On the Relationship between Use Case Maps and Message Sequence Charts

F. Bordeleau[1], D. Cameron[2]

1. School of Computer Science, Carleton University, Ottawa, Canada
emails: francis@scs.carleton.ca
2. Nortel Networks, Ottawa, Canada
email: dcameron@nortelnetworks.com

Abstract

The time it takes to develop telecommunication standards must be reduced to allow companies to get features to the market faster. In order to achieve this objective, a process based on the combination of a set of modeling techniques has been proposed by Nortel Networks in the context of the Wireless Intelligent Network (WIN) standard development. This process combines the use of standard notations like Message Sequence Charts (MSC), Specification and Description Language (SDL), Abstract Syntax Notation One (ASN.1), and Tree Tabular Combined Notation (TTCN). It also includes the use of Use Case Maps (UCM), a graphical high level scenario modeling technique, for stage one user requirements descriptions. UCM, which is not yet a standard, is going to be proposed as a candidate solution when ITU-T Study Group 10 meets later this year to consider Question 12/10 - URN: User Requirements Notation. In this paper, we introduce the UCM modeling technique, and discuss the relationship between UCM and MSC. Moreover, we define a transition that allows moving from UCM to MSC in a systematic and traceable manner.

1. Introduction

Telecommunication standards are becoming more and more important as the pressure to deliver the unified network to every corner of the globe grows. Delays are being met with increasing impatience.

We are involved in a line of research aimed at reducing the time it takes to develop telecommunication standards and thus reduce the time it takes to get features to market. This research has been commissioned by Nortel Networks people participating in the development of the Wireless Intelligent Network (WIN) standard. The WIN standard writers are feeling the pressure of operating companies who want product now to satisfy a growing demand from their customers for features. Our sponsors feel differing interpretations of text-based specifications is the principal reason behind the delays experienced in getting standards issued.

Our sponsors and their colleagues have proposed two notations in an effort to mediate interpretations in the WIN standard forum [6]. They have proposed and introduced Use Case Maps (UCMs) [4] for stage one user requirements descriptions and Message Sequence Charts (MSCs) [7] for stage two roaming information scenario descriptions. While MSCs is a well known and widely used notation, the UCM notation is relatively new. It has been developed by Ray Buhr,

now retired, of Carleton University and his students and colleagues and is the product of many years of research that focussed on the needs of product developers for ways to think about and express high-level system design. For information on UCM (notation, usage, and users) and tool support for UCMs, see [10].

Our sponsors believe that an executable and validatable standard will eliminate the interpretation delays and that an executable standard can be achieved using a pipeline of standardized design notations that have tool support. The move to standardize the Use Case Map notation is underway. UCMs will be proposed as a candidate solution when ITU-T Study Group 10 meets later this year to consider Question 12/10 - URN: User Requirements Notation.

Key to reducing time in this new way to make standards is a sequence of semi-automated transformations from one design notation into another at each stage in the pipeline. The order of transformations in the pipeline is Use Case Maps to Message Sequence Charts (MSCs) and MSCs to Specification Description Language (SDL). This paper will not discuss the MSC to SDL transformation. We mention the SDL transformation because by it we achieve the prerequisite of validatability and executability. At each stage of the pipeline the standards writer helps to complete the transformation from one notation to another by adding specifications permitted by the target notation. The ideal is that the target notation can express everything contained in the source notation and more. One of the objectives of this research is to flush out semantic issues in each notation by setting them in relationship to one another. In this way we can create the basis for achieving the ideal. The source notation is translated into the target notation automatically, but, as we shall see, gaps appear in the target notation due to its more concrete nature. The standards writer fills these gaps in, and as a consequence, the standard is progressively given more and more concrete expression. Requirements traceability is present because of the transformation relationships among the notations. In the end the reader is presented with a set of equivalent specifications: very compact, abstract UCMs, less compact, more concrete MSCs, and voluminous, very concrete SDLs.

UCMs provide end-to-end views of concurrent system behavior; structure is optional and, if present, minimal. Unbound UCMs, where there is no allocation of responsibilities to components, are ideally suited to capture requirements analysis. By looking at the unbound UCMs the customer sees the sequence of responsibilities on the functional mainline and on the exception paths. The customer does not care about internal components. Later, after the customer has confirmed the results of the requirements analysis, the designers can bind responsibilities to components, the prerequisite for generating the MSCs.

The UCM notation does not allow the user to specify how components communicate. A path segment between components represents a causality flow. The eventual implementation of the causality flow could be call-return, synchronous or asynchronous messaging or some other form of inter-component communication. There could be a number of exchanges between the components before passage is finally negotiated. Such detail is not germane at the UCM level, but it is so at the MSC level.

A UCM starts as a single path or scenario. It can then be elaborated to express alternatives to the main path. At this point it expresses many scenarios in one artifact. The transformation of a UCM to a set of MSCs is the resolution of the UCM into a set of valid paths: one MSC per valid path embedded in the UCM. The user can then modify the MSC to specify the details of the communication protocol to be used between the components.

In this paper, we will focus on UCMs and MSCs, and on the relationships between them. The core of this paper is taken from [2] in which an end-to-end systematic and traceable progression is defined between requirements level scenario descriptions and hierarchical state machines. The transition between scenario models and hierarchical state machines is discussed in [1].

The rest of the paper is organized as follows. Section 2 introduces the UCM modeling tech-

nique. Section 3 discusses the relationships between UCM and MSC. Section 4 defines a systematic and traceable transition between UCM and MSC. Finally, section 5 concludes.

2. Use Case Maps

UCM (Use Case Map) [4] is a high level scenario modeling technique defined for concurrent and real-time system design. It is based on a simple and expressive visual notation that allows describing scenarios at an abstract level in terms of sequences of responsibilities[1] over a set of components.

The primary objective of the UCM modeling technique is to capture and analyze system behavior at an abstract level; UCM describes scenarios at an abstraction level that is above both inter-component communication and detailed level component behavior. It allows focusing on individual scenario description, scenario interaction, and responsibility allocation, before introducing inter-component communication. As is, UCM models can be viewed as a specification for the modeling of inter-component communication.

UCM also provides important features:

- Superimposition of scenarios on system structure. This enables designers to visualize scenarios in the context of a system structure. It also provides a mechanism by which responsibilities can be allocated to system components.
- Combination of sets of scenarios in a single diagram. This enables designers to express scenarios and scenario interactions in a graphical manner. It also provides a mechanism that can be used by designers to analyze the overall system behavior that emerges from scenario combinations.
- Description of system dynamics both at the component level, where components may be created, destroyed, and moved from one location to another in the system, and at the scenario level, where the a scenario may be dynamically modified as the system evolve. This aspect of UCM is not explicitly addressed in this paper.

In this section, we describe the UCM terminology and notation used in this paper. Readers interested in more details are referred to [4] and [10].

2.1 Basic UCM Notation

A *use case path* represents a path along which scenarios flow in the system. It expresses the sequence of responsibilities that must be executed by system components in order to achieve the overall objective of the system in response to a given triggering event.

In this section, we first describe the basic notations used to describe paths, and then we describe how system components can be introduced in UCM maps.

Basic Path Notation

In Figure 1, the basic elements that compose a use case path are illustrated.

Start Point . The execution of a use case path begins at a start point. A start point is illustrated in UCM by means of a filled circle placed at the origin of a path segment. A start point is defined by means of a set of possible triggering events and an optional precondition.

Responsibilities. As previously mentioned, a use case path describes a sequence of responsibilities that need to be executed by system components in response to a given triggering event. At the UCM modeling level, these responsibilities are high level ones. A responsibility is usu-

1. In UCM, responsibilities are described using informal textual descriptions.

ally defined by means of a responsibility identifier and short textual description that describes in prose the nature of the responsibility. Thus, at this stage, responsibilities remain informal elements of a system model that need to be more precisely defined in later stages of the development process.

Responsibilities are visually illustrated in UCM by means of responsibility identifiers placed along path segments.

Path segment . A path segment expresses an ordered sequence of path segment elements that need to be executed by system components. It is visually illustrated in by means of a curve joining together the sequence of path segment elements. A path segment may be composed of zero or more path segment elements.

A path segment element is either: a *responsibility*, a *waiting place* (see section 2.4), or a *stub* (see section 2.4)

End bar. The execution of a path terminates at an end bar. An end bar is visually illustrated in UCM by means of a thick perpendicular line placed at the end of a path segment. An end bar is defined by means of an optional resulting event and a postcondition. A resulting event is defined by means of a unique identifier and an optional list of output parameters associated with the event.

Path.A complete UCM path may be composed of one or more path segments connected together by means of segment connectors (see definition of segment connectors in section 2.3). The first path segment of a path must start with a start point, and the last segment of a path must terminate with an end bar.

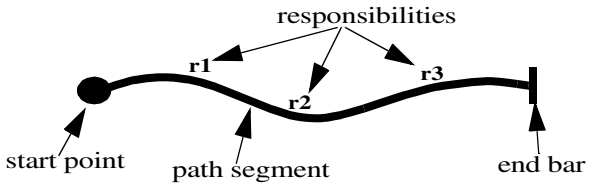


FIGURE 1. A simple UCM path

2.2 Introducing Components in UCM

The UCM modeling technique also allows for the description of paths in the context of the system structure. This is done by superimposing paths on a system structure as illustrated in the right diagram of Figure 2. In UCM, components are visually illustrated using labelled rectangles. At this level, the system structure is only defined as a set of components; inter-component communication is not yet defined. We call such a diagram a *bound use case map*, or simply *bound map*. In bound a map, responsibilities are allocated to components. For example in the bound map of Figure 2, responsibility x is allocated to component B, responsibility y is allocated to component C, and responsibility z is allocated to component D.

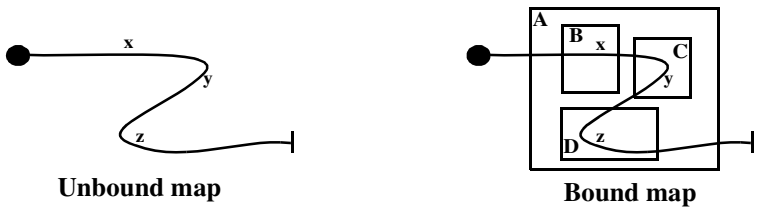


FIGURE 2. Unbound and bound maps

2.3 Path Segment Connectors

In addition to purely sequential paths, as the one illustrated in the previous section, the UCM notation can also be used to described more complex cases that involve concurrent path segments or alternative ones. To describe such cases, the UCM modeling technique uses path segment connectors. A path segment connector is either an AND-fork, an AND-joint, an OR-fork, or an OR-join. These connectors are illustrated in Figure 3. In this figure, each path segment is labelled with a different identifier. The execution of the four path diagrams given in this figure goes from left to right.

FIGURE 3. Path segment connectors

AND-fork. An AND-fork is used to illustrate a point along a path where the execution of a single path segment forks into the execution a set of two or more concurrent path segments. The semantics of the AND-fork given in Figure 3 is as follow. Once the execution of path segment a is completed, then the concurrent execution of path segments b and c may start.

AND-join. An AND-join is used to illustrate a point along a path where several concurrent path segments synchronize together and result in the execution of a single path segment. The semantics of the AND-join given in Figure 3 is as follow. Once the execution of path segment d and e is completed, then (and only then) the execution of path segment f may start.

OR-fork. An OR-forks (exclusive OR) is used to show a point along a path where alternative branches may be followed. Each branch is associated with a distinct path segment. The semantics of the OR-fork given in Figure 3 is as follow. Once the execution of path segment g is completed, then the execution of path segment h or i will be triggered. Thus, the OR-join diagram illustrates two possible paths: one formed by path segments g-h, and one formed by path segments g-i.

OR-join. An OR-join is used to illustrate a point along a path where two or more incoming path segments merge into a single one without requiring any synchronization or interaction between the incoming path segments. The semantics of the OR-join given in Figure 3 is as follow. The execution of either path segment j or k will result in the execution of path segment l. Thus, the OR-join diagram illustrates two possible paths: one formed by path segments j-l, and one formed by path segments k-l.

The AND-fork and AND-join segment connectors can also be generalized to describe more complex path synchronization with n incoming paths and m outgoing paths as illustrated in Figure 4a. The path connectors described above can also be combined to describe more complex paths. Some examples of the type of path constructions that can be described by combining these connectors are illustrated in Figure 4b, c, and d.

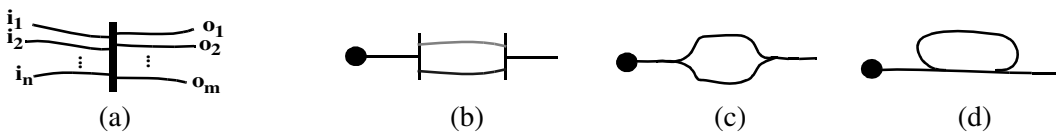


FIGURE 4. Combination of path segment connectors

2.4 Other Path Notations

Two other types of UCM notations are used in this chapter: *waiting place* and *stub*.

Waiting places are used to indicate a point along a path where the progression of the path is blocked until a predefined unblocking event occurs. We identify two different types of waiting places: a regular waiting place, simply called waiting place, and a timed waiting place, called timer. These are illustrated in Figure 5 and described below.

FIGURE 5. Waiting places

A *regular waiting place* identifies a point along the path at which the progression of a path is blocked until a predefined unblocking (or triggering) event occurs. After the unblocking event is received, the progression (execution) of the path may continue. Visually, waiting places are illustrated using filled circles placed along a path. Waiting places are used to illustrate points along paths where interactions with other paths or with the environment of the system occurs. Waiting places can be associated with both synchronous and asynchronous interactions (see section 2.6). A starting point constitutes a special use of waiting place.

A *timer* is a special type of waiting places that will only wait for a certain period of time before continuing on. Thus, the use of timer prevents the occurrence of deadlock. Timers are visually represented by clock-like icon placed along a path. Timers are usually followed by an or-fork connector that illustrates the two alternate paths that could be taken after the timer; one that illustrates the case where the expected unblocking event occurred before the timeout occurred (normal path), and the other that illustrates the case where the timeout event occurred first (time-out path). A timer can also be used along a path without being followed by an OR-fork to introduce time delay on the execution of the path.

The UCM modeling technique also provides a mechanism for path abstraction, called *stub*. A *stub* illustrates part of a path that is abstracted in the context of the map in which it is used. In a UCM model, the expansion of the stub is either described in a separate maps, or remains to be defined later when details will be added to the UCM model. Stubbing constitutes an important mechanism for iterative development. It also reduces the cluttering of models by hiding details that are less important in the context of a given map. An example of path stubbing is given in Figure 6. On the left side of the figure, a detailed path is shown. On the right side of the figure, the stubbed version of the same path is given. We observe that the path segment enclosed in the circle, in the left diagram, has been collapsed into a stub.

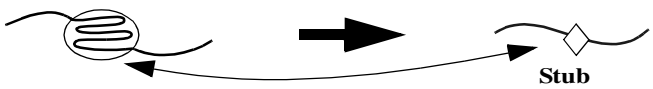


FIGURE 6. Stub

2.5 Related Path Set

We call *related path set* a set of paths that can be triggered from a single starting point. Formally, a related path set is composed of:

- A starting point,
- A set of path segments,
- A set of segment connectors, and
- A set of end bars (each end bar is associated with a distinct terminating path segment).

An abstract example of a related path set is given in Figure 7. This related path set expresses

four different possible paths: A-B, A-C-D-E, A-C-D-F, and A-C-G-(H ||| I)-J[2].

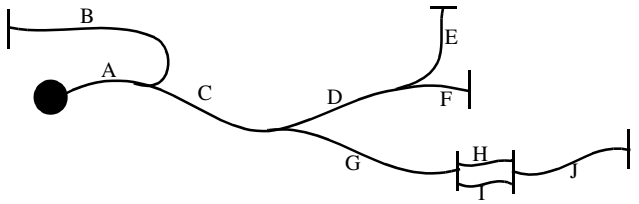


FIGURE 7. A UCM related path set

Because a related path set expresses a set of paths that can be triggered from a single triggering event, it constitutes a cohesive logical entity. The concept of related path set plays a central role in this chapter. It is particularly important in the transition between UCM and MSC.

2.6 Path Interaction Notation

An important aspect of the UCM modeling technique is that allows describing of path interactions. The different UCM notations used to describe path interactions are illustrated in Figure 8.

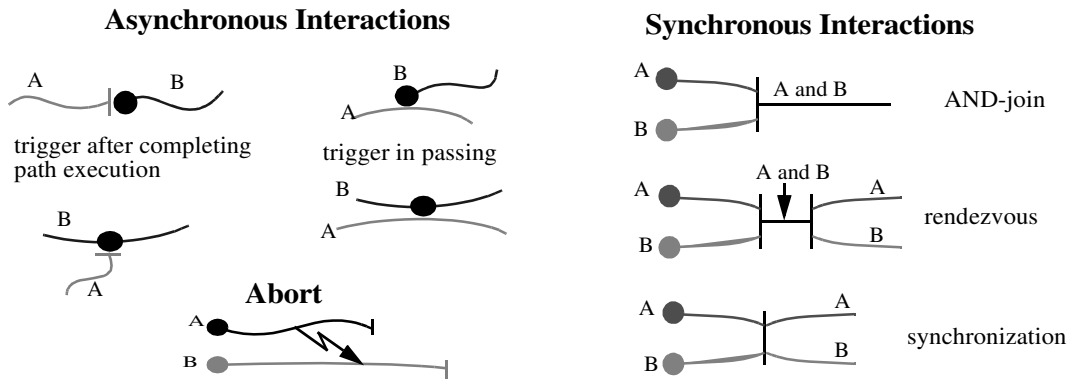


FIGURE 8. Path interaction notation

Asynchronous interactions

Trigger after completing path execution. This type of interaction is used to illustrate cases where the completion of the execution of a path triggers another path that is waiting on a waiting place. The waiting can either be a start point or a waiting place along a path. Both cases are illustrated in Figure 8.

Trigger in passing. This type of interactions is used to illustrate cases where a waiting place, positioned either at the beginning (start point) or along a path, is triggered by another path in an asynchronous manner.

Synchronous interactions

AND-join. This type of interaction is used to illustrate cases where the synchronization of two, or more, paths results in the execution of a single one. As described previously (see section 2.3) this type of interaction can be generalized for N incoming paths resulting in M outgoing ones.

Rendezvous. This type of interactions is used to illustrate cases where two, or more, paths

2. (H ||| I) means that path segments H and I are executed in parallel.

synchronize together to execute a certain path segment (sequence of responsibilities) before returning to the execution of their own respective path.

Synchronization. This type of interactions is used to illustrate cases where two, or more, paths synchronize together and then return to the execution of their own respective path.

Abort

The abort notation is used to describe cases where the execution of a given path, say path A, interrupts the execution of another, say path B. The abort notation is illustrated in Figure 8.

2.7 Composite Use Case Maps

An important aspect of the UCM modeling technique is that a set of paths can be coupled together in more complex diagrams, called *composite use case maps* (or simply *composite maps*). This allows expressing interactions and concurrency between sets of paths that are contained in different related path sets. In composite maps, the interaction between paths is visually expressed using path interaction notation (see section 2.6). This capability of grouping sets of paths into complex composite UCMs constitutes an important aspects of UCM modeling.

Two abstract examples of composite maps are given in Figure 9. In the first example (left side of Figure 9), scenario S1 triggers scenario S2 in passing, and then waits for the completion of scenario S2 before continuing its execution. In the second example (right side of Figure 9), a composite map illustrating more complex inter-scenario relationships between scenarios S3, S4, and S5 is given. The second example illustrates the use of a synchronous interaction and a timer (timed waiting place).

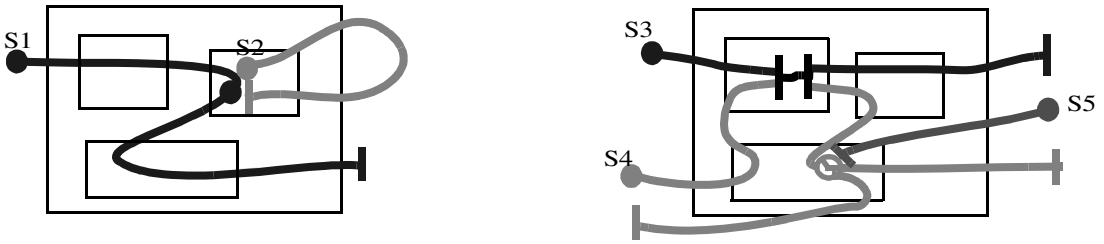


FIGURE 9. Composite use case maps

3. Relationship between MSC and UCM

UCM and MSC are two modeling techniques that focus on scenario description. In order to establish a transition between models produced by these two modeling techniques, we need to analyze how their respective concepts relate to each other. Here is a comparison of their main concepts:

- **Scenario Description.** One of the main conceptual differences between UCM and MSC lies in the abstraction level at which they describe scenarios. UCM describes scenarios in terms of sequences of responsibilities. UCM responsibilities are described at a high level of abstraction by a label and a brief textual description. Such a description abstract away inter-component communication. On the other hand, MSC describes scenarios in terms of sequences of inter-component messages, actions, and methods.
- **Components.** UCM and MSC both allow for component definition. On one hand, UCM defines components in terms of the role they play in a scenario (by means of a short textual description), and the set of responsibilities they must provide in the context of the scenario. UCM does not provide notation to describe the internal logic of components. UCM allows

describing, at a high level of abstraction, what a component does in the context of a scenario, but it does not describe how the component does it.

On the other hand, MSC allows for component description at a lower level of abstraction. MSC describes component behavior mainly in terms of a set of conditions, that may be viewed as states, and a set of messages exchanged between the component and its environment. (A set of actions and methods may also be included.)

- **Related path set.** Both description techniques provide for related path set description. On one hand, UCM describes scenarios by means of UCM related path sets, which allows composing several alternate paths into a single UCM diagram using segment connectors. On the other hand, MSC uses the HMSC notation to describe related path sets. Each HMSC describes a set of related message sequences.

  The main difference between the two modeling techniques lies in the fact that UCM uses a single type of notation to describe both individual scenarios and related path sets, while MSC uses the basic MSC notation to describe individual scenarios, but normally uses the HMSC notation to describe related path sets[3].
- **Scenario interactions.** The capability to explicitly describe scenario interactions is a feature that exists in UCM, but does not exist in MSC. Scenario interactions are captured in UCM models using path interaction notation (see section 2.6).

In conclusion, UCM and MSC are two scenario modeling techniques that express scenario in the context of system components. However, they describe scenarios and components at different levels of abstraction.

Relationship between elements of UCM and MSC Models

Prior to defining the details of the transition between UCM and MSC, it is important to understand the relationship that exists between elements of the UCM and MSC models. A schematic view of this relationship is illustrated in Figure 10.

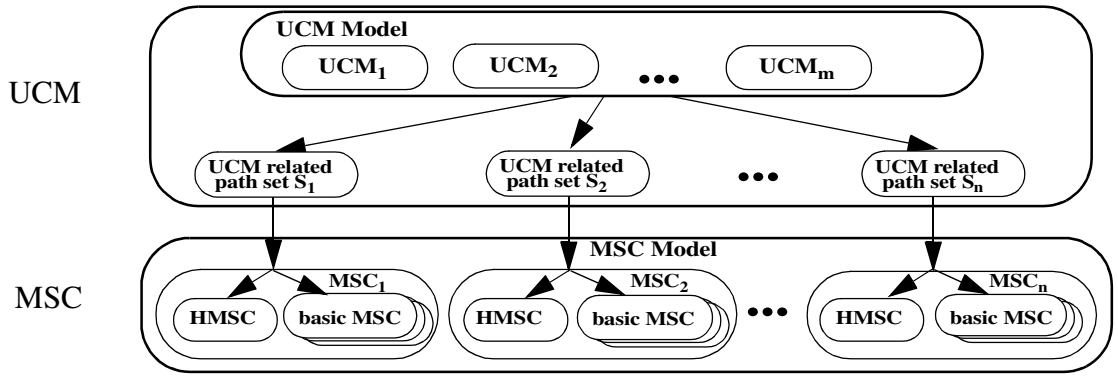


FIGURE 10. Relationship between UCM and MSC

A UCM model is composed of a set of maps, labelled UCM_1, UCM_2, and UCM_m in Figure 10. Each map is composed of a set of paths that each belong to a specific UCM related

3. It should noted here that related path sets can be completely described in basic MSCs using the inline expressions. The two notations are semantically equivalent. A HMSC can be transformed into an equivalent basic MSC, and vice-versa. However, the cumbersomeness of basic MSCs rapidly increases with the number of scenarios contained in a related path set. The resulting basic MSCs may become very difficult to read. For this reason, we recommend using the HMSC notation to describe related path set.

path set, labelled S_1, S_2, and S_n. The top level of the diagram (UCM part) expresses the fact that the set of related path sets that compose the system can be extracted from the set of maps described in a UCM model. Similarly, an MSC model is composed of a set of individual MSCs, labelled MSC_1, MSC_2, and MSC_n in Figure 10.

For the purpose of the transition between UCM and MSC, we establish a one-to-one relationship between UCM related path sets and MSCs. We define a basic MSC for each path segment contained in a UCM related path set, and define a HMSC that captures the UCM path segment connections. The HMSC describes the possible sequences of basic MSC execution.

At a detail level, we establish relationships between the following elements of the two models:

- UCM components and MSC instances
- UCM triggering and resulting events and MSC messages
- UCM preconditions and postconditions and MSC conditions (expressing system states)
- UCM responsibilities and MSC sequences of messages, actions and methods
- UCM path segments and basic MSCs
- UCM path connectors and HMSC alternative and parallel composition constructs (MSC reference connectors)
- UCM stubs and MSC references

It is important to note that when going from UCM to MSC, the explicit interactions between paths expressed in UCM maps are lost. MSC does not allow to explicitly express interaction between scenarios.

4. Transition from UCM to MSC

The objective of the transition between UCM and MSC is to express each of the paths contained in a UCM as a sequence of inter-component messages, component actions, and method calls in an MSC. The transition between UCM and MSC is defined as a three-steps process:

1. **Generation of HMSCs**
   This step consists in generating a HMSC that reflects the path structure of a UCM related path set. In this step, one HMSC is created for each related path set contained in the UCM model. This step can be completely automated, and therefore would not have to be performed by designers.
2. **Generation of Basic MSC Skeletons**
   This step consists in generating a MSC skeleton (see definition of MSC skeleton in section 4.2) for each UCM related path set contained in the UCM model. Like step 1, this step can be completely automated.
3. **Definition of Message Sequences**
   This step consists in expressing each UCM responsibility in terms of a message sequence, or more generally in terms of a sequence of MSC activities[4]. This step is where the designer gets involved.
   In the following sections, we separately describe these three steps.

4. MSC activities include messages, actions, and methods.

4.1 Generation of HMSCs

In this step, we are concerned with the generation of a HMSC that reflects the path segment structure of the UCM related path set. One HMSC is created for each related path set. The generation of a HMSC from a related path set is conducted as follows:

- One MSC reference is created in the HMSC for each path segment contained in the UCM related path set.
- MSC references are connected together using HMSC reference connectors (see section 2.3.2) to reflect the path segment structure of the UCM related path set.
- The precondition and postcondition associated with the UCM related path set are explicitly introduced in the HMSC as MSC conditions.

An example of the generation of an HMSC from a UCM is given in Figure 11. Because system components are not expressed in HMSCs, we have chosen not to illustrate them in the UCM of Figure 11. In this figure, we observe that one MSC reference box is generated for each path segment contained in the related path set, and that the connection between MSC reference reflects the path segment connections of the UCM related path set. If the related path set is composed of a single path segment, then it may be described directly by means of a basic MSC.

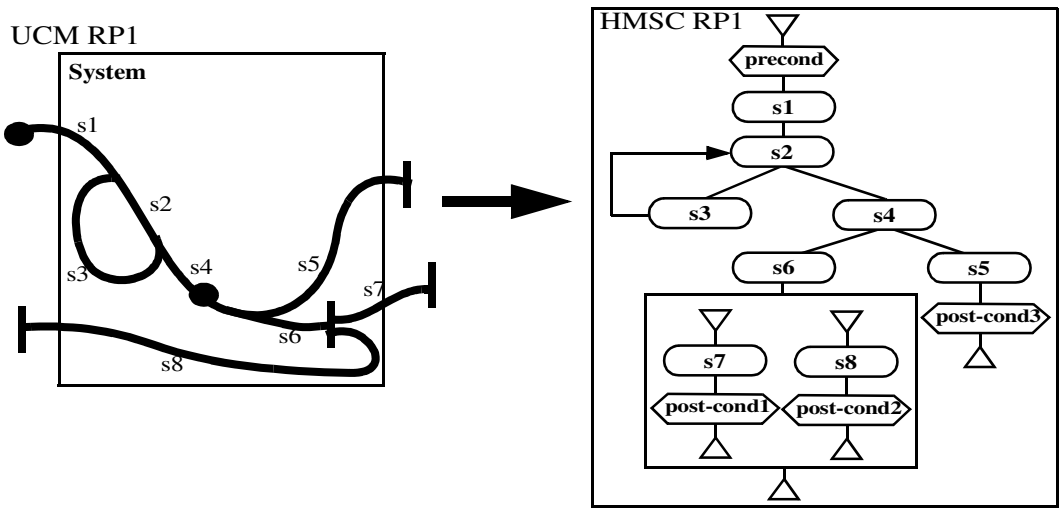


FIGURE 11. Generation of an HMSC from UCMs

In this step, explicit traceability may be maintained by labelling each MSC reference with the identifier of the UCM path segment to which it corresponds. This step could be completely automated in a tool.

4.2 Generation of Basic MSC Skeletons

A *Basic MSC skeleton* is simply a basic MSC that contains no message arrows, and no instance actions or methods. Basic MSC skeletons are only composed of instance timelines and optional initial and terminal conditions. In the transition between UCM and MSC, MSC skeleton are augmented with UCM responsibility identifier placed on the component timeline and with triggering and resulting events message arrows if the basic MSC skeleton corresponds to path segment that is linked to a start point or to an end bar.

The generation of a basic MSC skeleton from a UCM path segment is conducted as follows.

- One MSC instance (component) is created for each component contained in the UCM related path set.

If the path segment contains a start point, then a triggering event message arrow is created and placed at the top of the MSC, just after the system initial state (condition). This arrow is connected on its sender side to the component that generates the triggering event. This component may be either an internal component of the system, if the execution of the path is triggered by an internal component, or to the MSC frame, if the path is triggered by an external component (user) not explicitly represented in the UCM. However, since the receiver of the message is not yet defined at this stage, the message arrow remains unconnected on its receiver side.

Since the execution of a path always requires a triggering event to start, a triggering event message arrow always needs to be defined in a MSC. In cases where a MSC is defined at the highest level by means of a HMSC, the triggering event message arrow is placed in the basic MSC skeleton that corresponds to the first path segment executed in the UCM related path set to which the MSC corresponds.

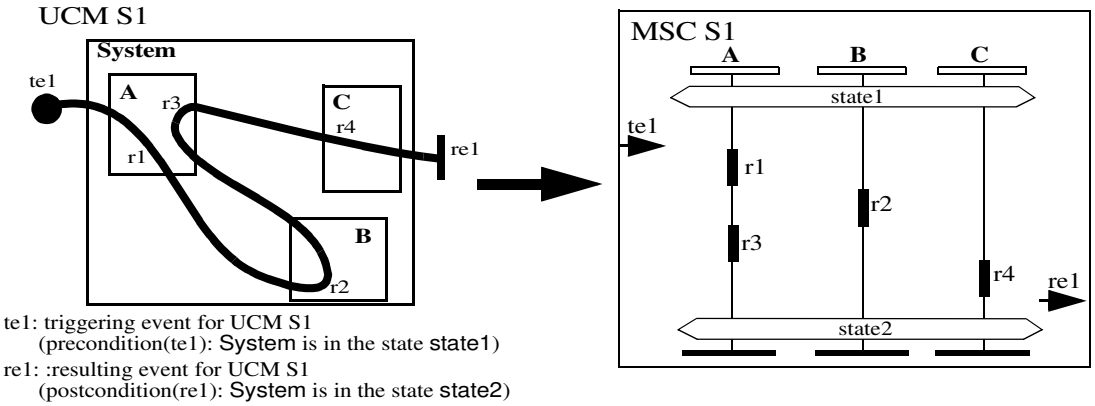
- If the path segment contains a UCM end bar for which a resulting event is specified, then a resulting event message arrow is created and placed just above the final condition (postcondition) in the basic MSC. It is connected on its receiver side to the component to which the resulting event is destined. This component may be either an internal component or the MSC frame in cases where the resulting event goes back to the environment[5]. Since the sender of the resulting event is not yet defined at this stage, the resulting event message arrow remains unconnected on its sender.

  If the path segment contains a UCM end bar, but no resulting event is specified for the end bar, then no resulting event message arrow is created. While the execution of a path always needs a triggering event to start, it may terminate without sending back a resulting event. Thus, resulting event message arrows are not defined in all cases.
- A responsibility is shown using a thick black line segment placed on the instance axis to which it has been allocated[6]. The causal ordering of the responsibility is maintained through the transition. Thus, responsibilities are placed sequentially on the component timelines.

In this step, we maintain explicit traceability by labelling the MSC with the same identifier as the UCM, and by labelling components, state conditions and responsibilities with the same identifiers as the ones used in the UCM.

In Figure 12, an example of the generation of a basic MSC skeleton from a path segment is illustrated. In this figure, we observe that one MSC instance is created for each component identified in the UCM, and that responsibilities have been placed on the different timelines. The sequential ordering of the responsibilities is preserved in the transition from UCM to MSC. We also observe that the triggering event and resulting event arrows have been placed on the frame of the UCM, but have not been connected to any component timeline. The connection of these arrows to particular component timelines will be done in the message sequence definition step (section 4.3). Also, the pre and postconditions that apply to the UCM path are expressed as initial and final conditions, labelled as state1 and state2, in the MSC.

5. The environment is composed of the set of components that are external to the system and that participate in the execution of the scenario.
6. Readers should note that inspite of the fact that the notation used in this paper to represent responsibilities in MSC look similar to the method symbol of MSC'2000, the two are semantically different.



te1: triggering event for UCM S1
(precondition(te1): System is in the state state1)
re1: :resulting event for UCM S1
(postcondition(re1): System is in the state state2)

FIGURE 12. Generation of an MSC skeleton from a UCM path

UCM Stubs in Basic MSC Skeletons

Stubs are represented in basic MSCs using MSC references.

In Figure 13, an example of expressing a UCM stub in a basic MSC skeleton is illustrated. On the top left corner of the figure, the UCM S1 is given. This UCM is mainly composed of two responsibilities, r5 and r8, and a stub, stub1. In the bottom left corner of the figure, the UCM corresponding to Stub1 is given. This stub is composed of two sequential responsibilities, r6 and r7.

On the top right corner of the figure, the basic MSC skeleton, MSC S1, that corresponds to UCM S1 is given. We observe in this basic MSC skeleton that the UCM stub is expressed as a MSC reference. The basic MSC skeleton that corresponds to stub1 is illustrated in the bottom right corner of the figure.

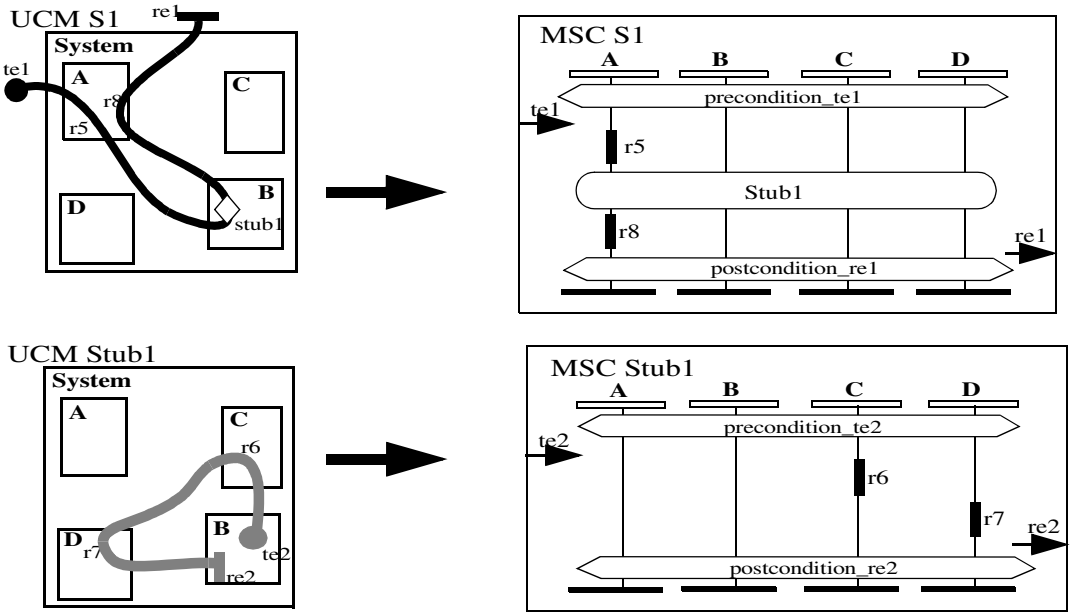


FIGURE 13. Expressing a UCM stub in a basic MSC skeleton

UCM Timers in Basic MSC Skeletons

Timers are represented in basic MSCs using MSC timer notation. If a UCM path segment contains a timer (or timed waiting place), then an MSC timer is introduced in the basic MSC skeleton. Timer message (setTimer, timeout, and clearTimer) arrows are also inserted in the MSC skeletons.

In Figure 14, an example illustrating the generation of a basic MSC skeleton from a UCM containing a timer is given. On the left side of the figure, UCM S1 is given. This UCM is composed of a first path segment that contains a timer and two alternate paths: one triggered by the reception of a timeout event and one triggered by the reception of an unblocking event. On the right of the figure, the basic MSC skeleton that corresponds to the first path segment of the UCM, i.e. the one containing the timer, is given[7]. We observe in this basic MSC skeleton that the UCM timer is expressed by means of an MSC timer. Also, an alternative inline expression is introduced immediately after the MSC timer. This inline expression contains two operands: the first one corresponds to the case where an unblocking message is received (this unblocking message is undefined in the MSC skeleton), and the second one corresponds to the case where a timeout message is received from the timer.

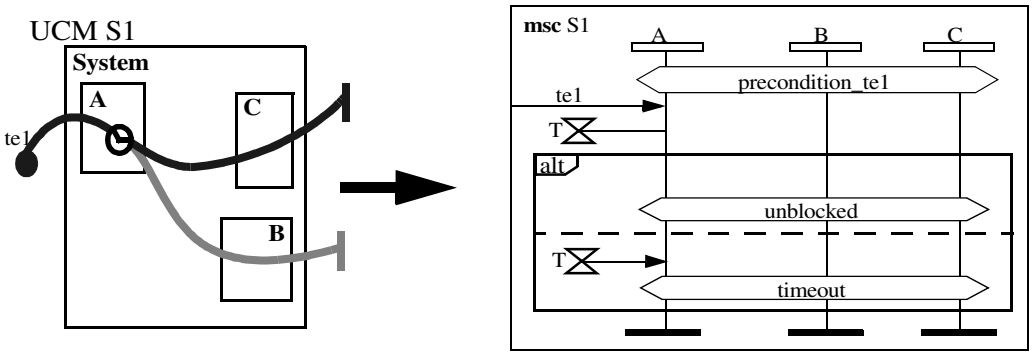


FIGURE 14. Expressing a UCM timer in a basic MSC skeleton

4.3 Definition of Message Sequences

The objective of this step is to express each of the UCM responsibilities placed on the instance axis of the different basic MSC skeletons in terms of a sequence of messages, actions, and methods.

In Figure 15, an example of message sequence definition in an MSC is given. This MSC constitutes a refinement of the basic MSC skeleton given in Figure 12. We observe, in this figure, that the triggering event (message_te1) and resulting event (message_re1) have both been connected to specific instances (components) in the MSC, which are respectively A and C. Also, we observe that every responsibility illustrated in Figure 12 has been expressed as a sequence of MSC elements (messages, actions, and methods).

- Responsibility r1 is expressed as a sequence of two messages (message1 and message2): one going from A to B, and one from B to A.
- Responsibility r2 is expressed as a message (message3) send from A to B, followed by the execution of action a1 by component B, and completed by a message (message4) send by B

7. In this example, we only illustrate the basic MSC skeleton that corresponds to the first segment of the UCM. The generation of an MSC skeleton from the complete UCM related path set requires the use of one HMSC and three basic MSC (one of each path segment).

to C.

- Responsibility r3 is expressed as a sequence of three messages (message5, message6 and message7) send between A and B.
- Responsibility r4 is expressed as a message (message8) sent from B to C, followed by a method call from C to A (message9 and the reply to message9, also named message9 as prescribed in [7]).

These only illustrate some examples of responsibility refinement. We impose no restrictions on the type of refinement that can be applied to responsibilities. The only constraint is that each responsibility must be expressed as a sequence of MSC messages, actions, and methods.

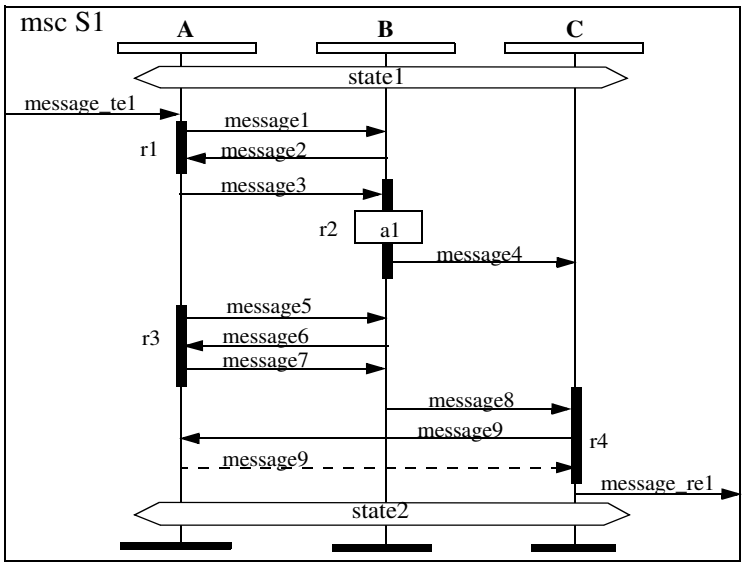


FIGURE 15. Description of message sequence in MSC S1

5. Conclusion

In this paper we introduced the UCM modeling technique and discussed its relationship with MSC. We also defined a transition that allows going from UCM to MSC in a systematic and traceable manner. This approach allows combining the two modelling techniques together in a single process in a FORWARDLY consistent manner. The main advantage of combining UCM and MSC is that it allows describing scenarios at different levels of abstraction. UCM is used for requirements capture and high-level design, while MSC is used to describe scenarios at a more detailed design level. Also, UCM and MSC focus on different aspects of system modeling. UCM focuses on high-level path description, responsibility allocation, and scenario interactions (or more generally on inter-scenario relationships), while MSC focuses on inter-component communication (or messaging). These properties of UCM and MSC make them suitable for use in the pipeline, standards-making process we are attempting to implement.

Our research to date has focussed on forward-looking consistency and traceability. We are now in a position to appreciate that we must also consider backward-looking consistency and traceability. For example, let's consider a male standards writer who makes extensive alterations to an MSC generated from a UCM. Up to this point, these alterations have not violated the initial relationship between the UCM and the MSC. That is, if we were able to generate a partial UCM from this MSC, we would get a partial UCM that could be superimposed perfectly on the original UCM. The standards writer now decides to add an active component to the MSC and to define

messaging between it and the existing components. He makes this decision out of insight not available when the original UCM was created. What does he do to ensure consistency between the UCM and the MSC? Does he go back to the UCM, add the component, regenerate the MSC, reapply the changes and then continue on his way? Does he manually modify teh UCM in a manner he thinks will make it consistent with the current state of the MSC? What if the standard contains a hundred UCMs and a thousand MSCs? What if this situation keeps arising over and over again? The potential for burdensome "overhead" is clear. Moreover, if the standards writer manually alters the UCM to make it consistent with the MSC, will he necessarily obtain the agreement of his colleagues that he did it correctly?

We want the standards writer to be able to make whatever changes he wants to the MSCs generated from teh UCM without worrying about inconsistency and without incurring "overhead". the solution is to regenerate the UCM from the MSCs once all the modifications to the latter have been completed for a particular review cycle. We believe that traceabilty and consistency from the perspective of reducing time to market demands that mappings between notations be done in both a forward and backward direction. We have not yet addressed the backward direction in our research. It is our intention to do so at some point in the future.

We expect exactly the same issue to arise in the relationship between teh SDL generated from a set of MSCs. Designers may make changes to the SDL that will make it inconsistent with the MSCs from which the SDL is derived. Furthermore, the changes to the SDL must be propagated upwards to the UCMs; this will be accomplished by regenerating the UCMs from the MSCs regenerated from the SDL.

The systematic nature of the transition defined in this paper allows maintaining a strong traceability between the two models in a forward direction. This forward traceability and its complementary backwards traceability to be defined in future research should facilitate the eventuaal integration of the two modeling techniques in tools. A set of more detailed and formal traceability relations between UCM and MSC is defined in [2].

Bibliography

[1] F. Bordeleau, J.P. Corriveau, B. Selic. “A Scenario-Based Approach for Hierarchical State Machine Design”, *Proceeding of the 3rd IEEE International Symposium on Object-Oriented Real-time Distributed Computing (ISORC’2000)*. March 15 - 17, 2000, Newport Beach, California. (available at http//www.scs.carleton.ca/~francis/publications/isorc2000.pdf)

[2] F. Bordeleau. *A Systematic and Traceable Progression from Scenario Models to Communicating Hierarchical State Machines*. Ph.D. Thesis, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, 1999. (available at http//www.scs.carleton.ca/~francis/publications/phdthesis.pdf)

[3] F. Bordeleau, R.J.A. Buhr. “UCM-ROOM Modeling: From Use-Case Maps to Communicating State Machines”, *Proceedings of IEEE Conference and Workshop on Engineering of Computer-Based Systems 1997 (ECBS’97)*. March 24-28 1997, Monterey, California. (available at http// www.scs.carleton.ca/~francis/publications/ecbs97.pdf)

[4] R. J. A. Buhr, R. S. Casselman. *Use Case Maps for Object-Oriented Systems.* Prentice Hall, 1996

[5] R.J.A. Buhr. "Use Case Maps for Attributing Behaviour to Architecture", *Fourth International Workshop on Parallel and Distributed Real Time Systems (WPDRTS)*, April 15-16, 1996, Honolulu, Hawaii. (http://www.sce.carleton.ca/ftp/pub/UseCaseMaps/attributing.ps)

[6] J. Hodges, J. Visser. “Accelerating Wireless Intelligent Network Standards through Formal Techniques”, *Proceedings of IEEE Vehicular Technology Conference 1999 (VTC’99). May 16-19, 1999, Houston, Texas.*

[7] ITU (2000). *Message Sequence Charts (MSC’2000)*. Recommendation Z.120. Geneva.

[8] ITU (1992). *Specification and Description Language (SDL’92)*. Recommendation Z.100. Geneva.

[9] IS-771. *WIN (Wireless Intelligent Network) Standard.* (Scheduled for ANSI-4 Revision E)

[10] Use Case Maps web site. http://www.UseCaseMaps.org.