Timed Extensions for SDL*

Marius Bozga[1], Susanne Graf[1], Laurent Mounier[1], Iulian Ober[2], Jean-Luc Roux[2], and Daniel Vincent[3]

[1] VERIMAG, Centre Equation, 2 avenue de Vignate, F-38610 Gières
Marius.Bozga@imag.fr
[2] Telelogic Technologies, 150 rue Nicolas Vauquelin, F-31106 Toulouse Cedex
Iulian.Ober@telelogic.com
[3] France-Telecom R&D, DTL, 2 avenue Pierre Marzin, F-22307 Lannion Cedex
Daniel.Vincent@rd.francetelecom.fr

Abstract. In this paper we propose some extensions necessary to enable the specification and description language SDL to become a more appropriate formalism for the design of real-time and embedded systems. The extensions we envisage concern both roles of SDL: first, in order to make SDL a better real-time *specification* language, allowing to correctly simulate and verify real-time specifications, we propose a set of *annotations* to express in a flexible way assumptions and assertions on timing issues such as execution durations, communication delays, or periodicity of external inputs; second, in order to make SDL a better real-time *design* language, several useful real-time programming concepts are added. In particular we propose to extend the basic SDL timer mechanism by introducing new primitives such as cyclic timers, interruptive timers, and access to timer value. All these extensions rely on a clear and powerful time semantics for SDL, which extends the current one, and which is based on *timed automata with urgencies*.

Keywords: SDL, time semantics, timed automata, urgencies.

1 Introduction

The ITU–T Specification and Description Language (SDL, [10]) is increasingly used in the development of real-time and embedded systems. For example many recent telecommunication protocols (such as RMTP-II [18] or PGM [17]) integrate such real-time features in their architecture, and these non-functional aspects are essential in the expected behaviour of the application. This kind of system imposes particular constraints on the development language, and SDL is a suitable choice in many respects: it is formal, it is supported by powerful development environments integrating advanced facilities (such as simulation, model checking, test generation, code generation, etc.), and thus it can cover several phases of the software development, ranging from analysis to implementation and on-target deployment.

* This work is supported by the INTERVAL IST-11557 European project on timed extensions for SDL, MSC and TTCN.

R. Reed and J. Reed (Eds.): SDL 2001, LNCS 2078, pp. 223–241, 2001.

It appears however that several important needs for a real-time systems developer are not covered by SDL. These problems range from pure *programming issues* (such as the lack of useful primitives commonly available in real-time operating systems) to *specification issues*, (such as the difficulty to describe in a appropriate way the assumptions under which the system is supposed to be executed). Clearly, the needs are not the same for both uses of the language, and, in many cases, the *programming side* has been given priority in the supporting tools to the detriment of the *specification side*.

Several proposals already exist to extend SDL with real-time features. We can mention for example the work carried out on performance evaluation [8,13,15,12], on schedulability analysis [4], or on real-time requirements [11]. In this paper we are more concerned with the use of SDL as a specification language for real-time systems and its application for formal validation. In particular, one of the important questions we address is what kind of real-time features should be modeled in SDL and at which level of abstraction.

The simplest use of time which is frequent in communication protocols, is the use of timeouts (whose value is often meaningless) in order to avoid infinite waiting. The time semantics of SDL, together with the fact that timeouts are notified via a signal in the message queue of the process, corresponds exactly to this use: no guarantee can be given when the signal arrives and is dealt with, but it is after some finite time. Nevertheless, when SDL is used as a programming language, it is often done with much more restricted assumptions on the possible time behaviour in mind, and, if they are correct, the implemented system will behave as expected. As such assumptions are not (and cannot be) expressed explicitly, the specification cannot be validated: the verification using the standard SDL progress of time may invalidate even apparently time independent safety properties.

A typical workaround used for obtaining a convenient result at simulation time consists of using on one the hand timers to force minimal waiting, and, on the other hand, a very restricted interpretation of time progress, allowing it only when the system is not active. This "synchrony hypothesis" is in general as unrealistic as the standard semantics. The correct assumption would be that certain tasks will be executed timely, whereas for others this cannot be guaranteed and the correctness of the system must be verified even if they take longer than expected.

We propose a solution to reconcile these two extreme choices that relies on a more flexible time semantics for SDL, based on timed automata with urgencies [5]. In particular, urgencies give a very abstract means to express *assumptions* on the environment and on the underlying execution system, such as action durations, communication delays, or time constraints on external inputs. From the user point of view, all these "non functional" extensions are offered in a uniform way by means of *annotations* on the SDL specification.

The propositions presented in this paper are the results of the INTERVAL IST project and preliminary work of its partners. The aim of INTERVAL is to take into account real-time requirements during the whole development process