Scenario-based Software Architecture Modeling Using Message Sequence Charts

Gerardo Padilla, Cuauhtémoc Lemus, and Miguel A. Serrano

Centro de Investigación en Matemáticas (CIMAT)
Apdo. Postal 402, Guanajuato, Gto, 36000, MEXICO
(gpadilla, clemola, masv)@cimat.mx

Abstract

Modern approaches in software development propose the use of software architectures to help handle the complexity of large software system developments. An important technique to define, analyze, validate, and evaluate a software architecture is through the use of scenarios. A scenario is a brief description of a single interaction of a stakeholder with a system. Scenario modeling is a well-accepted practice among software architects. However, there is no agreement on how to model scenarios. Message Sequence Charts (MSC) is a graphical-textual notation used to specify scenarios. This notation is standardized by the International Communication Union (ITU) and has a well-defined semantics. In this paper we present an approach for describing architectural scenarios using MSC. The paper focuses in the features proposed by the MSC notation to describe complex behaviors and time constraints. In addition, a brief discussion on the advantages of using MSC to validate architectural scenarios is presented. This validation is performed using an executable interpretation of MSC.

1. Introduction

The software development process is composed of different systematic stages: analysis, design, implementation, testing, and maintenance. Modern approaches, such as the Unified Process [1], propose the use of software architectures to help handle the complexity of large software system developments. The main benefits of a software architecture-based development include:

? The architecture is the earliest document of the development process that brings to perspective critical design decisions that are difficult to change. These decisions have a great impact on the final product.

? The architecture may be the common vehicle of communication among stakeholders of the system (user, client, architect, designer, developer, etc.) supporting tradeoff analysis between system qualities of the system (performance, security, maintainability, reliability, etc.).

There are several definitions of software architecture from practitioners and researchers. In this paper we use the one proposed by the IEEE Std 1471-2000 [2]: "The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution."

An architectural scenario (or scenario) is a brief description of a single interaction of a stakeholder with a system [3]. This concept is similar to the notion of use cases prevalent in the object-oriented community [4].

In order to design a software architecture, an architectural creation process must be defined and established. The fundamental premise of the software architecture creation process is the transformation of a set of functional and quality requirements specifications into a software architectural description. In [3], [5], and [6] software architecture processes are proposed, where the driving force behind architecture description, analysis, and evaluation is the use of scenarios.

Scenarios are the underpinnings in an iterative development method. Scenarios are useful for

1. Requirement elicitation and validation
2. Helping stakeholders understand the architecture
3. Identify flaws and limitations of the architecture
4. Identify architectural views and their representation (process view, logical view, implementation view, deployment view, and use case view) [1]
5. Understand the impact of anticipated changes on the architecture

During the past decade, several notations have been developed in order to specify, verify, and validate communication systems (based on communication protocols). The International Communications Union (ITU), formerly CCITT, standardizes some of these notations. Particularly, the Message Sequence Charts (MSC) [7] is a notation used to specify interactions

among communicating entities. The official name for this notation is ITU-T Recommendation Z.120 (11/99) Message Sequence Charts.

The MSC is a standardized textual-graphical notation that has formal semantics [8]. The MSC has been used widely to capture requirements, to visualize interactions of the system's execution simulations [9], and to verify design models [10]. MSC has been proposed to be part of the UML 2.0 because of its easy understanding and expressiveness [11].

The MSC includes useful issues to describe complex and precise scenarios in the system such as timing and data mechanisms.

In this paper we propose MSCs as a mean to describe architectural scenarios. Additionally, we propose the use of an executable interpretation of MSC to validate the system execution. This process is based on an Abstract Execution Machine. The validation requires two elements: the architectural scenarios and a set of recorded traces from the execution of the system.

The advantage of using an executable interpretation of the notation is presented in the use of automated tools. If a model (defined by a set of diagrams) can be interpreted and mapped to executable elements then the tools can be used to generate code, generate test cases, validate the system, etc.

The paper is organized as follows. Section 2 is an overview of the most important elements of the MSC notation. Section 3 introduces a software architecture modeling example using MSCs. Section 4 describes the validation process using an executable interpretation of MSCs. And finally, Section 5 provides the conclusions and final remarks on the content of this paper.

Related Work

A scenario-based architecture description and evaluation is proposed in [5]. The Software architecture is modeled using Use Case Maps (UCM). In [6] a method to describe architectures is presented; scenarios are used to define the meaning of quality requirements.

The use of UML Sequence Diagram to define architectural scenarios is proposed in [12]. The Architecture Tradeoff Analysis method (ATAM) [13] explains and describes the use of scenarios as a mean to evaluate an architecture description.

2. Message Sequence Chart (MSC)

Message Sequence Chart is a graphical-textual notation used to describe communication interactions among entities. This section presents an overview of some modeling constructors described in the MSC recommendation. The example and figures presented in this section describe operational scenarios for a toaster machine.

Basic Message Sequence Chart

A basic MSC (bMSC) is presented in Figure 1. Usually the bMSC is called just MSC. The MSC depicts a scenario composed of instances (represented by boxes, **User**, **Control**, and **Heating** in Figure 1) and messages (represented by arrows, **start**, **start_ack**, and **hot** in Figure 1). Every instance owns its time axis (vertical line). Messages between instances are shown as arrows connecting the axes.

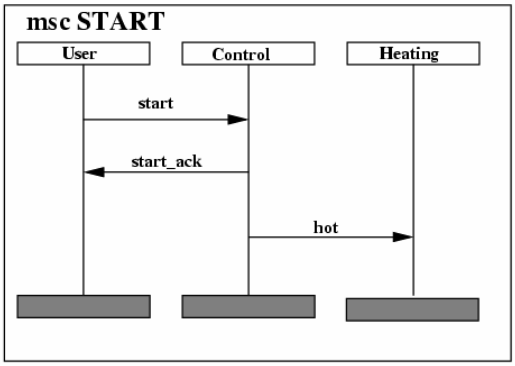

Figure 1. Basic Message Sequence Chart (bMSC).

The communication is one-to-one and asynchronous. There is no explicit information about the communication media. Besides message exchange, a MSC may contain other elements such as internal actions, timer events, conditions, and co-regions [7].

Every message is composed of two events: sending and receiving as shown in Figure 2. Figure 2 is modified from Figure 1 to show explicitly the events of sending and receiving. Notice that this extended notation is not in the MSC recommendation. We proposed this extension as an enhancement to improve the understanding of message events. The sending event of the message **start** is represented by **!start**, and the receiving by **?start**.

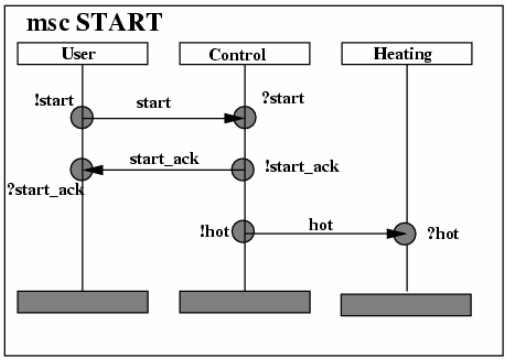

Figure 2. Basic MSC with explicit event representation (non standardized notation).

Semantically, an MSC describes a partially ordered set of events, defined by the order imposed by the time axis of each instance and the relation send-receive of each message. For example, in Figure 2, the time axis order of instance **User** establishes that the event **!start** precedes the **?start_ack** event. The relation send-receive for the **start** message establishes that the **!start** event precedes the **?start_ack** event. An MSC describes a set of traces (a trace is a sequence of events) computed by the partial order of events. The traces described in the MSC presented in Figure 1 are:

Trace 1: start!, start?, start_ack!, start_ack?, hot!, hot?

Trace 2: start!, start?, start_ack!, hot!, start_ack?, hot?

Trace 3: start!, start?, start_ack!, hot!, hot?, start_ack?

This set of traces defines the notion of execution in the MSC that provides the basis for the validation proposed in Section 4.

Structured MSC (sMSC)

Structured MSC is a more expressive MSC. Basically, a bMSC is extended to handle more complex interactions such as parallelism, iteration, and alternative selection. These complex interactions are described using inline expressions. The inline expression is graphically represented with a box containing, in the left upper corner, a word denoting the composition operation (alt, loop, and par). This word may define different operations: parallel (par), loop (loop), and alternative composition (alt). Parallel and Alternative composition requires sections; the sections of an inline expression are separated by a dotted line (there is no restriction in the number of sections). Examples of alternative and parallel inline expressions are presented in Figure 3 and 4, respectively.

Figure 3 shows an example of alternative composition. Two sections are denoted, one section contains the message **msg2**, and the other contains the message **msg3**. In this case, according to the interpretation provided in [7], two sections may occur, but not both. The question about which section should be selected, can be formally interpreted as follows: **msg1** is sent; then either message **msg2** is sent (first section occurs), or message **msg3** is sent (second section occurs).

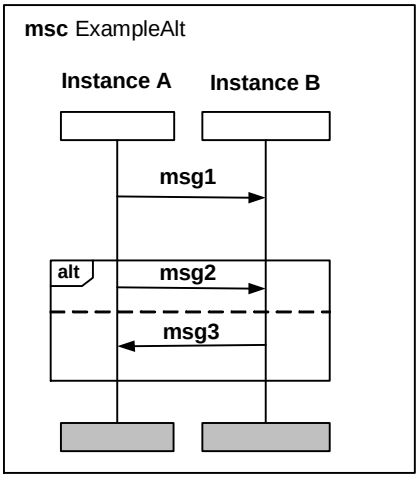

Figure 3. Structured Message Sequence Chart (sMSC) denoting alternative composition.

Figure 4 presents an inline expression describing a parallel composition operation. The interpretation is simple: every section may occur simultaneously. But, outside from the inline expression the order described must be accomplished.

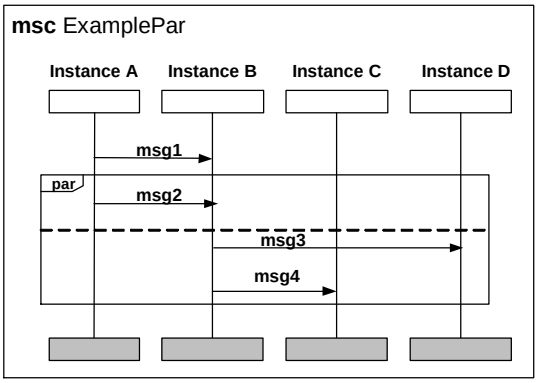

Figure 4. Structured Message Sequence Chart (sMSC) denoting parallel composition.

High-Level MSC (HMSC)

High-level MSCs provide a mean to graphically define how a set of MSCs can be combined. Figure 5 presents an example of a HMSC. A HMSC is a directed graph where each node is either: a start symbol (an inverted triangle), an MSC reference (the round corned box), a connection point (circle), or others elements. The MSC reference is a link to an sMSC, bMSC, or other HMSC.

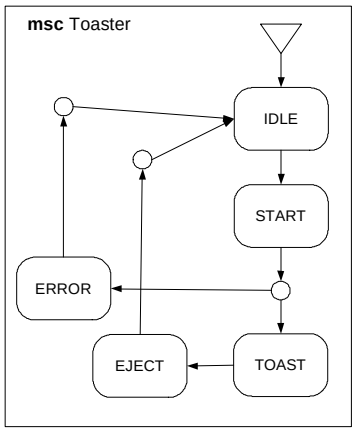

Figure 5. High-Level MSC (HMSC).

The flow lines connect the nodes in the HMSC and they indicate the sequencing that is possible among the nodes in the HMSC. If there is more than one outgoing flow line from a node this indicates an alternative. Figure 6 presents the linked MSCs used in the **msc Toaster** presented in Figure 5.

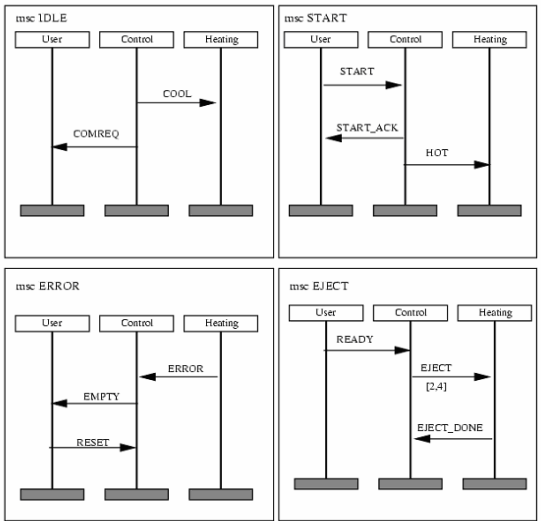

Figure 6. Combining bMSC and HMSC.

MSC Time constraints specification

Timing constraints and measurements are introduced into MSC to support the notion of quantified time for the description of real-time systems with a precise meaning of the sequence of events in time [7]. Time constraints can be specified in order to define the time at which events may occur. According to the recommendation, the time progress (i.e., clocking) is equal for all instances in an MSC. Also, all the clock values are equal, i.e., a global clock is assumed.

There are three main areas where time can be used: as time observations, time constraints, and timer related events. Time observations are relative or absolute measurements between two events. These measurements can be also used to specify time constraints.

Figure 7 shows an example of time constraints (time points). The absolute timing constraints, represent with the "@" symbol, denotes that execution of this MSC must start when the global clock starts, or the occurrence of this scenario restarts the global clock. There is a relative timing constraint in the MSC, the time consumed between the sending of the message **initialize** and the reception of the message **ready** must be 10 ms.

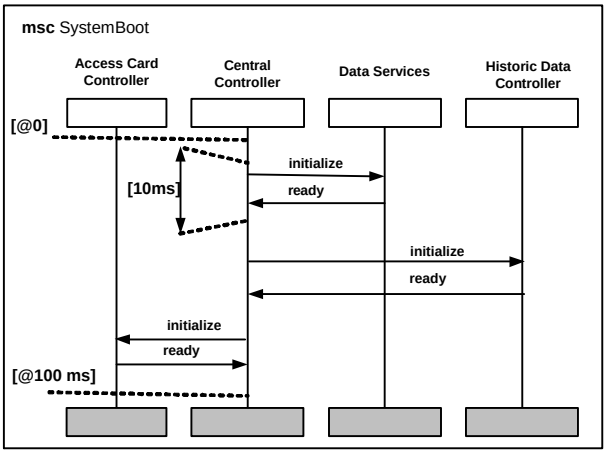

Figure 7. Time constraints example.

Measurements are used to observe the delay between the enabling and occurrence of an event (for relative timing) and to measure the absolute time of the occurrence of an event (for absolute timing) [7]. In order to distinguish between absolute and relative timing different time marks are used ("@" for absolute, and "&" for relative). Figure 8 shows an example of relative and absolute timing. The time consumed between the reception of the message **access_granted** and the sending of the message **save_log** is recorded in the variable **rel1**. Two additional absolute

measurements are performed: the time when first event occurrence inside the MSC is recorded in variable **abs1** and the absolute time for the last event inside MSC is recorded in variable **abs2**.

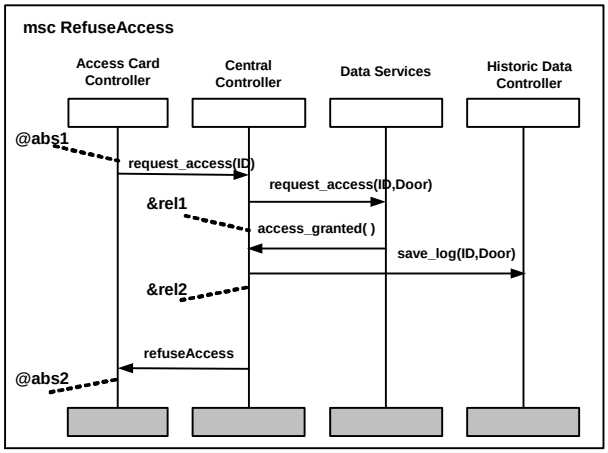

Figure 8. Time observations.

3. Scenario-based Architecture Modeling

In this section we present an example of using MSC to describe architectural scenarios. We enhance the example presenting a model of subsystems using UML notation.

Architecture specification example

In the example below, an alarm system is modeled. The main components of this system are depicted in Figure 9 using an UML Subsystem Diagram to describe the subsystem dependencies [4].

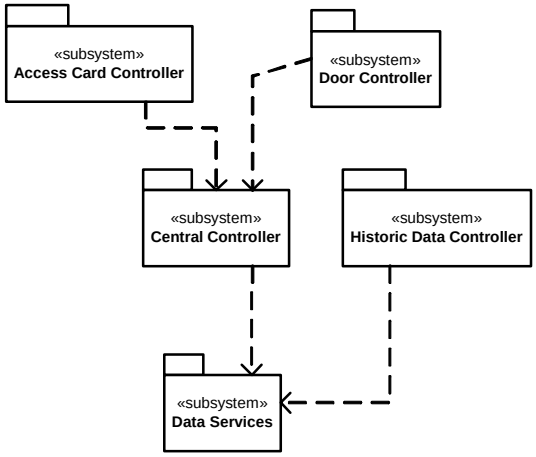

Figure 9. Basic Architectural Subsystems for the Security System.

The system is composed of five subsystems: 1) A central controller which is responsible for the entire system operation, 2) an Access Card Controller, 3) a Door Controller is responsible from controlling the interfaces among the system and the external devices, 4) the Historic Data Controller subsystem keeps the log information, and 5) the Data Services subsystem works as interface with the Data Base system. Some dependencies are drawn explicitly.

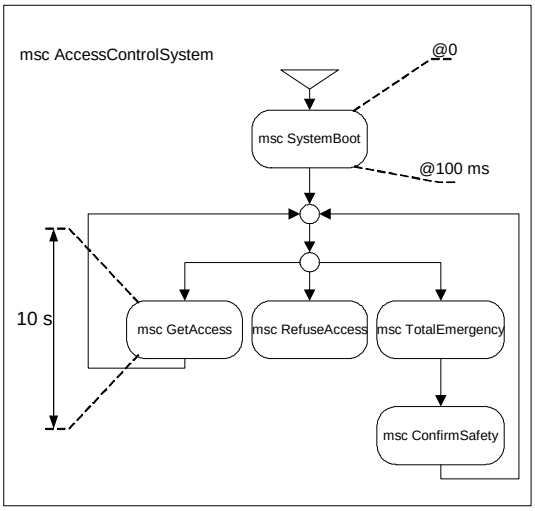

Figure 10. HMSC AccessControlSystem.

The hierarchical organization for a subset of scenarios specified in the system is depicted in Figure 10. The initial scenario, **msc SystemBoot**, describes the set of messages among modules when the system is initialized (Figure 7). The HMSC presented in Figure 10 has two constraints. Both of them indicate a time constraint for the MSC. The meaning of this constraint is as follows: The scenario must fulfill the requirement of time in order to be valid, i.e., the time between the first and the last event inside of it must be less than the time specified (10s for **msc GetAccess**). In addition, **msc SystemBoot** indicates that the time spent to initialize the system must be less than 100 ms from the time it was turned on.

Figure 11 shows the **msc GetAccess**. This scenario describes the interactions among subsystems when an external user request access using the Access Card Controller subsystem. Notice the usage of inline expressions to denote parallelism of the messages **save_log(ID, Door)** and **open(Door)**. Two time constraints are presented in Figure 11. The first one denotes the maximum time (60 milliseconds) between the sending **request_access(ID,Door)** event and the receiving **access_granted( )** event. The second time constraint shown in Figure 11 is similar.

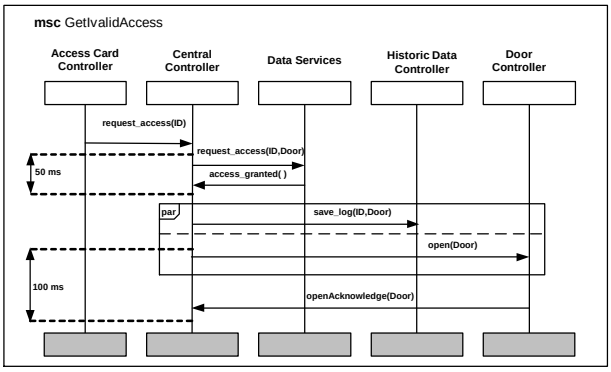

Figure 11. MSC GetAccess with time constraints.

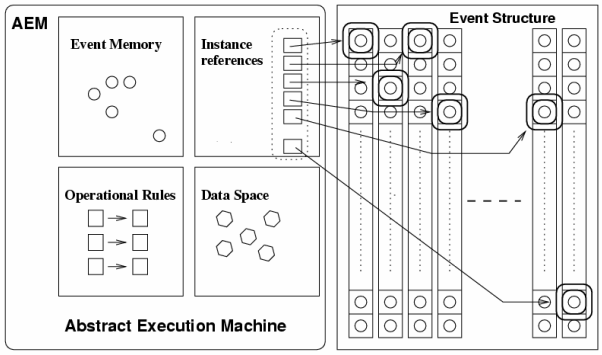

Figure 13. The Abstract Execution Machine.

4. Scenario-based Architecture Validation

The formal semantics of MSC allows the development of tools that automate the visualization, simulation, and in some cases, test generation [14]. Using an executable interpretation (as it was presented in Section 2), it is possible to develop a "validation engine". The validation engine can be built using an Abstract Execution Machine (AEM) [15]. The validation process requires two elements, the MSCs and a set of recorded traces from the system execution.

The Abstract Execution Machine

In order to use the MSC as input to the AEM, we need to provide an interpretation for the MSC (we explain only the basic MSC for the sake of simplicity). Our interpretation does not change the semantic order described in [7].

Event Structure: The Event Structure is a vector of sequences. The elements of the sequences are events. Figure 12 is an example of event structure showing our interpretation of a bMSC.

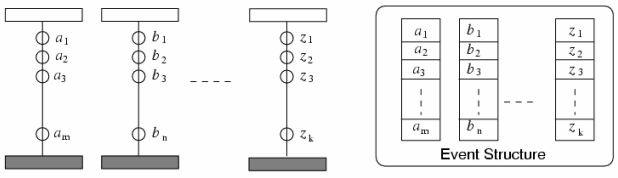

Figure 12. Event Structure.

Abstract Execution Machine: The AEM is composed of three major parts: Instance References, Event Memory Data Space, and Operational Rules. Figure 13 shows a graphical representation of the AEM.

The *Instance References* are responsible of tracking the execution progress, i.e., which events have occurred or are ready to occur. It may be conceived as a container of events. The *Event memory* is responsible of keeping the dynamic information related to the events happening, e.g., the sending of message *m* at time *t*. The *Operational Rules* define the execution progress and availability of events.

Operational Rules: In order to describe the operational rules, two predicates were defined: 1) *enabled*(Event), which determines if an event is able to occur; and 2) the rule *compute_progress*(InstanceReference), which determines if the current instance reference is empty then move to the next instance reference within the sequence.

Having these components, the AEM can function as a generator of traces, i.e., the AEM can provide a set of possible events that may occur (in some domains it is called test oracle). Another application, as mentioned in this paper, is the AEM as a trace acceptor (i.e., the AEM can accept or reject the events provided by a system execution traces and its MSCs).

Validation

The validation process using MSC as input is a well-known practice in the Telecommunications industry [10]. Usually, basic MSC is used only to validate the system. The lack of using sMSC and HMSC is originated by their inherent complexity. We propose the use of the AEM since the AEM can generate the traces from structured MSC.

In order to perform the validation, the final implementation of the system should be able to record the event traces using an event logger mechanism (this mechanism records the instance name, event, and the time

when it happens, we call this tuple an *event stamp*). MSCs and the recorded traces are used as input to the AEM to validate the execution. The validation procedure can be described as follows:

1. The AEM computes the first set of probable events to happen.
2. The AEM uses the first event in the trace recorded to match it.
   a. If the event matches then,
      i. If the event is the end of the event structure then reports a successful validation
      ii. If the event is not the end of the event structure then go to Step 3.
   b. If the event does not match, then the execution did not satisfy the behavior described in the MSC. The validation process reports a failure validation.
3. The event matched is eliminated from the trace recorded. The AEM computes the progress and offers a new set of probable events.
4. Go to Step 2

If we need to validate the time constraints, the AEM can include a time-progress mechanism to compute the time between events. This would be a new operation rule based on predicates defined using the time constraints.

Beyond Time Constraints

In the previous section, we showed how to use MSC to specify architectural scenarios having time constraints. Furthermore, the MSC can be used to describe more issues, such as data manipulation during message interactions, and conditional interactions. These characteristics allow the architect to model complex interactions and, in some cases, non-functional (extra-functional) properties.

5. Conclusions

In this paper, we have presented the use of MSC notation to describe architectural scenarios. The MSC notation provides a set of powerful constructs that can be used to specify complex interactions among system's entities, as well as, time constraints. Even more, the usage of HMSC provides a mean to structure a set of software architecture scenarios. Finally, we have described how to validate system execution traces using an executable interpretation of MSC using an Abstract Execution Machine.

References

[1] I. Jacobson, G. Booch and J. Rumbaugh, *The Unified Development Process*, Addison-Wesley, Madrid, Spain, 1999.

[2] IEEE Std. 1471-2000, *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*, IEEE, Piscataway, N.J., 2000.

[3] Kazman,R., S. Carriere and S. Woods "Toward a Discipline of Scenario-Based Architectural Engineering", *Annals of Software Engineering*, Kluwer Academic Publishers, vol. 9, pp. 5-33, 2000.

[4] OMG Unified Modeling Language Specification Version 1.5, OMG, Needham, MA, 2002.

[5] H.d. Bruin and H.v. Vliet, "Scenario-Based Generation and Evaluation of Software Architectures," *3rd International Conference Generative and Component-Based Software Engineering*, Springer-Verlag, LNCS 2186, 2001, pp. 128-139.

[6] J. Bosh, *Design & Use of Software Architectures*, Pearson Education Limited, London, UK, 2000.

[7] Recommendation ITU-T Z.120 (11/99), *Message Sequence Charts*, International Telecommunication Union, Geneva, 1999.

[8] Recommendation ITU-T Z.120 Annex B (Z.120 Annex B (04/98), *Message Sequence Charts Formal Semantics*, International Telecommunication Union, Geneva , 1998.

[9] *SDT 3.1 Reference Manual*, Telelogic AB, Malmö, Sweden, 1996.

[10] J. Grabowski, D. Hogrefe, I. Nussbaumer and A. Spichiger, "Test Case Specification Based on MSCs and ASN.1", *Seventh SDL Forum*, Elsevier Science, Oslo, Norway, 1995, pp. 307-322.

[11] ad/03-01-02, *Superstructure*, U2 Partners' UML, 2nd revised submission, http://u2-partners.org/uml2-proposals.htm, 2002.

[12] C. Hofmeister, R. Nord and D. Soni, *Applied Software Architecture*, Addison-Wesley, Reading Massachusetts, 2000.

[13] P. Clements, R. Kazman and M. Klein, *Evaluating Software Architectures: Methods and Case Studies*, SEI Series in Software Engineering, Pearson Education Limited, Indianapolis, IN, 2002, p. 323.

[14] S. Mauw, M. Reniers and T. Willemse, "Message Sequence Charts in the software engineering process", *Handbook of Software Engineering and Knowledge Engineering*, World Scientific Publishing Co, 2001, pp. 437-463.

[15] B. Jonsson and G. Padilla, "An Execution Semantics for MSC2000", *10th International SDL Forum*, Springer-Varlag, LNCS 2078, pp. 365-378, 2001.