Use Case Maps as a Feature Description Notation

Daniel Amyot

Mitel Corporation, Kanata, Canada, and
School of Information Technology and Engineering, University of Ottawa
150 Louis-Pasteur, Ottawa, Ontario, K1N 6N5, Canada
Email: damyot@site.uottawa.ca
Web: http://www.UseCaseMaps.org

Abstract. We propose Use Case Maps (UCMs) as a notation for describing features. UCMs capture functional requirements in terms of causal scenarios bound to underlying abstract components. This particular view proved very useful in the description of a wide range of reactive and telecommunications systems. This paper presents some of the most interesting constructs and benefits of the notation in relation to a question on a User Requirements Notation recently approved by ITU-T Study Group 10, which will lead to a new Recommendation by 2003. Tool support, current research on UCMs, and related notations are also discussed.

1 Introduction

The modeling of reactive systems requires an early emphasis on behavioral aspects such as interactions between the system and the external world (including the users), on the cause-to-effect relationships among these interactions, and on intermediate activities performed by the system. Scenarios are particularly good at representing such aspects so that various stakeholders can understand them.

Owing to their distributed and critical nature, telecommunications systems are representative of complex reactive systems. Emerging telecommunications services and features require industries and standardization bodies (ANSI, ETSI, ISO, ITU, TIA, IETF, etc.) to describe and design increasingly complex functionalities, architectures, and protocols. This is especially true of wireless systems, where the mobility of users brings an additional dimension of complexity. Recent and upcoming technologies based on agents, XML, and IP, which involve complex and sometimes unpredictable policy-driven negotiations between communicating entities, also raise new modeling issues as protocols and entities become more dynamic in nature and evolve at run time.

The design and standardization of telecommunication systems and features results from a design process frequently comprised of three major stages, shown in Figure 1. At stage 1, features are first described from the user's point of view in prose form, with use cases, and with tables. The focus of the second

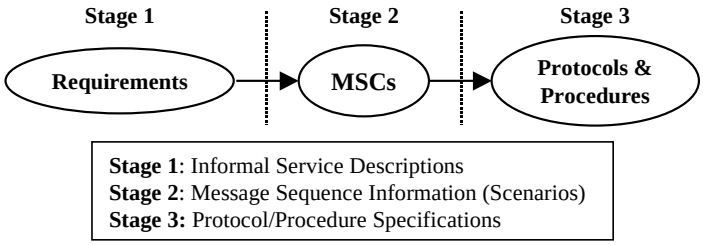


Fig. 1. Three-stage methodology

stage is on control flows between the different entities involved, represented using sequence diagrams or *Message Sequence Charts* (MSCs [22]). Finally, stage 3 aims to provide (informal) specifications of protocols and procedures. Formal specifications are sometimes provided (e.g. in SDL [21]), but overall they still suffer from a low penetration [10,17], especially in North-America [2,18]. ITU-T developed this three-stage methodology two decades ago to describe services and protocols for ISDN. Naturally, such descriptions emphasize the reactive and behavioral nature of telecommunications systems. In this methodology, scenarios are often used as a means to model system functionalities and interactions between the entities such that different stakeholders may understand their general intent as well as technical details.

Due to the inherent complexity and scale of emerging telecommunications features, special attention has to be brought to the early stages of the design process. The focus should be on system and functional views rather than on details belonging to a lower level of abstraction, or to later stages in this process. Many ITU-T members recognize the need to improve such three-stage process in order to cope with the new realities cited above. In particular, Study Group 10, which is responsible for the evolution of standards such as MSC, SDL, and TTCN, recently approved a question for study during the next ITU-T Study Period to develop a User Requirements Notation (URN) based on scenarios. The objective of the question is to develop a new Recommendation by the year 2003 [25].

This question focuses on what notation may be developed to complement MSCs, SDL and UML in capturing user requirements in the early stages of design when very little design detail is available. Such notation should be able to describe features as user requirement scenarios without any reference to states, messages or system components. Reusability of scenarios across a wide range of architectures is needed with allocation of scenario responsibilities to architectural components. The notation should enable simpler modeling of dynamic systems, early performance analysis at the requirements level, and early detection of undesirable interactions among features or scenarios.

While UML activity diagrams provide some capability in this area [27], a requirements notation with dynamic refinement capability and better allo-

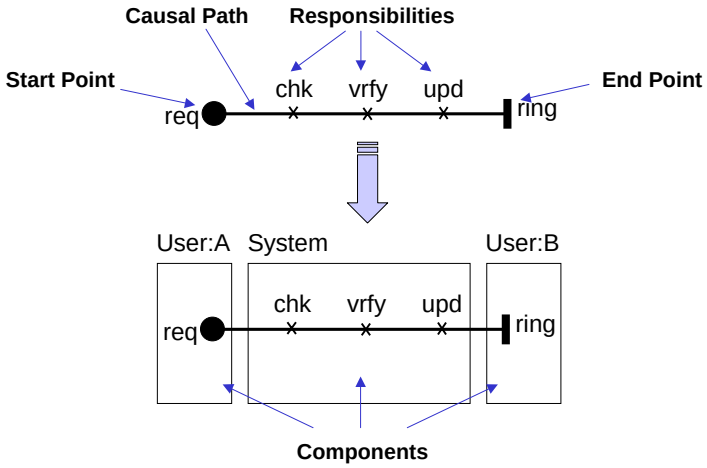


Fig. 2. Simple Use Case Map

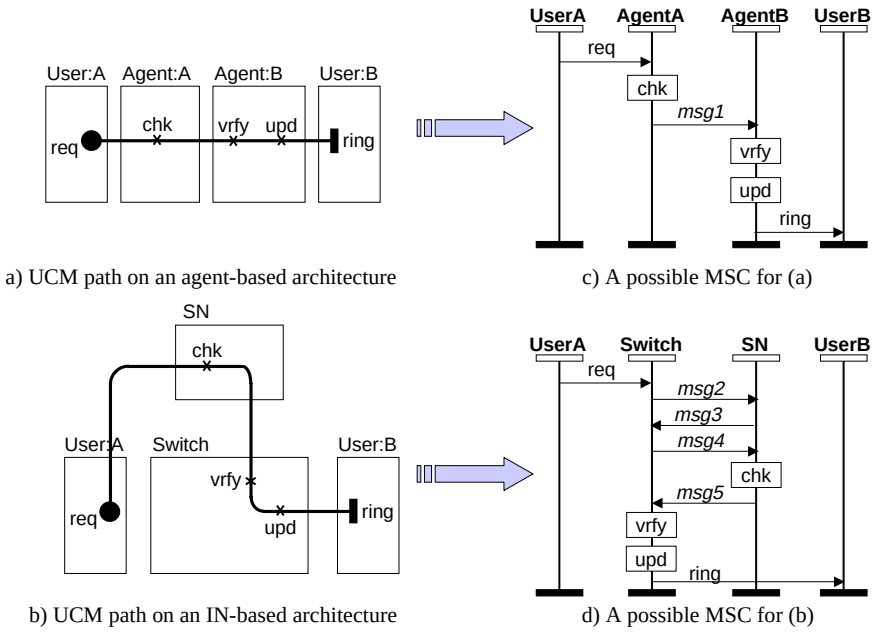
cation of scenario responsibilities to architectural components is required [7]. The Use Case Map notation offers such capabilities. The basics of this notation and its main benefits are illustrated in Section 2. Section 3 presents various domains of application of UCMs as a feature description notation followed by an overview of current tool support, research directions, related notations, and formalization issues. A brief conclusion follows in Section 4.

2 Use Case Maps

2.1 Basic Notational Elements

The *Use Case Map* (UCM) notation [13,14] is used for describing *causal relationships* between *responsibilities*, which may potentially be bound to underlying organizational structures of abstract *components* (see Figure 2). Responsibilities are generic and can represent actions, activities, operations, tasks to perform, and so on. Components are also generic and can represent software entities (objects, processes, databases, servers, functional entities, network entities, etc.) as well as non-software entities (e.g. users, actors, processors). The relationships are said to be causal because they involve concurrency and partial orderings of activities and because they link causes (e.g., preconditions and triggering events) to effects (e.g. postconditions and resulting events). In a way, UCMs show related use cases in a map-like diagram.

The scenario in Figure 2 represents a simplified call connection initiated by user A on req. The system first checks whether the call should be allowed (chk) and then verifies whether the called party is busy or idle (vrfy). In both cases here, we assume that the call request goes through. Then, the system status is updated (upd) and a resulting ringing event occurs at B's side (ring).



a) UCM path on an agent-based architecture

c) A possible MSC for (a)

b) UCM path on an IN-based architecture

d) A possible MSC for (b)

Fig. 3. UCM path bound to two different component structures, and potential MSCs

2.2 UCMs, Architectures and Messages

UCM paths and their responsibilities are useful for describing features at an early stage in the design cycle (e.g. at stage 1), even when no component is involved (e.g. Figure 5). It is then possible to bind UCM paths to a suitable structure of components, which leads to a visual integration of scenarios and architecture in a single view. UCM scenario paths possess a high degree of reusability and they lead to behavioral patterns that can be utilized across a wide range of applications. For instance, the UCM path from Figure 2 can be bound to alternative architectures therefore enabling early architectural reasoning. Figure 3(a) uses an agent-based architecture whereas Figure 3(b) uses a more conventional architecture based on Intelligent Networks (IN).

UCM paths are also more likely to survive evolutions and other modifications to the underlying architecture than scenarios described in terms of message exchanges or interactions between components. For instance, Figure 3(c) is an MSC capturing the scenario from Figure 3(a) in terms of message exchanges. This is a straightforward interpretation with artificial messages (in italic characters). In this system, each user can communicate with its agent only, and agents can communicate with other agents. Other such MSCs could possibly be derived from the same scenario path.

Figure 3(d) is a potential MSC extracted from the same scenario path, but this time bound to an IN-based architecture. Complex protocols or negotiation mechanisms could be involved between the switch and the service node, hence resulting in multiple messages. Communication constraints (not shown here) could also prevent users from communicating directly with service nodes; therefore the switch needs to be involved as a relay to refine the causal relationship between req and chk.

When extracting MSC-like scenarios from informal requirements, as it is often done in the three-stage methodology shown in Figure 1, many design decisions become buried in the details of the scenarios. For instance, the Wireless Intelligent Network (WIN) standard attempts to use the IN reference model but, due to legacy descriptions of former versions of the ANSI-41 North-American wireless standard, it only provides MSC scenarios where the components represent network elements belonging to the physical plane [9]. Design decisions such as the allocation to UCM responsibilities to functional entities (the logical components in the distributed functional plane) and the allocation of functional entities to network entities are lost [3]. Since this standard does not impose a specific mapping of functional entities to network entities, different vendors who build network entities may use different mappings (and this is actually happening). Designers must reverse-engineer information and scenarios that would be explicit in a UCM view where responsibilities and other constructs are bound to functional entities. This delays the design and implementation phases and leads to multiple interoperability problems. Such problems are unfortunately common in standards.

By using a UCM view, many issues related to messages, protocols, communication constraints, and structural evolutions (e.g. from one version of the structure to the next) can be abstracted from, and the focus can be put on intended functionalities and on reusable causal scenarios in their structural context.

2.3 UCMs and Scenario Integration

UCMs can also help structuring and integrating scenarios in various ways, e.g. sequentially, as alternatives (with OR-forks/joins) or concurrently (with AND-forks/joins). However, one of the most interesting constructs for scenario integration is certainly the *dynamic stub*, shown as a dashed diamond in Figure 4.

While static stubs (not shown here) contain only one sub-map (called *plug-in*), dynamic stubs may contain multiple sub-maps whose selection can be determined at run-time according to a *selection policy*. Such a policy can make use of preconditions, assertions, run-time information, composition operators, etc. in order to select the plug-in(s) to use. Selection policies are described with a (formal or informal) language suitable for the context where they are used. The plug-in maps are sub-maps that describe locally how a

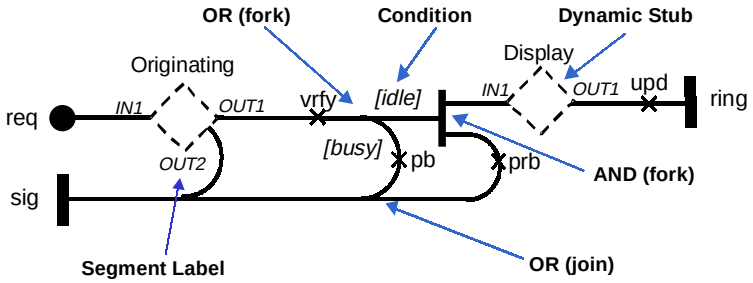


Fig. 4. Basic Call UCM with two dynamic stubs

feature modifies the basic behavior. Multiple levels of stubs and plug-ins can be used.

Figure 5 shows four UCMs. The top-level UCM is the Basic Call of Figure 4, which contains two dynamic stubs. Each of these stubs includes a DEFAULT plug-in (which happens to be the same in both cases) that represents how the basic call reacts in the absence of other features.

The Originating stub has two plug-ins:

- *Originating Call Screening* (OCS), which checks whether the call should be denied or allowed (chk). When denied, an appropriate event is prepared (pd) and signaled (sig). Its *binding relationship*, which connects the input/output segments of a stub to the start/end points of its plug-in, is $\{\langle IN1, in1\rangle, \langle OUT1, out1\rangle, \langle OUT2, out2\rangle\}$.
- TEENLINE, which denies the call provided that the request is made during a specific time interval and that the personal identification number (PIN) provided is invalid or not entered in a timely manner. The zigzag path leaving the timer represents a timeout path. The binding relationship for this feature is also $\{\langle IN1, in1\rangle, \langle OUT1, out1\rangle, \langle OUT2, out2\rangle\}$.

The Display stub contains only one feature:

- *Call Number Delivery* (CND), which displays the number of the originating party (disp) concurrently with the rest of the scenario (update and ringing). The binding relationship is $\{\langle IN1, in1\rangle, \langle OUT1, out1\rangle\}$.

Adding features to such UCM collections is often achieved by creating new plug-ins for the existing stubs, or by adding new stubs containing either new plug-ins or instances of existing plug-ins. In all cases, the selection policies need to be updated appropriately.

Stubs and selection policies tend to localize the places on scenarios where undesirable feature interactions can occur [4,6,26], hence simplifying the analysis. They can also be used to specify priorities of some features over others. For instance, TEENLINE could be given a sequential priority over OCS in the Originating stub. Many spurious interactions between features are hence avoided by structuring and integrating the scenarios in the proper context.

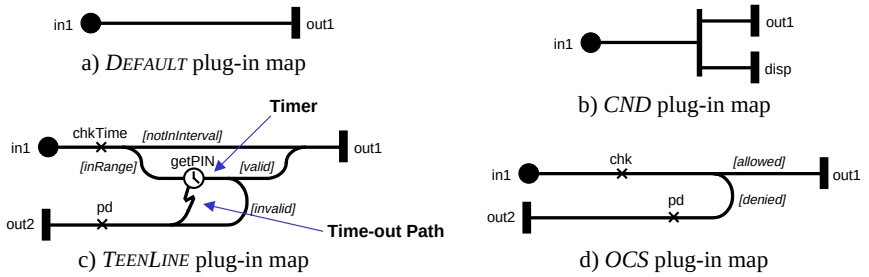


Fig. 5. Basic Call UCM with its four plug-ins

2.4 Performance and Agent-Oriented Annotations

The UCM core notation has been extended over the years to cover different fields of applications such as performance analysis at the requirements level (with *timestamps*) [30,31] and agent-oriented design (with *goal tags*). These areas are also recognized by ITU-T Study Group 10 as relevant to the description of features.

Timestamps are located on UCM paths in order to identify various performance constraints and response time requirements. For example, Figure 6 shows two timestamps attached to a UCM. Response time requirements (expected response time, probability, etc.) can be defined between pairs of timestamps (e.g., T1 and T2) at the scenario path level. In order to generate useful performance simulation models, other notation elements can be annotated, for instance start points with arrival characteristics (exponential, deterministic, uniform, Erlang, etc.) and responsibilities (associated data store, performed service requests, etc.).

Goal tags are used to associate high-level goals and intentions to UCM scenario paths. This is particularly relevant to the description of agent systems, where distributed goals are intended to be achieved by collaborating agents. Figure 7 shows the symbol used to represent goal tags. Similar to

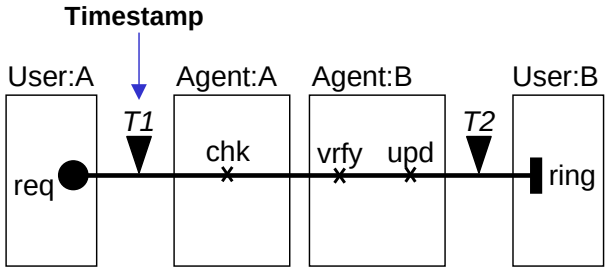


Fig. 6. Performance annotations with timestamps

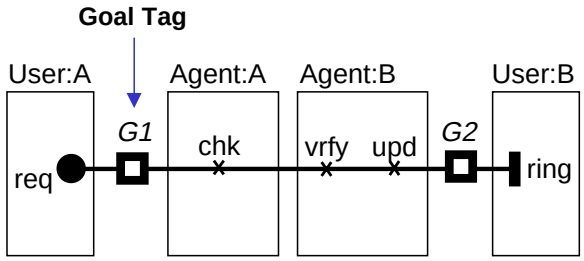


Fig. 7. Agent-oriented goal annotations with goal tags

timestamps, these tags can be coupled in pairs to describe goals and their pre/post-conditions.

2.5 Additional Notation Elements

The UCM notation contains many additional elements, including:

- *Component types and attributes:* types can be used to visually distinguish active components (e.g. processes), passive components, containers, agents, etc. A component may also possess several attributes (e.g. replication factor and mutual exclusion).
- *Dynamic components and dynamic responsibilities:* dynamic stubs show how behavior patterns can evolve at run time. A structure of component can also evolve at run time through the use of dynamic components, which represent, in a static way, roles that can be filled by actual instances of components at different times. Dynamic responsibilities are used to create, delete, store, retrieve, or move dynamic components.
- *Path interactions:* UCM paths can also be combined through the use of synchronous or asynchronous path interactions, which often involve a special point on a path called *waiting place*.
- *Aborts:* a scenario path can abort the evolution of another path through this construct. This can also be used to model exceptions.
- *Failure points:* failure points represent explicit places on a UCM path where a scenario could stop before reaching its end point. This is useful when robustness needs to be addressed at a high level of abstraction.

Although these advanced constructs are often useful for the description of features, they will not be discussed further in this paper. The interested reader can however consult the UCM Quick Reference Guide (Appendix A) and the UCM virtual library of the *UCM User Group* for more information [34].

3 UCMs as a Feature Description Notation

Use Case Maps have a number of properties that satisfy many of the requirements described in Section 1: scenarios can be mapped to different architectures, variations of run-time behavior and structures can be expressed, and scenarios can be structured and integrated incrementally in a way that facilitates the early detection of undesirable interactions and the early evaluation of performance. Performance becomes a property of paths, rather than just a non-functional property of a whole system, as it is often considered to be [14,30,31]. Macroscopic behavior patterns are described independently of details belonging to connections between components (e.g. messages), and large-scale dynamic situations and issues can be made visible at a glance.

The UCM notation is currently applicable to a wide range of areas, it is already supported by a tool, and it is the topic of several research projects. It also possess interesting characteristics that are difficult to find all at once in other notations for describing features. This will be discussed briefly in the following sections.

3.1 Areas of Application

Use Case Maps are well suited for describing requirements and high-level designs of reactive (event-driven) and distributed systems and their features. UCMs have a history of applications to the description of features and telecommunications systems of different natures (e.g. [3–6,8]), to the avoidance and detection of undesirable interactions between scenarios or features (e.g. [4,6,15,26]) and to early performance analysis (e.g. [30,31]).

UCMs are however not restricted to telecommunications systems. They are also being used to describe systems and features from various domains such as (in no particular order) airline reservation applications, elevators, railway control systems, agent systems, network management applications, Web applications, graphical user interfaces, drawing packages, multimedia applications, banking applications, object-oriented frameworks, "work patterns" of software engineers, and many others [34].

3.2 UCM Navigator Tool

The UCM notation is also supported by a tool: the *UCM Navigator* [24]. This tool has been developed at Carleton University (Ottawa) in order to help drawing correct UCMs. Among other features, this tool supports the path and component notations found in Appendix A, and it maintains various bindings (plug-ins to stubs, responsibilities to components, sub-components to components, etc.). Also, it allows users to navigate much like a Web browser, and to visit and edit the plug-ins related to stubs of all levels (Figure 8).

The UCM Navigator is transformation-based and it ensures that UCMs are syntactically correct by construction. The tool saves, loads, exports and

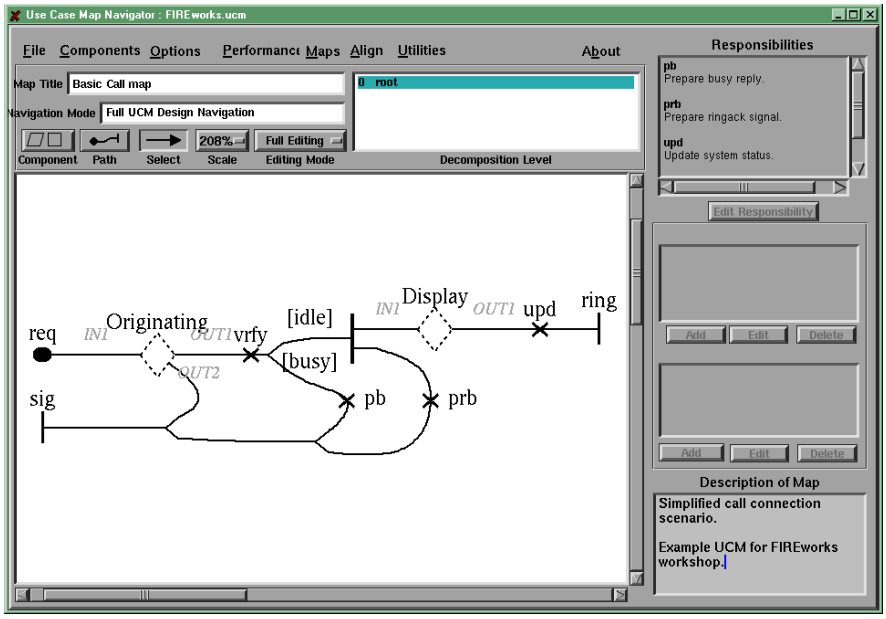


Fig. 8. UCM Navigator screen shot

imports UCM as XML files, which are valid according to a UCM Document Type Definition (DTD) [34]. This DTD describes the current formal definition of UCMs, which is based on hypergraphs, and the UCM Navigator ensures that static semantic rules and other constraints are satisfied. No formal dynamic semantics currently exists for the UCM notation. The tool can also export UCM figures in three formats: Encapsulated PostScript (EPS), Maker Interchange Format (MIF), and Computer Graphics Metafile (CGM). Flexible reports can be generated as PostScript files ready to be transformed into hyperlinked and indexed PDF files. Multiple platforms are currently supported: Solaris, Linux (Intel and Sparc), HP/UX, and Windows (95, 98, 2000 and NT).

3.3 Current UCM Research

Many researchers are developing links between Use Case Maps and other notations and languages, something that is necessary in order to produce concrete implementations and products from features and other functional requirements captured as UCMs. Among others, UCMs are currently being mapped to:

- (H)MSCs or UML sequence diagrams (e.g. Figure 3) [6,12]. This generation is in the process of being formalized and automated in the UCM Navigator tool.

- Hierarchical state machines such as UML statechart diagrams or ROOM-charts [12].
- LOTOS models, which enable formal validation, verification, and detection of undesirable interactions [4–6,8].
- SDL models, which also enable formal validation [29].
- UML and UML-RT models [7,12].
- Other research projects include the generation of performance models (e.g. in Layered Queuing Networks-LQNs), of abstract test cases for functional testing (e.g. in the Tree and Tabular Combined Notation-TTCN), and of programs in agent-oriented languages.

The notation is also evolving under the guidance of a newly created *UCM Working Group* composed of members from industry and universities. In particular, this group intends to present a contribution to ITU-T proposing Use Case Maps as an appropriate notation to capture *functional* requirements (a companion notation such as the one in the NFR framework [16] would have to capture non-functional requirements). UCMs would hence represent part of the answer to the User Requirements Notation (URN) question.

The multiple connections between UCMs and other languages enable the creation of many design trajectories relevant to telecommunications systems, as suggested in the introduction. In particular, we envision the following trajectory, inspired from [2,3,12,18]: requirements capture and architectural reasoning is done with UCM/URN (stage 1), which are first transformed into MSCs or interaction diagrams (stage 2), then into state machines in SDL or UML-RT statechart diagrams (stage 3), and finally into concrete implementations (possibly through semi-automated code generation). Inspection, validation, verification, performance analysis, interaction detection, and test generation can be performed to various degrees at all stages.

3.4 UCMs and Other Notations for Describing Features

Features can be described in a number of ways, for instance with goals, logical properties, and scenarios. These approaches are not necessarily mutually exclusive (e.g. goals can be associated to UCM scenarios, as seen in Section 2.4). Goals and properties are usually large-grained, declarative, and cover more situations than scenarios, which are more operational and which capture partial and non-exhaustive views of the system. However, the discovery and structuring of goals and properties is not an easy task, whereas the construction of scenarios is often simpler [28]. Scenarios are also more in line with current practices in ITU-T and other standardization bodies. According to many people in such organizations, it is better to find a practical notation that will improve the current situation and that will be used than to aim for the best theoretical solution, which is unlikely to be used at all [18].

Scenarios can also be textual rather than graphical (e.g. Jacobson's use cases [23]). Although textual scenarios can adequately describe requirements

in many situations, especially when appropriate construction guidelines are provided [11], these scenarios are still very much linear in nature. Graphical notations such as UCMs enable the description of scenarios in two dimensions as well as the compact representation of multiple scenarios.

Still, many graphical scenario notations could be considered as good candidates for URN in the context of feature descriptions. Here is a brief comparison between UCMs and four such notations, namely MSCs, Chisel diagrams, Petri Nets, and UML activity diagrams:

- *Message Sequence Charts (MSCs)*: as shown in Section 2, MSCs and similar interaction diagrams suffer from a premature commitment to messages and components. This is not always appropriate for the early stages of feature definitions because details irrelevant to the requirements level must be considered, and this in turn hides the original intent of the feature. UCMs abstract from messages and improve the reusability of scenarios across component architectures.
- *Chisel diagrams*: Aho et al. have performed empirical studies with telecommunication engineers to create the Chisel notation [1]. Chisel diagrams are trees whose branches represent sequences of (synchronous) events taking place on component interfaces. Nodes describe these events (multiple concurrent events can take place in one node) and arcs, which can be guarded by conditions, link the events in causal sequences. Chisel diagrams describe multiple abstract scenarios, but like MSCs they focus on inter-component interactions even if the components are hidden from this particular view. UCMs can abstract from these events, and UCMs also support advanced concepts such as dynamic stubs, which have no equivalent in the Chisel notation. Note that similar to the UCM-LOTOS translation mentioned in Section 3.3, a Chisel-LOTOS mapping has been defined and automated by Turner [33].
- *Petri Nets (PNs)*: these are abstract machines used to describe system behaviour visually with a directed graph containing two types of nodes: places and transitions. Basic PNs suffer from a state explosion problem when complex problems are addressed. Various extensions for data types and modules, which help to cope with this problem, are currently being standardized as High-Level Petri Nets [20]. The main benefits of PNs over UCMs are their formality and executability. However, their use as a requirements notation for features (if such thing exists) is still remote from the current ITU-T practice, and PNs also lack concepts such as dynamic stubs and a view that combines visually behavioral scenarios and component architectures. PNs could however represent a candidate language for formalizing the UCM dynamic semantics.
- *UML activity diagrams*: probably the next best alternative to UCMs for URN, activity diagrams share many characteristics with UCMs [27]: focus on sequences of actions, guarded alternatives, and concurrency; start and end points from each notation have a similar purpose; and complex

activities can be refined. However, activity diagrams are usually unrelated to components (*swimlanes*, which could be seen as enabling bindings of activities to components, provide functional grouping only), and their sub-diagrams are less flexible and less powerful than UCM stubs. Activity diagrams could however be evolved to support these attractive UCM concepts [7], which would then help satisfy ITU-T's goal of linking URN to MSC, SDL, and UML.

3.5 On the Formalization of UCMs

The UCM notation enjoys an enthusiastic community of users and it has been used successfully as a feature description notation in the domains of telecommunications and other reactive systems. Several users however complain about the lack of formal foundations behind the notation, and work is being done to address this situation (e.g. [7]). We like to think of UCMs as a back-of-envelope notation that offers an attractive level of abstraction for describing feature requirements at early stages, without getting dragged into low-level details. An unfortunate consequence is that UCMs are open to misinterpretation as much as any other non-formal notation. Conventions, standard styles, and patterns can help to cope with this issue to some extent, but formalization is required as well. However, finding the appropriate degree of formalization, which would not sacrifice the appealing and semi-formal characteristics of UCMs, remains an issue. The URN effort represents a great opportunity for finding a practical solution to this problem. Formalization could also be done by integrating UCMs to UML, for instance by adapting and improving the activity diagrams notation and semantics. High-level Petri Nets could also be used as a semantic model. In any case, UCMs possess several attractive constructs and concepts that should be seriously considered in any full-fledged feature description language.

4 Conclusion

Scenarios are a popular and practical approach to the design of reactive systems. The UCM notation enables the early description of features in terms of causal scenarios bound to underlying abstract components. This paper illustrates the core constructs of the notation, its main benefits, and its place in common design trajectories. It also shows that the UCM notation satisfies many of the needs expressed in ITU-T's question for study about a User Requirements Notation.

The UCM notation is evolving and is the target of many research and development projects. Industrial partners appreciate this notation because it allows for senior designers, system architects and product managers, who possess good knowledge of the domain and of existing systems, to "work again", i.e. they can communicate their knowledge efficiently to junior designers, who then take care of the specifics. Senior people do not need to

know all the details surrounding recent or emerging technologies to describe desired features in a way that design teams can understand and refine. We are hence very confident in the usefulness of the notation and of the level of abstraction it addresses.

Acknowledgements

This work results from collaborations and discussions with many colleagues, co-authors, and partners from the University of Ottawa's LOTOS group, the UCM User Group, Mitel Corporation and Nortel Networks. I am also indebted towards my supervisor, Luigi Logrippo, and my former co-supervisor, Ray Buhr, for their encouragement. Finally, I would like to thank Communications and Information Technology Ontario (CITO), the Natural Sciences and Engineering Research Council of Canada (NSERC), Mitel Corporation and Nortel Networks for their financial support.

References

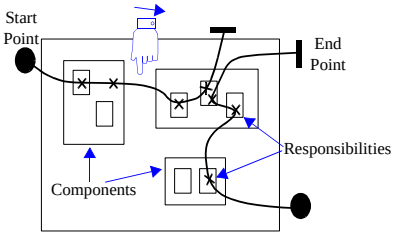
1. Aho, A., Gallagher, S., Griffeth, N., Scheel, C. and Swayne, D.: "Sculptor with Chisel: Requirements Engineering for Communications Services". In: *Fifth International Workshop on Feature Interactions in Telecommunications and Software Systems (FIW'98)*, Lund, Sweden, October 1998. IOS Press, Amsterdam, 45–63.
2. Amyot, D., Andrade, R., Logrippo, L., Sincennes, J., and Yi, Z.: "Formal methods for mobility standards". In: *Proc. of the 1999 IEEE Emerging Technologies Symposium on Wireless Communications and Systems*, Richardson, Texas, USA, April 1999.
3. Amyot, D. and Andrade, R.: "Description of Wireless Intelligent Network Services with Use Case Maps". In: *SBRC'99, 17th Brazilian Symposium on Computer Networks*, Salvador, Brazil, May 1999.
4. Amyot, D., Buhr, R. J. A., Gray, T., and Logrippo, L.: "Use Case Maps for the Capture and Validation of Distributed Systems Requirements". In: *RE'99, Fourth IEEE International Symposium on Requirements Engineering*, Limerick, Ireland, June 1999, 44–53.
5. Amyot, D. and Logrippo, L.: "Use Case Maps and LOTOS for the Prototyping and Validation of a Mobile Group Call System". In: *Computer Communication*, 23(12), 1135–1157, May 2000.
6. Amyot, D., Charfi, L., Gorse, N., Gray, T., Logrippo, L., Sincennes, J., Stepien, B. and Ware, T.: "Feature Description and Feature Interaction Analysis with Use Case Maps and LOTOS". In: *Sixth International Workshop on Feature Interactions in Telecommunications and Software Systems (FIW'00)*, Glasgow, Scotland, UK, May 2000.
7. Amyot, D. and Mussbacher, G.: "On the Extension of UML with Use Case Maps Concepts". In: *<<UML>>2000, 3rd International Conference on the Unified Modeling Language*, York, UK, October 2000.

8. Andrade, R.: "Applying Use Case Maps and Formal Methods to the Development of Wireless Mobile ATM Networks". In: *Lfm2000: The Fifth NASA Langley Formal Methods Workshop*, Williamsburg, Virginia, USA, June 2000.
9. ANSI/TIA/EIA: *ANSI 771, Wireless Intelligent Networks (WIN). Additions and modifications to ANSI-41 (Phase 1)*. TR-45.2.2.4, December 1998.
10. Ardis, M. A., Chaves, J. A., Jagadeesan, L. J., Mataga, P., Puchol, C., Staskauskas, M. G., and Olnhausen, J. V.: "A Framework for Evaluating Specification Methods for Reactive Systems - Experience Report". In: *Transactions on Software Engineering*, IEEE, 22 (6), 1996, 378–389.
11. Ben Achour, C., Rolland, C. Maiden, N. A. M. and Souveyet, C.: "Guiding Use Case Authoring: Results of an Empirical Study". In: *RE'99, Fourth IEEE International Symposium on Requirements Engineering*, Limerick, Ireland, June 1999, 36–43.
12. Bordeleau, F.: *A Systematic and Traceable Progression from Scenario Models to Communicating Hierarchical Finite State Machines*. Ph.D. thesis, SCS, Carleton University, Ottawa, Canada, August 1999.
13. Buhr, R. J. A. and Casselman, R. S.: *Use Case Maps for Object-Oriented Systems*, Prentice-Hall, 1996.
14. Buhr, R. J. A.: "Use Case Maps as Architectural Entities for Complex Systems". In: *Transactions on Software Engineering*, IEEE, December 1998, 1131–1155.
15. Buhr, R. J. A., Amyot, D., Elammari, M., Quesnel, D., Gray, T., and Mankovski, S.: "Feature-Interaction Visualization and Resolution in an Agent Environment". In: *Fifth International Workshop on Feature Interactions in Telecommunications and Software Systems (FIW'98)*, Lund, Sweden, October 1998. IOS Press, Amsterdam, 135–149.
16. Chung, L., Nixon, B. A., Yu, E. and Mylopoulos, J.: *Non-Functional Requirements in Software Engineering*. Kluwer Academic Publishers, 2000.
17. Craigen, D., Gerhart, S., and Ralston, T.: *Industrial applications of formal methods to model, design, and analyze computer systems: an international survey*. Noyes Data Corporation (Publisher), USA, 1994.
18. Hodges, J. and Visser, J.: "Accelerating Wireless Intelligent Network Standards Through Formal Techniques". In: *IEEE 1999 Vehicular Technology Conference (VTC99)*, Houston, USA, 1999.
19. ISO, Information Processing Systems, OSI: LOTOS - *A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour*. IS 8807, Geneva, Switzerland, 1989.
20. ISO/IEC: *High Level Petri Net Standard*, DIS 15909, JTC 1/SC 7, Geneva, Switzerland, 1999.
21. ITU-T: *Recommendation Z.100, Specification and Description Language (SDL)*. Geneva, Switzerland, 2000.
22. ITU-T: *Recommendation Z.120, Message Sequence Chart (MSC)*. Geneva, Switzerland, 2000.
23. Jacobson, I.: "The Use Case Construct in Object-Oriented Software Engineering". In: John M. Carroll (ed.), *Scenario-Based Design: Envisioning Work and Technology in System Development*. John Wiley and Sons, 1995, 309–336.
24. Miga, A.: *Application of Use Case Maps to System Design with Tool Support*. M.Eng. thesis, Dept. of Systems and Computer Engineering, Carleton University, Ottawa, Canada, October 1998.

25. Monkewich, O.: *New Question 12: URN: User Requirements Notation*. Canadian contribution to ITU-T Study Group 10, COM10-D56, November 1999.
26. Nakamura, M., Kikuno, T., Hassine, J., and Logrippo, L.: "Feature Interaction Filtering with Use Case Maps at Requirements Stage". In: *Sixth International Workshop on Feature Interactions in Telecommunications and Software Systems (FIW'00)*, Glasgow, Scotland, UK, May 2000.
27. Object Management Group: *Unified Modeling Language Specification, Version 1.3*. June 1999.
28. Rolland, C., Souveyet, C. and Ben Achour, C.: "Guiding Goal Modelling using Scenarios". In: *IEEE Transactions on Software Engineering, Special Issue on Scenario Management*. Vol. 24, No. 12, December 1998.
29. Sales, I. and Probert, R. L.: "From High-Level Behaviour to High-Level Design: Use Case Maps to Specification and Description Language". In: *SBRC'2000, 18th Brazilian Symposium on Computer Networks*, Belo Horizonte, Brazil, May 2000.
30. Scratchley, W. C. and Woodside, C. M.: "Evaluating Concurrency Options in Software Specifications". In: *MASCOTS'99, Seventh International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems*, College Park, MD, USA, October 1999, 330–338.
31. Scratchley, W. C.: *Evaluation and Diagnosis of Concurrency Architectures*. Ph.D. thesis, Dept. of Systems and Computer Engineering, Carleton University, Ottawa, Canada, June 2000.
32. Selic, B.: "Turning Clockwise: Using UML in the Real-Time Domain". In: *Communications of the ACM*, 42(10), October 1999, 46–54.
33. Turner, K. J.: "Formalising the Chisel Feature Notation". In: *Sixth International Workshop on Feature Interactions in Telecommunications and Software Systems (FIW'00)*, Glasgow, Scotland, UK, May 2000. IOS Press, Amsterdam, 241–256.
34. *Use Case Maps Web Page and UCM User Group*, March 1999. `http://www.UseCaseMaps.org`

N.B. Many of these papers are available in the UCM virtual library: `http://www.UseCaseMaps.org/pub/`

A UCM Quick Reference Guide

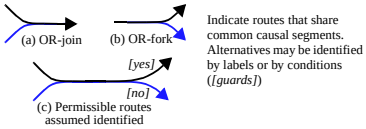


Imagine tracing a path through a system of objects to explain a causal sequence, leaving behind a visual signature. Use Case Maps capture such sequences. They are composed of:

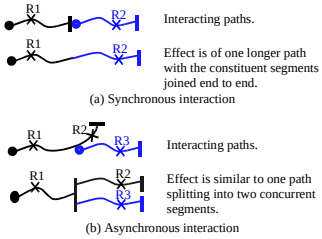
- **start points** (filled circles representing preconditions and/or triggering causes)
- causal chains of **responsibilities** (crosses, representing actions, tasks, or functions to be performed)
- and **end points** (bars representing postconditions and/or resulting effects).

The responsibilities can be bound to **components**, which are the entities or objects composing the system.

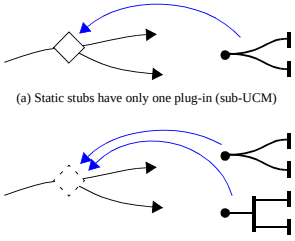
A1. Basic notation and interpretation



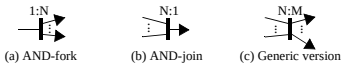
A2. Shared routes and **OR-forks/joins**.



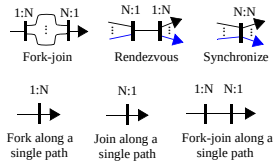
A3. Path interactions.



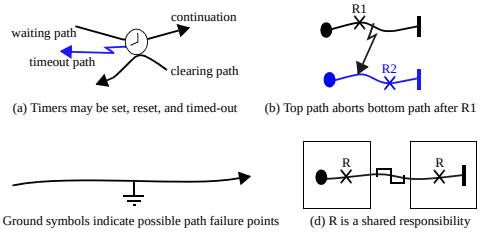
A6. **Stubs** and **plug-ins**.



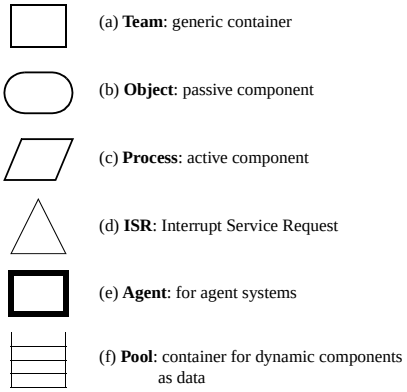
A4. Concurrent routes with **AND-forks/joins**.



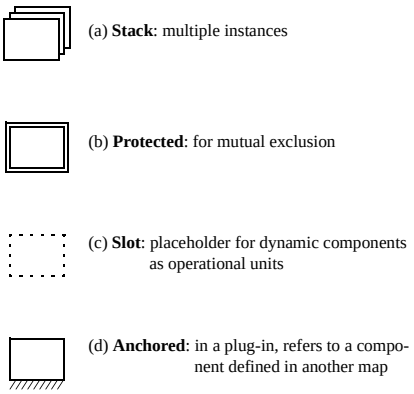
A5. Variations on AND-forks/joins.



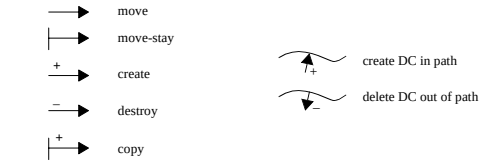
A7. **Timers, aborts**, **failures**, and **shared responsibilities**.



A8. Component types.

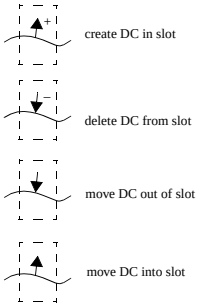


A9. Component attributes.

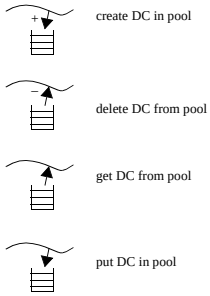


(a) Movement of DCs as data

b) Directly into or out of paths

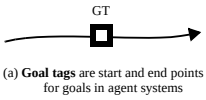


(c) Into or out of slots

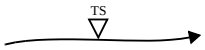


(d) Into or out of pools

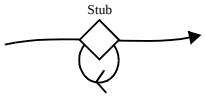
A10. Movement notation for **dynamic components** (DCs).



(a) **Goal tags** are start and end points for goals in agent systems



(b) **Timestamps** are start and end points for response time requirements



(c) **Direction arrows** can be used when path direction is ambiguous

A11. Notation extensions