

UCM Styles and UCM Path Traversal

Daniel Amyot, Gunter Mussbacher

Mitel Networks

Daniel_Amyot@mitel.com

Objectives

- ◆ Give an overview on Use Case Maps (UCM) styles (guidelines on how to apply UCMs depending on the project context)
- ◆ Introduce one possible path traversal mechanism and point out general issues which need to be resolved
- ◆ Discuss implications of UCM styles on the path traversal mechanism
- ◆ Understand the relationship of UCMs and MSCs or UML Sequence Diagrams
- ◆ Demonstrate the generation of MSCs from UCMs with the UCMNAV tool - the UCM Navigator

Table of Contents

◆ Use Case Map Styles for Small and Large Systems

- ◆ Path Traversal of Use Case Maps
- ◆ Generation of Message Sequence Charts From UCMs

Applicability of Use Case Maps Styles

- ◆ Individual Maps
 - Document few key features
 - Get an initial understanding of the system and early feedback from stakeholders
- ◆ Standard Root Map
 - Document small, evolving system
 - Interacting features increase the system's complexity
- ◆ Large System
 - Document large, evolving system
 - Interacting features increase the system's complexity

UCM Style - Individual Maps

- ◆ Describe each feature individually (hierarchy of UCMs may be used for a feature)
- ◆ Each UCM belongs only to one feature and is not reused for any other feature
- ◆ Mainly concerned with ...
 - Understandability of single features, ability to specify test cases, tool dependency (non-authors)
- ◆ Not concerned with ...
 - Scalability, feature interaction, evolveability, reusability across features
- ◆ Problems with consistency (too much cut & paste)

UCM Style - Standard Root Map

- ◆ Describe base feature on stub-rich root map with default plug-ins (hierarchy inherent)
- ◆ Describe variations of the base feature (i.e. other features) with the base root map and one or more different plug-ins to the stubs
- ◆ Mainly concerned with ...
 - Feature interaction, evolveability, reusability across features, understandability of single features, ability to specify test cases, tool dependency (non-authors)
- ◆ Not concerned with ...
 - Scalability

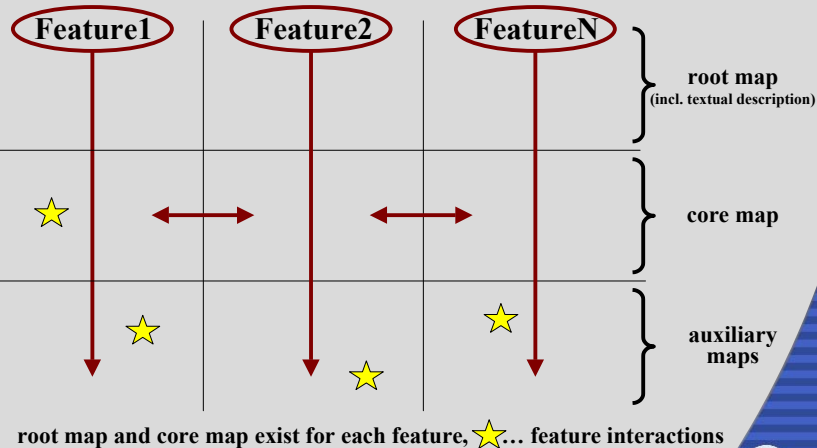
UCM Style - Standard Root Map - Problems

- ◆ Feature maps get polluted
 - Path segments on a feature map exist only because of a different feature
 - Difficult to distinguish the path segments which belong to the feature and the ones that belong to another feature
- ◆ Feature maps are distributed
 - Plug-ins required for feature are disjoint
 - No obvious connection exists between the plug-ins
- ◆ Feature definitions are buried in map hierarchy
 - Difficult to find beginning
 - Difficult to follow feature path

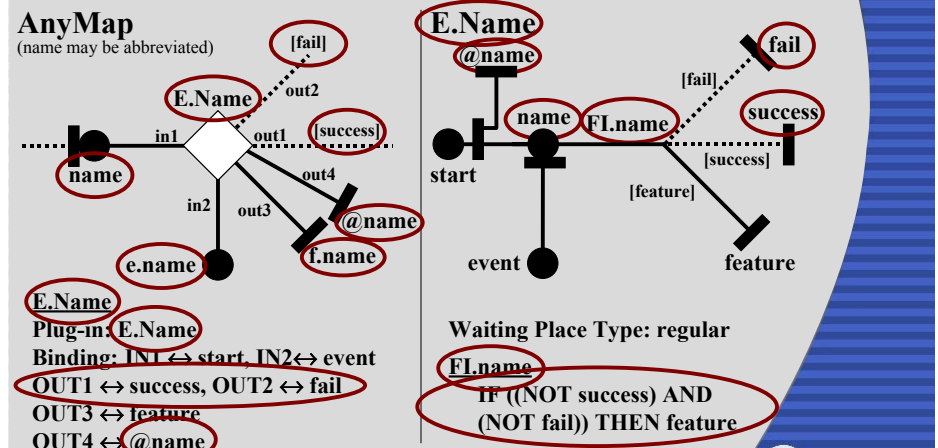
UCM Style - Large System

- ◆ Describe each feature on a separate root map (hierarchy of UCMs may be used for a feature)
- ◆ Integrate features by explicitly defining interaction points such as locations where ...
 - a variation of the feature occurs or the system is ready to deal with an event that may not be related to the feature
- ◆ Mainly concerned with ...
 - Scalability, feature interaction, evolveability, reusability across features, understandability of single features, ability to specify test cases, tool dependency (non-authors)

UCM Style - Large System - Overview



UCM Style - Large System - Event Stub Structure

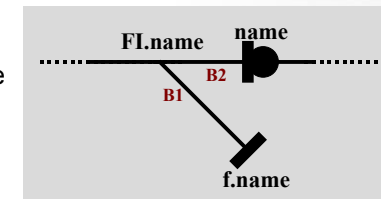


UCM Style - Large System - Event Stub Structure

- ◆ If a path passes through the stub by entering
 - At the "name" start point, then the system is now ready to deal with the next event inside the stub (the "@name" end point may be used to continue the scenario by connecting to the next event)
 - At the "e.name" start point and the system is ready, then the FI (feature interaction) fork "FI.name" will decide which out-path to take
- ◆ Out-paths in the "E.Name" plug-in
 - Typical out-paths are success, fail, @, and feature
 - At least the @ and feature out-paths need to exist, but there may also be more than the ones listed above

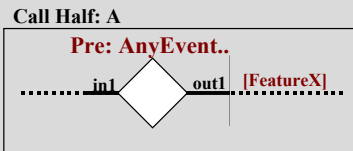
UCM Style - Large System - Event Stub Structure

- ◆ The part of the "E.name" plug-in which lies after the waiting place may look different for each instance of the event stub structure
- ◆ The FI fork may be used on its own without the rest of the Events pattern (see below for example)
 - Definition of FI fork explicitly states the reason for taking the feature branch (e.g. B1: feature X, B2: else)



UCM Style - Large System - Precondition Structure

- ◆ Use a stub to represent preconditions of a feature
- ◆ Bind stub to map where “E.Name” stub or FI fork is
 - Allow reuse of scenario paths by binding stub NOT directly to the “E.Name” plug-in
 - Precondition stubs appear mainly at the core map level
 - In-path of a precondition stub may not be bound (label start point with “dummy” (e.g. if out-path is bound to FI fork))



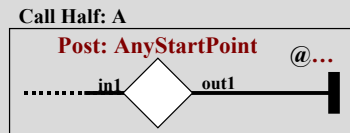
Pre: **AnyEventStubOrFIfork**
 Plug-in: **MapWithEventStub...**
 Binding: IN1 ↔ e.name
 OUT1 ↔ f.name

FeatureX



UCM Style - Large System - Postcondition Structure

- ◆ Postcondition
 - Use a stub to represent postconditions of a feature
 - Bind stub to map where path continues
 - Postcondition stubs appear mainly at root map level
- ◆ Don't use same stub for pre- & postcondition

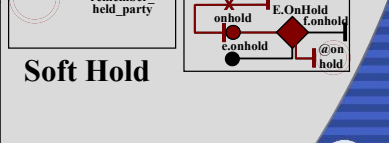
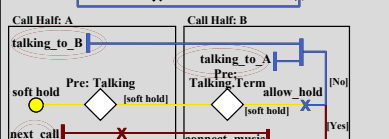
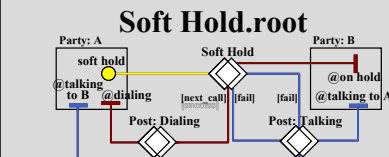
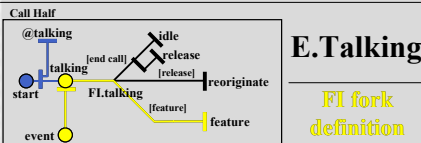
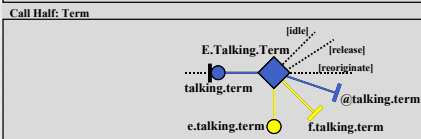
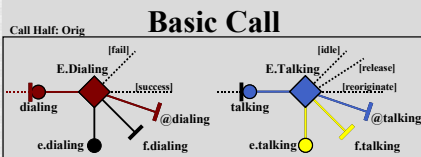


Post: **AnyStartPoint**
 Plug-in: **MapWithStartPoint**
 Binding:
 IN1 ↔ AnyStartPoint (name)
 OUT1 ↔ AnyEndPoint (@name)

Feature X



UCM Style - Large System - Summary - Example - Soft Hold



Key Points - UCM Styles

- ◆ Different contexts require different styles to efficiently document requirements and high level design
- ◆ The “Individual Maps” style is applicable to a small set of key features
- ◆ The “Standard Root Map” style is applicable to a small system
- ◆ The “Large System” style is applicable to large systems and provides a good balance of independent specification of features and integration of features
 - Patterns have been identified to achieve the above
 - Event stub, precondition stub, postcondition stub



Table of Contents

- ◆ Use Case Map Styles for Small and Large Systems

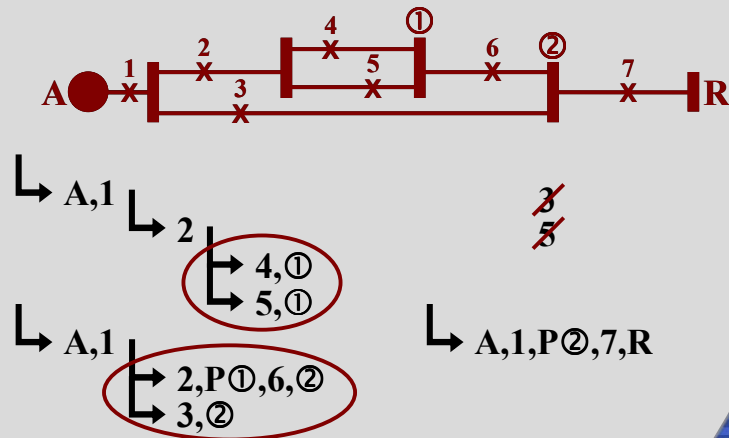
◆ Path Traversal of Use Case Maps

- ◆ Generation of Message Sequence Charts From UCMs

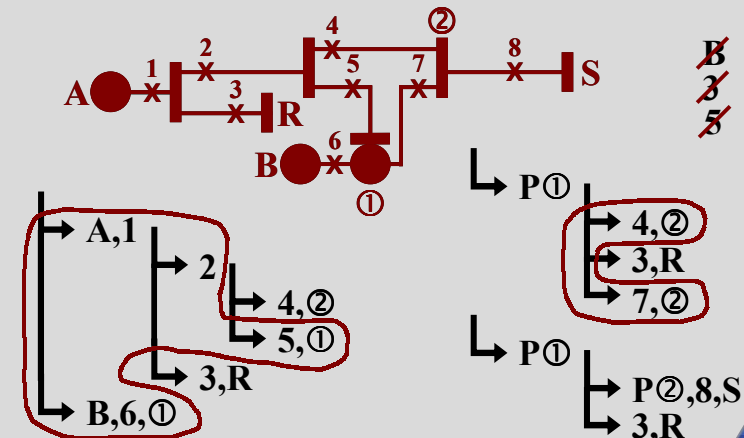
UCM Path Traversal

- ◆ Starts at one or more parallel start points as defined by user
- ◆ Starts with initial values (true, false, or undetermined) for each path data variable as defined by the user
- ◆ Moves from path element A to path element B if continuation criteria are met for element A
 - Each UCM path element has specific criteria
- ◆ Issues a warning if path traversal is stuck

UCM Path Traversal - Example I



UCM Path Traversal - Example II



Applications of UCM Path Traversal

- ◆ Highlighting
- ◆ Animation
 - Requires sequence numbers
- ◆ MSC generation
 - Requires component information
 - Well-nestedness transformation and warning mechanism
- ◆ LQN generation
 - Requires arrival and device characteristics, device demands, data access modes, response-time requirements
- ◆ Test case generation
 - Requires controllable and observable activities

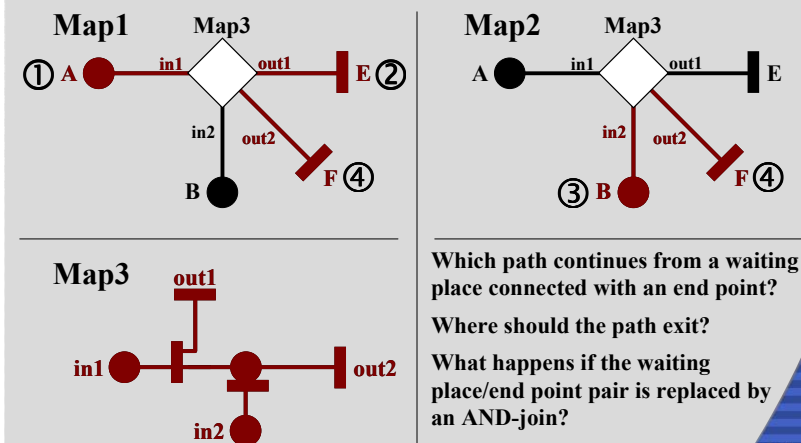
UCM Path Traversal Issues

- ◆ Component and plug-in instances
 - Is a component a new instance or does it reference an existing component on the same or another map?
 - If path traversal visits a plug-in map more than once, is it visiting the same plug-in or a new instance?
 - ◆ If the plug-in is a new instance, the path segment on the plug-in map is also a new instance but components on the plug-in map may not necessarily be new instances
- ◆ Loops
 - Suggested path traversal mechanism may be able to deal with loops
 - ◆ Needs to be tested - may require pre-scanning of the path
 - For which applications do loops have to be preserved?

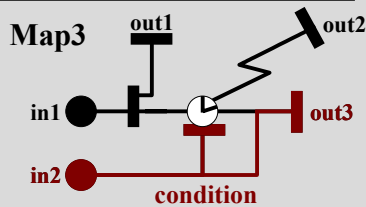
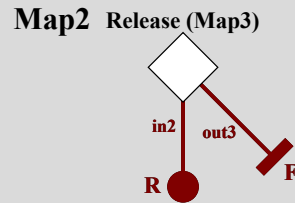
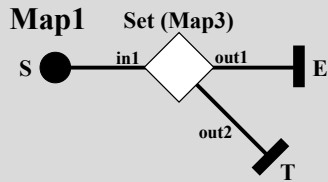
Implications of UCM Style on Path Traversal

- ◆ Individual Maps & Standard Root Map
 - No special requirements on path traversal mechanism
- ◆ Large System
 - Additional requirements on path traversal mechanism due to the use of precondition, postcondition, and event stubs
 - ◆ Precondition (postcondition) stubs allow a scenario to start (end) at a waiting place
 - ◆ In-path of precondition stub may not be bound
 - ◆ Out-path of precondition stub is guarded
 - ◆ The @name out-path of an event stub is connected to many start points representing all possible next events (guards are required at these start points)
 - ◆ Semantics of waiting place inside event stub to be clarified

Implications of Large System UCM Style on Path Traversal



Implications of Large System UCM Style on Path Traversal



How to use information about being at a waiting place or timer for a decision at a choice point?

Key Points - UCM Path Traversal

- ◆ Many correct path traversal mechanisms exist because of breadth-first and depth-first approaches and various ways of dealing with concurrency
- ◆ Path traversal is instrumental for advanced functionality such as highlighting, animation, as well as the generation of MSC, LQN, and test cases
- ◆ The biggest open issues for path traversal revolve around component & plug-in instances as well as loops
- ◆ The Large System UCM style puts additional requirements on the path traversal mechanism

Table of Contents

- ◆ Use Case Map Styles for Small and Large Systems
- ◆ Path Traversal of Use Case Maps

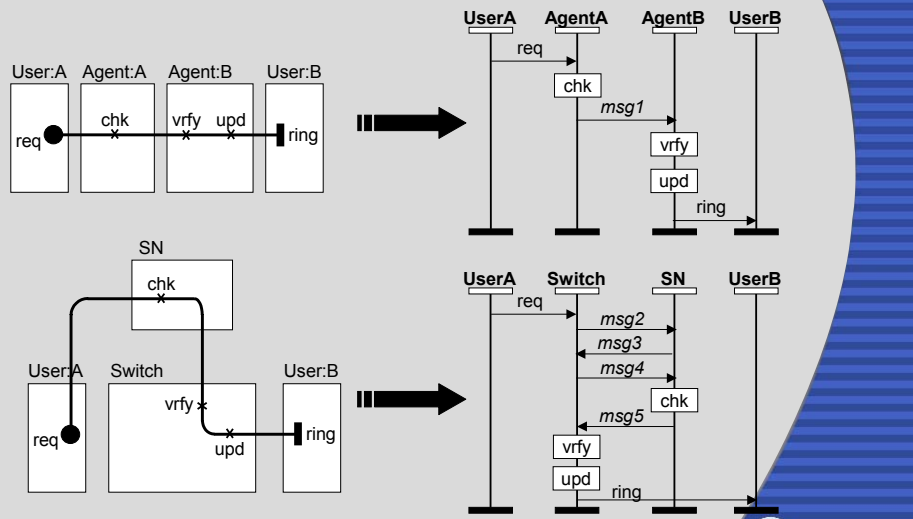
◆ Generation of Message Sequence Charts From UCMs

Motivation for Transformation

- ◆ UCMs are good for... **(Stage 1)**
 - Describing multiple scenarios abstractly
 - For analysing architectural alternatives
- ◆ MSC & Sequence Diagrams are better for... **(Stage 2)**
 - Developing and presenting the details of interactions
 - Describing lengthy sequences of messages in scenarios
 - Providing access to well-developed methodologies and tools for analysis and synthesis
- ◆ UCM-to-MSC transformation helps to further bridge the gap between Stage 1 descriptions (**requirements**) and Stage 2 descriptions (**design**).

Stage 1 ... Requirements and Service Description, Stage 2 ... Message Sequence Information

Refining UCM with Messages

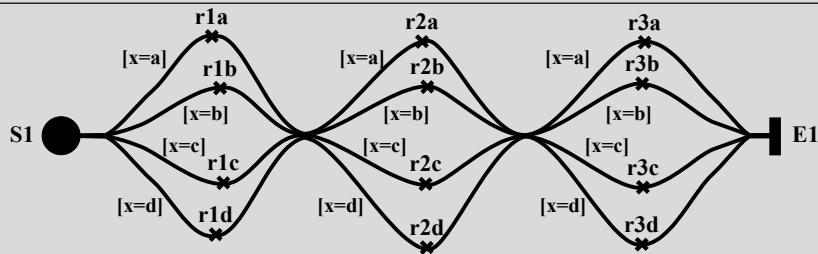


From UCM to MSC

- ◆ UCM component → MSC instance
- ◆ UCM path crossing from one component to another → abstract MSC message (“implements” causal flow)
- ◆ UCM start (or end) point → abstract MSC message
- ◆ UCM pre/post-condition → MSC condition
- ◆ UCM responsibility → MSC action
- ◆ UCM OR-fork or dynamic stub with multiple plug-ins → multiple basic MSCs
- ◆ UCM AND-fork → MSC parallel inline box
- ◆ UCM loop → MSC loop box
- ◆ UCM timer → MSC timer



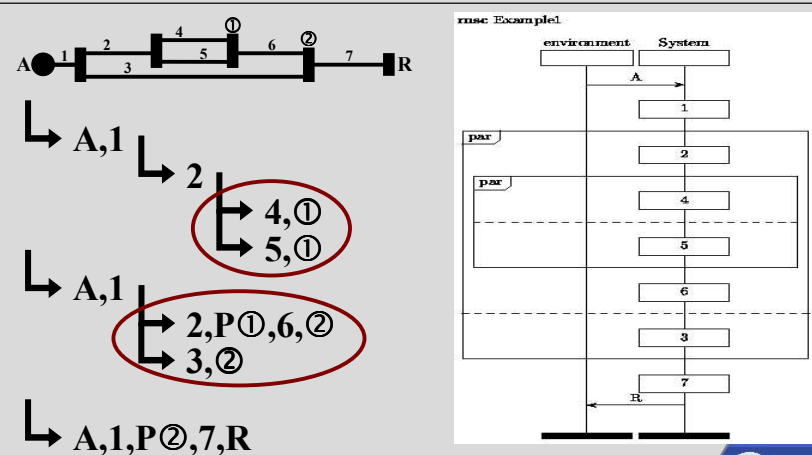
Need for Path Data Model



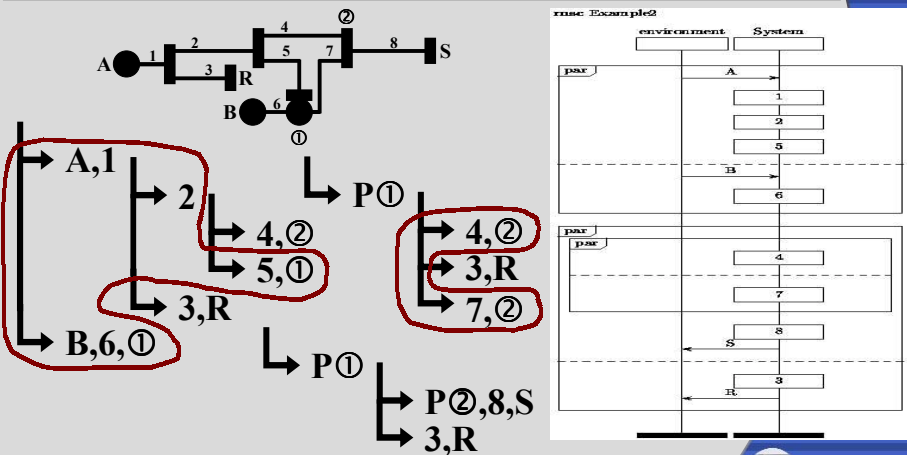
- ◆ 64 potential combination of end-to-end paths
- ◆ Reduced to 4 when conditions are taken into account
- ◆ Similar problem with combinations of plug-ins in dynamic stubs (e.g. embedded or in sequence)
- ◆ Path data model can help identify specific scenarios



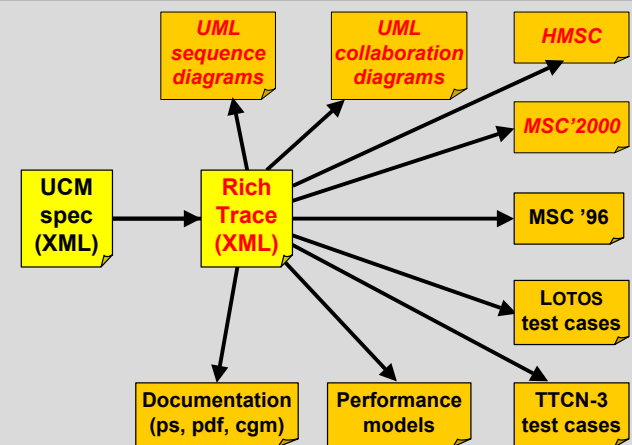
MSC Generation - Example 1



MSC Generation - Example II



Why Stop at MSCs?



Key Points - MSC Generation

- ◆ Much value in a tool-supported translation
 - Effortless (push of a button)
 - MSCs in-sync with UCMs, forward traceability
 - Basis for further refinement
 - ◆ Synthetic abstract message may be refined into more concrete protocol messages...
 - Help to bridge the requirements/design gap
- ◆ Need for path data model and scenario specifications

Main References

- ◆ Web site: <http://www.UseCaseMaps.org/>
- Buhr, R.J.A., *Use Case Maps as Architectural Entities for Complex Systems*, In: Transactions on Software Engineering, IEEE, Vol. 24, No. 12, December 1998, pp. 1131-1155.
- Buhr, R.J.A. and Casselman, R.S., *Use CASE Maps for Object-Oriented Systems*, Prentice Hall, 1996.
- Miga, A., Amyot, D., Bordeleau, F., Cameron, D., and Woodside, M., *Deriving Message Sequence Charts from Use Case Maps Scenario Specifications*, 10th SDL Forum, Copenhagen, Denmark, June 2001.
- Mussbacher, G. and Amyot, D., *A Collection of Patterns for Use Case Maps*, In: First Latin American Conference on Pattern Languages of Programming (SugarLoafPLoP 2001), Rio de Janeiro, Brazil, October 2001.