton, FL: CRC Press, 15-56.
- Forbrig, P. (1999), Task and object-oriented development of interactive systems - how many models are necessary?. In: *Proceedings of Design Specification and Verification of Interactive Systems Workshop-DSVIS'99*, Braga, Portugal, 225-237.
- Jambon, F. (1996), Formal modeling of task interruption. In: *Proceedings of the CHI '96 conference companion on Human factors in computing systems: common ground*. ACM Press, 45-46.
- Jarke, M. (1999), Scenarios for modelling. *Communications of the ACM*, 42(1), 47-48.

- Krutchén, P. (1999), Use case storyboards in the rational unified process. In: *Workshop on Integrating Human Factors in Use Case and OO Methods-13th ECOOP*. June 14-19, Lisbon, Portugal.
- Landauer, T.K. (1995), *The trouble with computers: usefulness, usability and productivity*. MIT Press.
- Nunes, N.J. & Falcão e Cunha, J. (1999), Detailing use-cases with activity diagrams and object views. In: *Workshop on Integrating Human Factors in Use Case and OO Methods-13th ECOOP*. June 14-19, Lisbon, Portugal.
- Paternò, F. (2001), Towards a UML for interactive systems. In: M.R. Little, L. Nigay (eds.), *8th International Conference of EHCI 2001*, Toronto, Canada, May 2001, 7-18.
- Randolph, G.B. & Mayhew, D.J. (1994), *Cost-justifying usability*. Academic Press.
- Seffah, A. & Hayne C. (1999), Integrating Human Factors in Use Case and OO Methods. . In: *Workshop on Integrating Human Factors in Use Case and OO Methods-13th ECOOP*. June 14-19, Lisbon, Portugal.
- Setten, M., van der Veer, G. & Brinkkemper, S. (1997), Comparing interaction design techniques: a method for objective comparison to find the conceptual basis for interaction design. In: *Proceedings of the Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques*. August 18-20, Amsterdam, The Netherlands, 349-357.
- Standish Group. (1995), *CHAOS: A Recipe For Success*, The Standish Group International, INC.