Using UML for Implementation Design of SDL Systems

Jacqueline Floch[1], Richard Sanders[1], Ulrik Johansen[1], and Rolv Bræk[2]

[1] SINTEF Telecom and Informatics, Trondheim, Norway
{Jacqueline.Floch, Richard.Sanders, Ulrik.Johansen}@informatics.sintef.no
[2] Department of Telematics,
Norwegian University of Science and Technology, Trondheim, Norway
Rolv.Braek@item.ntnu.no

Abstract. The purpose of Implementation Design is to bridge the gap between an abstract system, e.g. in SDL, and its implementation in hardware and software. Expressing the Implementation Design decisions is a general challenge in systems engineering. Notations have been defined to describe abstract models and implementations, but little work has been done to define a notation for Implementation Design. In this paper, we discuss UML solutions and extend them. Our work is intended as a contribution to a common SDL approach, and an inspiration to ITU-T Study Group 10 question 11 on deployment and configuration language DCL. We also present how the ProgGen tool was extended in order to generate code controlled by the Implementation Design model.

1 Introduction

There exist several differences between SDL [1] systems and real systems. While an SDL system consists of abstract entities such as processes and channels, a real system is composed of concrete entities such as computers, software processes, buses, and communication networks. [2] discusses how real systems differ from SDL. There are:

- *fundamental* differences originating from the nature of real components and their "imperfections". They encompass processing time, errors developed by physical components and noise, physical distribution and limitation of resources.
- *conceptual* differences originating from the way that components function. They encompass concurrency, communication modes (e.g. stream vs. message), synchronisation, and data abstractions.

When realising an SDL system, one must decide how to implement the abstract functions. These decisions and the concrete system should be documented. In Open Distributed Processing [3] parlance this means describing the Engineering Viewpoint and the Technology Viewpoint, while SDL contributes to the Computational and Information Viewpoint.

R. Reed and J. Reed (Eds.): SDL 2001, LNCS 2078, pp. 90–107, 2001.

The term *deployment* has recently been introduced to denote the Engineering and the Technology Viewpoints [4]. We prefer the more general term *Implementation Design*. While the deployment model describes a particular configuration of run-time processing elements and the software components that live on them, the Implementation Design additionally describes the implementation principles in terms of generic types that can be instantiated in a given deployment. This enables reuse of implementation principles. When applying an Implementation Design model to an SDL system, a concrete system will result.

In addition to defining implementation principles, there is a need to define specific configurations, including aspects such as priorities, capacity parameters and dynamic reconfiguration. These issues are the specific concern of configuration languages, and are not fully treated in this paper.

Although notations have been defined to describe functionality (for example SDL, UML) and implementation (for example C++, Java), little work has been done to define a notation for describing Implementation Design. Some of the few attempts hitherto have been:

- SOON [2], which represents hardware and software structures and the mappings from the SDL entities to the concrete system components. Although we have found it to be very useful, it suffers from lack of tool support. Moreover, SOON lacks support for describing other non-functional properties of the system, such as information that is needed during code generation and installation.
- UML Implementation Diagrams [4], which describe the system software structure and the deployment of software components on a hardware platform. These solutions, although as yet immature and incomplete, have gained our attention.

In this paper, we discuss how appropriate the UML solutions are, and suggest extensions to them. We believe that SDL users would benefit from a common approach to Implementation Design. This paper is intended as a contribution to such an approach and an inspiration to the ITU-T Study Group 10 question on deployment and configuration language DCL [5], to which other projects also contribute [6,7].

In the following, we first recall the main purposes of Implementation Design. Then we describe how UML basic concepts and extension mechanisms can be used for Implementation Design. Examples based on the Access Control system presented in [2,8] are given. Finally we illustrate, using ProgGen [9], how the Implementation Design can be used to control code generation.

2 Implementation Design

Every system development must bridge the gap between the abstract and the concrete system [10]. The Implementation Design model is an important document for the system developers, and for tools such as code generators, configuration and building tools. The model should describe implementation aspects of the concrete system, and relate it to the abstract SDL system. It should model: