# An Execution Semantics for MSC-2000*

Bengt Jonsson[1] and Gerardo Padilla[2]

[1] Dept. of Computer Systems, P.O. Box 325, S-751 05 Uppsala, Sweden
bengt@docs.uu.se
[2] Telelogic AB, Uppsala, gpadilla@docs.uu.se

**Abstract.** Message Sequence Charts (MSCs) is a visual notation for expressing requirements on communicating systems. Their expressive power has traditionally been somewhat limited, and additional information is usually needed by tools that manipulate them: for example, to derive test suites. The new standard MSC-2000, developed by ITU-T, extends earlier versions by constructs for data and high-level control, so that it may be possible to derive test sequences directly from MSC requirements, without the need for additional information. Motivated by this, we present an execution semantics for a significant part of the MSC-2000 standard. The semantics has the form of an Abstract Execution Machine, which can either accept or generate sequences of events that are consistent with a given MSC. In the former case, the Abstract Execution Machine can be used as a test oracle, in the latter as a test sequence generator.

## 1 Introduction

Message Sequence Charts (MSCs) is a graphical notation for description and specification of interaction between entities of a communicating system. MSCs may be used for requirement specification, interface specification, simulation and validation, test case specification, and documentation. The main use of MSCs has been as abstract description of the communication behavior of communicating systems, while omitting other aspects of a system's behavior. Used in this way, MSCs do not, by themselves, contain enough information for the generation of test sequences, or for attempting to synthesize a system design. Tools for test generation, such as Autolink or TestComposer [17], and TGV [6] derive test sequences from the combination of a test purpose specification in MSC and a design model in SDL.

One of the motivations for our work is to investigate whether and how MSCs can be used to describe requirements more completely, in a way that allows test sequences to be generated without involving other information about the system (such as in an SDL model). MSCs must then be able to express many aspects of a system that can be represented in, e.g., an SDL model. The new standard

* Supported in part by Telelogic AB, and by NUTEK through the ASTEC Competence Center. This work was carried out while Gerardo Padilla was at Telelogic AB, Uppsala

R. Reed and J. Reed (Eds.): SDL 2001, LNCS 2078, pp. 365–378, 2001.

MSC-2000 [11], developed by ITU-T, extends the earlier standard MSC-96 [10] by constructs for data and high-level control, so that this may be possible.

In many approaches to test generation from MSCs (such as [7,18]), an MSC test purpose is viewed as specifying a set of acceptable sequences of events of the implementation. In the approaches of Autolink, TestComposer, or TGV, the test purpose is often incomplete (missing parameters and data values), and the missing information is supplied by matching it with an SDL model. There are also other approaches to testing, see [4,14,15], where a requirement is translated to a test oracle, which monitors test execution and reports deviations from the original requirements.

Thus, part of the test generation in the above approaches consists in viewing an MSC as an acceptor or generator of sequences of events. It is not difficult to understand what sequences are represented by a simple basic MSC. However, this is less obvious for MSCs following the newer standard MSC-2000, which contains advanced control structures and treatment of data. When trying to understand how MSC-2000 can be used for testing, we found a need for an understandable and implementable description of what sequences of events are represented by an MSC. This motivates our development of a semantics for a part of MSC-2000, which views an MSC as a (structured) abstract machine, whose executions generate or accept sequences of events in the same way as runs of a finite automaton generate or accept strings.

In this paper, we present the major aspects of our execution semantics for MSC-2000. Our aim has been to define a semantics, which in a straight-forward way explains how an MSC "executes", defined in terms of its graphical structure. The semantics derives from each MSC a state machine, which maintains the "current state" of the MSC. Based on the current state of the MSC, transitions can be performed, which are equipped with labels that explain what action is currently performed, and which result in new states of the MSC. The possible transitions are defined through rules for how an MSC may transform between global states while performing events (such as transmitting or receiving messages). We have attempted to make these rules close to an intuition obtained from the graphical presentation of MSCs. Our rules thus provide an operational semantics for the MSC, which can be presented and understood in rather simple terms and be implemented quite directly.

As a contribution, this paper thus presents an operational semantics for MSCs, which is presented in a direct manner by showing how an MSC may "execute" by changing its global state, without translation to an immediate formalism. Each construct of MSC-2000 that we consider is defined by one or two rules. In particular, we present a simple operational semantics for high-level control constructs (such as inline expressions) and data aspects of the MSC-2000 standard [5]. A main problem here is that inline expressions impose control structures that are global to several instances, which otherwise execute asynchronously. In the execution semantics, this "global control" is represented by global execution steps where participating instances are synchronized.