Bridging the Requirements/Design Gap in Dynamic Systems with Use Case Maps (UCMs)

Daniel Amyot, Gunter Mussbacher
Strategic Technology Group
Mitel Corporation
350 Legget Drive, P.O. Box 13089
Kanata, Ontario, Canada K2K 2W7
+1 613 592-2122
{daniel_amyot, gunter_mussbacher}@mitel.com

ABSTRACT
Two important aspects of future software engineering techniques will be the ability to seamlessly move from analysis models to design models and the ability to model dynamic systems where scenarios and structures may change at run-time. Use Case Maps (UCMs) are used as a visual notation for describing causal relationships between responsibilities of one or more use cases. UCMs are a scenario-based software engineering technique most useful at the early stages of software development. The notation is applicable to use case capturing and elicitation, use case validation, as well as high-level architectural design and test case generation. UCMs provide a behavioural framework for evaluating and making architectural decisions at a high level of design. Architectural decisions may be based on performance analysis of UCMs. UCMs bridge the gap between requirements and design by combining behaviour and structure in one view and by flexibly allocating scenario responsibilities to architectural components. They also provide dynamic (run-time) refinement capability for variations of scenarios and structure and they allow incremental development and integration of complex scenarios. Therefore, UCMs address the issues mentioned above.

Keywords
Architectural reasoning, design, model derivation, requirements, scenarios, UCM styles, UML, URN, use cases.

1 OVERVIEW OF USE CASE MAPS

Use Case Maps (UCMs) [6,7] visually describe causal relationships between responsibilities superimposed on organizational structures of abstract components. *Responsibilities* represent generic processing (actions, activities, operations, tasks, etc.). *Components* are also generic and can represent software entities (objects, processes, databases, servers, etc.) as well as non-software entities (e.g. actors or hardware). The relationships are said to be causal because they link causes (e.g., preconditions and triggering events) to effects (e.g. postconditions and resulting events) by arranging responsibilities in sequence, as alternatives, or concurrently. Essentially, UCMs show related use cases in a map-like diagram, whereas UCM *paths* show the progression of a scenario along a use case.

2 BENEFITS OF USE CASE MAPS

The two main strengths of UCMs reside in their capacity to, first, model dynamic (run-time) refinement for variations of behaviour and structure and second, visually integrate behaviour and structural components in a single view.

Dynamic (Run-time) Refinement

The *dynamic stub* is a notational element of UCMs that can be used to localize and visualize, at design time, how alternative behavioral scenarios could evolve at run time. These alternatives are described with sub-maps, called *plug-ins*, associated with the dynamic stub. The *selection policy* of a dynamic stub defines which plug-ins should be selected at run-time. This mechanism addresses modeling issues of future systems where protocols and communicating entities become more dynamic in nature and may not be known at design-time but evolve at run time.

The UCM notation can incrementally combine and integrate scenario paths and dynamic stubs make more local the potential conflicts that could arise between scenarios (hence leading to simpler analysis). The selection policies can help avoiding or resolving these conflicts.

Bridging the Gap Between Requirements and Design

By combining behavior and structure in one view in an explicit and visual way, UCM enable the understanding of scenario paths in their context as well as high-level architectural reasoning. As UCM paths can easily be decoupled from structures and are less likely to change than underly-

ing architectures, UCMs improve the reusability of scenarios and lead to behavioral patterns that can be utilized across a wide range of applications. If a structure is modified, path elements need only to be reallocated to appropriate components.

UCMs facilitate abstract thinking at the right level of detail for architectural design since UCM descriptions abstract from messages, protocols, communication constraints, data, control, and structural evolution while focusing on general causal flows between causes, responsibilities, and effects.

3 SCOPE OF USE CASE MAPS

UCMs fit in many different software engineering methodologies and design processes, e.g. the Unified Modeling Language [9]. UML consists of diagram groups emphasizing either structural or behavioural aspects of a system. Structural diagrams can capture some system requirements such as the architecture and the application domain. Structural diagrams share connections with behavioral diagrams, where the referenced entities often come from structural diagrams. Yet, there exists a conceptual gap between functional requirements (and use cases) and their realization in terms of behavioral diagrams, as illustrated in Fig. 1.

Requirements and use cases often provide a black-box view where the system is described according to its external behavior. UML behavioral diagrams have a glass-box view describing internal behavior in a detailed way. A UCM view represents a useful piece of the puzzle that helps bridge the gap between requirements and design. UCMs can provide a *gray-box view*, a traceable progression from functional requirements to detailed views based on states, components and interactions, while at the same time combining behavior and structure in an explicit and visual way.

Initially, requirements engineers and customers use UCMs to describe the behavioural aspect of the system. System architects and designers will then refine UCMs and reason about the architecture by adding component structures to the maps. From UCMs, more detailed UML views or models in other modeling/specification languages may be derived, including class diagrams [6], Message Sequence Charts [5], Statecharts/RoomCharts [4], and LOTOS [1] and SDL specifications [10] (to validate UCMs and designs and to detect undesirable interactions early in the design cycle). Other research projects include the generation of performance models [11] and of validation test cases.

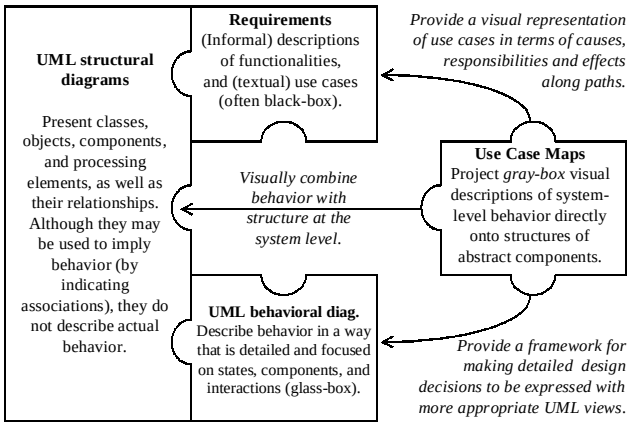

Fig. 1 UCMs as a missing piece of the UML puzzle.

ITU-T has recognized the need for a User Requirements Notation (URN) and initiated its standardization process which may lead to a recommendation by 2003. UCMs, with its unique characteristics [3], play an integral role in the current proposal for the standard [8]. Similarly, UCMs have found considerable support in the UML community [2] and are under consideration for UML 2.0. Tool support (UCMNAV) and a UCM user group are also available [12].

REFERENCES

1. Amyot, D., Buhr, R.J.A., Gray, T., and Logrippo, L. Use Case Maps for the Capture and Validation of Distributed Systems Requirements. In *Proc. IEEE RE'99*, (Limerick, Ireland, June 1999), 44-53.
2. Amyot, D. and Mussbacher, G. On the Extension of UML with Use Case Maps Concepts. In *<<UML>>2000*, (York, UK, Oct.). LNCS 1939, 16-31.
3. Amyot, D. and Eberlein, A. An Evaluation of Scenario Notations for Telecommunication Systems Development. In: *Proc. 9th ICTS* (Dallas, TX, March 2001).
4. Bordeleau, F. *A Systematic and Traceable Progression from Scenario Models to Communicating Hierarchical Finite State Machines*. Ph.D. thesis (Carleton University, Canada, Dec. 1999).
5. Bordeleau, F. and Cameron, D. On the Relationship between Use Case Maps and Message Sequence Charts. In: *Proc. SAM2000* (Grenoble, France, June 2000).
6. Buhr, R.J.A. and Casselman, R.S. *Use Case Maps for Object-Oriented Systems*. Prentice Hall, 1996.
7. Buhr, R.J.A. Use Case Maps as Architectural Entities for Complex Systems. In: *IEEE Transactions on Software Engineering*, 24 (12), Dec. 1998, 1131-1155.
8. Cameron, D. et al. *Draft Specification of the User Requirements Notation*. Canadian contribution CAN COM 10-12 to ITU-T, Nov. 2000
9. OMG (1999) *Unified Modeling Language Specification, Version 1.3*. June 1999. http://www.omg.org
10. Sales, I. and Probert, R. From High-Level Behaviour to High-Level Design: Use Case Maps to Specification and Description Language. In: *Proc. SBRC'2000* (Belo Horizonte, Brazil, May 2000).
11. Scratchley, W.C. (2000) *Evaluation and Diagnosis of Concurrency Architectures*. Ph.D. thesis (Carleton University, Canada, Nov. 2000).
12. *Use Case Maps Web Page, User Group, and Virtual Library*. On-line at http://www.UseCaseMaps.org